ESCUELA POLITÉCNICA DEL EJÉRCITO

Departamento de Ciencias de la Computación
Carrera de Ingeniería en Sistemas e Informática

Análisis de Patrones de Software y su aplicación en un Framework de Desarrollo utilizando plataforma .net

Previa a la obtención del Título de:

INGENIERO EN SISTEMAS E INFORMÁTICA

POR: **IVÁN AURELIO FERNÁNDEZ SILVA**
JUAN GABRIEL JALIL MORENO

SANGOLQUÍ, 07 de marzo de 2007

CERTIFICACIÓN

Certifico que el presente trabajo fue realizado en su totalidad por los Sres. IVÁN AURELIO FERNÁNDEZ SILVA y JUAN GABRIEL JALIL MORENO como requerimiento parcial a la obtención del título de INGENIEROS DE SISTEMAS E INFORMÁTICA

_____

Fecha

_____

Ing. Rolando Reyes

DEDICATORIA

Dedicada a mi familia, mi novia y mis amigos, espero que algo de lo que aquí hay, les ayude en algún momento para seguir su camino.

Iván Fernández

DEDICATORIA

Este trabajo es dedicado a todos esos años en los que junto con amigos y colaboradores continuamente se trabajó para sacarle cada día mas provecho a las oportunidades que tuvimos. A todos los que no nos quisimos quedar solo con lo aprendido sino ver que pueden existir otras formas de hacer girar la rueda.

Juan G. Jalil

AGRADECIMIENTOS

A mi familia y amigos, por siempre estar a mi lado y apoyarme cuando lo necesito, sin dejarme caer y dándome el soporte para seguir adelante. Muchas gracias a todos quienes han estado a mi lado durante toda mi carrera, las malas noches, los problemas y los buenos momentos, por que sin todos ellos no hubiera llegado tan lejos.

Iván Fernández

AGRADECIMIENTOS

A quienes fueron mi guía y un “primer impulso” para lograr lo que he conseguido hasta ahora: mis padres, quienes me dieron su apoyo incondicional durante ya tantos años. A todas esas personas, amigos y familia que confiaron en mí a todo momento y a mi hijo Samuel que sacrificó junto a mí muchos días de sol.

Juan G. Jalil

Tabla de Contenidos

Listado de tablas

Listado de cuadros

Listado de figuras

Listado de anexos

Listado de bloques de código

Resumen

Los Patrones de Software fueron concebidos como una forma para solucionar problemas comunes al proceso de desarrollo y diseño de software, éstos fueron pensados de forma que definan el diseño e interacción de los objetos, además de proveer una plataforma de comunicación que sea elegante y reutilizable. Un patrón de diseño no es algo terminado, es una descripción o una referencia de cómo se debe resolver un problema que puede ser utilizado en circunstancias muy variadas.

En este documento se habla acerca de los Patrones de Software como elementos para ayudar al desarrollo de aplicaciones y componentes que sean más flexibles, mantenibles y reutilizables. El objetivo es mostrar a un equipo de desarrollo de software elementos que apoyen a la construcción de una aplicación empresarial. Los temas que son tratados se refieren directamente al desarrollo y codificación de una aplicación. Usualmente en las metodologías de desarrollo de software se trata sobre el proceso de desarrollo, la documentación, la realización de especificaciones de requerimientos, en esta tesis el enfoque será hacia como codificar una clase, una interfase o un componente para que éste sea extensible, reutilizable o mantenible con más facilidad.

El documento empieza con una descripción teórica de UML, los frameworks, la tecnología .net de Microsoft utilizada para el desarrollo, las aplicaciones distribuidas y la arquitectura de aplicaciones orientadas a servicios. Esto servirá

al lector para obtener una idea global del enfoque de la tesis y de la tecnología utilizada en los siguientes capítulos.

En el capítulo tres se realiza un estudio de los patrones de software más comunes, realizando una clasificación de los patrones de software más comunes y se describe cada uno de ellos. Se concluye con un análisis de los patrones para realizar la selección de los que son útiles para cumplir con los objetivos de la tesis.

En el capítulo cuatro se realiza la definición del framework de desarrollo, explicando su propósito y requerimientos. Luego se hablará sobre el framework planteado explicando la estructura de los proyectos que lo componen y la intención de cada una de sus partes, mostrando también los patrones utilizados en cada uno de los bloques o capas del framework.

En el capítulo cinco se guiará al lector paso a paso por la construcción de una aplicación empresarial “prototipo” que utilice el framework propuesto en el capítulo cuatro. Una aplicación empresarial está compuesta por un gran número de componentes que deben interactuar entre ellos para dar a la aplicación la funcionalidad requerida. Dentro de la aplicación estos componentes desarrollados deberían ser lo suficientemente flexibles como para ser modificados sin que esto afecte al resto de la aplicación.

CAPÍTULO I

1. INTRODUCCIÓN

1.1. Antecedentes

La mayor parte de metodologías de desarrollo de software se enfocan en realizar buenas especificaciones de requerimientos, documentación, diagramas de clases y bases de datos, así como definir procesos ordenados de desarrollo, pero no existen estándares de cómo se debe proceder con la implementación de lo que esos diagramas describen, las metodologías no ayudan a los desarrolladores a realizar la implementación de una clase que un diagrama describe, ni tampoco como los componentes de un sistema deben interactuar entre ellos.

Cuando se diseña un sistema empresarial, usualmente no existe un “arquitecto” de software que muestre a los desarrolladores el camino a seguir para lograr una implementación una arquitectura escalable y robusta, la mayoría de veces solamente se implementa el sistema utilizando los diagramas generados en su documentación, no se hace un diseño adicional de los componentes descritos en los diagramas ni de la forma en la que interactúan para que en el futuro sean reutilizables y escalables.

Durante todos los ciclos del desarrollo de una aplicación se necesitan soluciones a problemas comunes y es precisamente en esta parte donde un patrón puede brindar soluciones ya probadas a estos problemas; esto ayuda a

que el equipo de desarrollo se concentre más en realizar tareas para mejoras de la aplicación y no en resolver los problemas comunes relacionados con el diseño y mantenimiento de las aplicaciones.

Otro gran problema del desarrollo de aplicaciones viene en la fase de mantenimiento, en donde ciertas veces el equipo de desarrollo original no siempre está disponible para realizar la tarea de mantenimiento y cuando algo así ocurre el nuevo equipo se ve rodeado por muchos problemas al realizar los cambios requeridos, aunque exista una completa documentación.

1.2. Justificación

Los Patrones de Software toman un enfoque diferente al mostrar al desarrollador la manera en la que se debe realizar la implementación, extendiendo el alcance de las metodologías de desarrollo.

Los Patrones de Software hacen que un sistema sea mucho más mantenible, bajando el tiempo de desarrollo que una empresa debe cubrir en esta fase, además con un software cuya codificación sea más ordenada se logra minimizar la dependencia de una empresa a sus desarrolladores. Los patrones pueden brindar a las aplicaciones estándares de diseño y arquitectura resultando en aplicaciones más escalables, compatibles con otros sistemas y fáciles de mantener.

Existen otras ventajas relacionadas a la comunicación y entendimiento entre los diseñadores, desarrolladores y gente de mantenimiento, al tener una terminología de trabajo común.

1.3. Importancia

El objetivo de los Patrones de Software es mostrar a los desarrolladores las mejores prácticas que les permitan mejorar la calidad de un sistema, implementando soluciones flexibles y escalables, incrementando la facilidad de adaptar el software a los cambios de especificación e incrementando su posible reutilización.

Utilizando Patrones de Software se puede obtener mejores resultados, al combinar el conocimiento que ha sido recogido de la experiencia de desarrolladores a nivel mundial, logrando por ejemplo realizar cambios de forma más rápida, precisa y con el menor impacto en la estructura del software, disminuyendo los costos de mantenimiento y desarrollo de un software.

De la experiencia que se puede lograr al utilizar patrones de software, es posible recopilar los resultados en un framework que puede servir a la empresa para el desarrollo de múltiples soluciones en un menor tiempo y con una mayor calidad.

1.4. Objetivos

1.4.1. Objetivo General

Analizar los Patrones de Software y su aplicación en un framework para el desarrollo de aplicaciones distribuidas utilizando plataforma .net de Microsoft.

1.4.2. Objetivos Específicos

- Estudiar los Patrones de Software más comunes.
- Evaluar los patrones de software estudiados y seleccionar los aplicables para la construcción del framework.
- Definir un framework basado en Patrones de Software que utilice la tecnología .net de Microsoft.
- Desarrollar una guía para la implantación del framework definido.
- Desarrollar y documentar una aplicación prototipo que utilice la guía desarrollada.

1.5. Alcance

Este proyecto tiene como punto de inicio la investigación de los Patrones de Software más conocidos en la actualidad. Esta información servirá de base para la definición de un framework que recopile los Patrones de Software y las mejores prácticas aceptadas por comunidades, grupos de usuarios y desarrolladores a nivel mundial.

Con el estudio de los Patrones de Software se podrá evaluar la aplicabilidad de cada patrón en la tecnología .net de Microsoft y cuáles de ellos serán utilizables y servirán de base para el desarrollo de un framework que definirá una serie de procesos estables y estandarizados para el desarrollo de aplicaciones empresariales.

Se desarrollará una guía que documenta todos los pasos, tareas y estándares que un desarrollador debe seguir para la construcción e implementación de una aplicación empresarial basada en el framework sugerido. La guía estará compuesta por manuales de arquitectura, descripción general de los patrones implementados en los diferentes niveles de la aplicación, diseño de componentes empresariales y estándares de codificación de software.

Junto con la guía se entregará una aplicación prototipo en la que se mostrarán las principales funcionalidades del framework para que las empresas de desarrollo la utilicen como base en sus proyectos.

CAPÍTULO II

2. MARCO TEÓRICO

2.1. Patrones de Software

El concepto de patrones de software inicialmente fue creado por Christopher Alexander, un arquitecto, quien utilizó patrones para diseñar y construir tanto edificios como ciudades. Sus patrones se enfocaban en definir estructuras de entradas, jardines y caminos que cumplieran con cierto tipo de condiciones, para de esta forma tener formas predefinidas para solucionar situaciones comunes.

Alexander se refería a los patrones como una regla que es comprendida por tres partes, un problema, un contexto y una solución. De esta forma un patrón es un elemento, que permite la utilización de una misma construcción espacial, en el caso de la ingeniería civil, para solucionar un problema, siempre que el contexto lo permita.

Si bien los patrones de Alexander son orientados a la ingeniería civil, son aplicables también al desarrollo de software, en software los patrones facilitan la reutilización del diseño y de la arquitectura, capturando las estructuras estáticas y dinámicas de colaboración de soluciones exitosas a problemas que surgen al construir aplicaciones.

En el año de 1987 Ward Cunningham y Kent Beck escribieron un artículo llamado “Usando Patrones de Software para Software Orientado a Objetos”[1] en el cual se describen cinco patrones que resuelven problemas en el diseño de aplicaciones con Smalltalk para Windows, este fue uno de los primeros esfuerzos para utilizar el trabajo de Alexander hacia el desarrollo de software.

A partir de esto la comunidad reconoció rápidamente la utilidad de los patrones y empezaron a escribir sobre ellos. Posteriormente aparecieron artículos, revistas y libros sobre este tema, siendo el libro “Design Patterns: Elements of Reusable Object-Oriented Software”[2] el que trajo una aceptación a gran escala de los Patrones de Software en el mundo del software orientado a objetos. Este libro escrito por la llamada Gang of Four (GoF) es generalmente considerado como la base del estudio de patrones de software y es utilizado hasta la actualidad.

El libro de patrones de GOF, habla solamente de los patrones de diseño, sin embargo hoy en día los patrones de software han sido agrupados y recogidos de las mejores prácticas de diseño y del amplio conocimiento de expertos e intentan obtener el máximo beneficio de una de las características de la programación orientada a objetos como llamada reutilización.

Los patrones actualmente no se enfocan solamente en micro arquitecturas como son los patrones de diseño, si no que este concepto de reutilización se ha

[1] W. Cunningham, K. Beck, Constructing abstractions for object-oriented applications

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides., Design Patterns: Elements of Reusable Object Oriented Software

expandido, hasta definir patrones para el diseño de arquitecturas completas de sistemas muy grandes, patrones para acceso a almacenes de datos, e incluso patrones para pruebas o implantación de sistemas. Aún de esta manera el concepto base de los patrones se mantiene igual y es en base la reutilización de soluciones probadas para problemas específicos dentro de un contexto común.

2.2. UML - Lenguaje Unificado de Modelamiento (Unified Modeling Language)

Los diagramas de UML son muy utilizados para describir la funcionalidad de software a varios niveles de abstracción, por esta razón UML es utilizado como lenguaje de modelamiento para la descripción de patrones de software.

Por lo general se utilizan dos tipos de diagramas UML para la descripción de los patrones: diagramas de clases para mostrar la estructura del patrón, diagramas de secuencia para mostrar la interacción entre los distintos componentes, de esta forma es posible definir los patrones de una manera más estándar y que pueda ser interpretada por la gente de desarrollo de aplicaciones.

2.2.1. Introducción

UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa

para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir.[3]

UML sirve como un estándar para describir un sistema de información y todos sus componentes internos y externos, y su relación con esquemas de bases de datos, lenguajes de programación, todo en un ambiente orientado a objetos. UML es soportado por el grupo OMG (Object Management Group)[4] para el intercambio de objetos distribuidos.

Si bien UML no se propone como una metodología o un proceso de desarrollo de software soporta y puede servir de base para describir un proceso unificado de desarrollo.

La actual versión de UML es 2.0 y está compuesto por 13 tipos de diagramas agrupados de la siguiente forma.

Tabla 2-1: Diagramas UML 2.0[5]

Área	**Vista**	**Diagramas**	**Conceptos Principales**
Estructural	Vista Estática	Diagrama de Clases	Clase, asociación, generalización, dependencia,

[3] http://www.creangel.com/uml/home.php
[4] http://en.wikipedia.org/wiki/Object_Management_Group
[5] http://www.creangel.com/uml/diagramas.php

			realización, interfaz.
	Vista de Casos de Uso	Diagramas de Casos de Uso	Caso de Uso, Actor, asociación, extensión, generalización.
	Vista de Implementación	Diagramas de Componentes	Componente, interfaz, dependencia, realización.
	Vista de Despliegue	Diagramas de Despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de Estados de máquina	Diagramas de Estados	Estado, evento, transición, acción.
	Vista de actividad	Diagramas de Actividad	Estado, actividad, transición, determinación, división, unión.
	Vista de interacción	Diagramas de Secuencia	Interacción, objeto, mensaje, activación.
		Diagramas de Colaboración	Colaboración, interacción, rol de colaboración,

			mensaje.
Administración o Gestión de modelo	Vista de Gestión de modelo	Diagramas de Clases	Paquete, subsistema, modelo.
Extensión de UML	Todas	Todos	Restricción, estereotipo, valores, etiquetados.

Un diagrama UML pretende describir una vista del mundo real y tomar los datos importantes de ésta vista para poder describir completamente un modelo a un nivel de detalle que sea comprensible no solo para la persona que diseña el modelo.

Un diagrama está compuesto por un conjunto de grafos bidimensionales como íconos, símbolos, rutas, cadenas y relaciones con otros elementos del modelo, cada uno puede conectarse con otro elemento del modelo mediante relaciones para mostrar su relación e interacción. Dichos elementos van contenidos dentro de un paquete, que sirve para agrupar los elementos dependiendo de un punto funcional o un ámbito en específico; es decir, pueden haber varios paquetes y dentro de estos paquetes los elementos gráficos más pequeños.

Se describen a continuación los diagramas más importantes:

2.2.2. Diagramas de clases

Un diagrama de clases está conformado por todas las clases que se encuentran en el modelo de un sistema. Cada clase muestra su estructura mediante atributos, operaciones, herencia y relación con otras clases.

Una clase se representa de la siguiente forma:

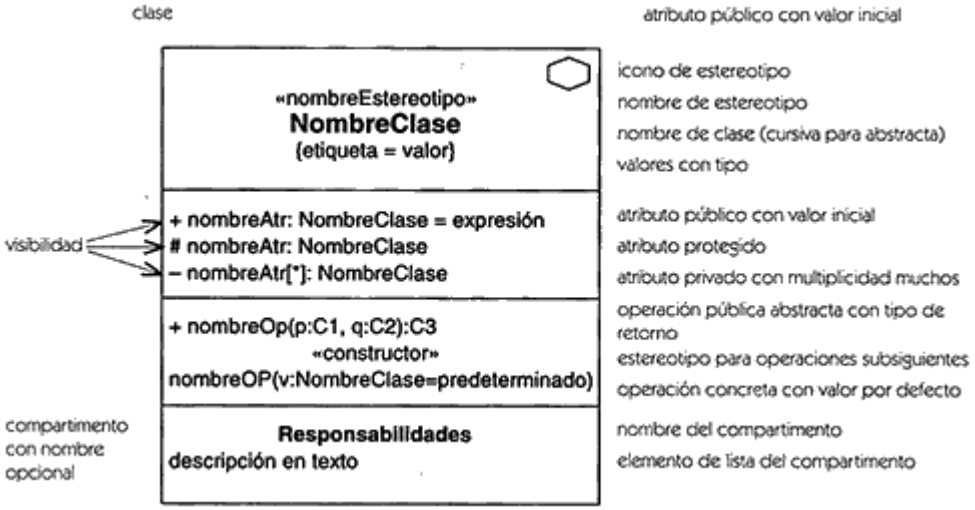


Figura 2-1: Estructura de un diagrama de clases[6]

- El primer rectángulo tiene el nombre de la clase
- El segundo contiene los atributos definidos con su nivel de visibilidad hacia otras clases:
    - (-) Privado: totalmente invisible
    - (#) Protegidos: visibles para las clases amigas (friends) y para las clases derivadas de la original.

[6] http://www.creangel.com/uml/clases.php

- (+) Públicos: visibles a otras clases.

- El Tercero las operaciones

Adicionalmente las clases tienen dos tipos de relaciones con otras clases:

Generalización

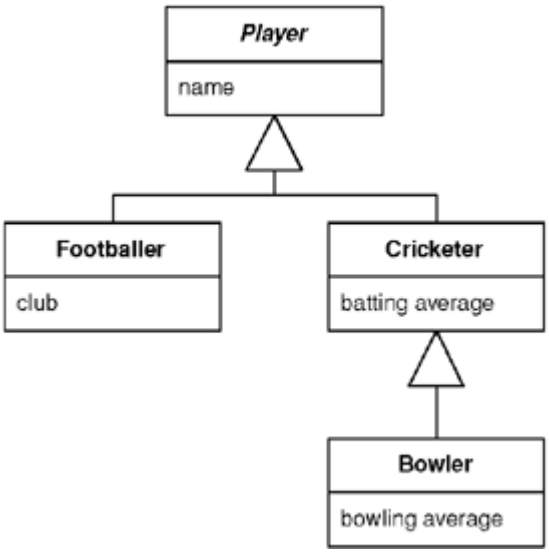


Figura 2-2: Generalización de clases[7]

Consiste en crear una clase más grande o superclase tomando como base las propiedades comunes de clases más pequeñas que tengan relación. Las subclases heredan propiedades de sus clases padre, es decir, atributos y de la clase padre están disponibles en sus clases hijas.

[7] Martin Fowler, Enterprise Application Architecture Patterns

Asociación

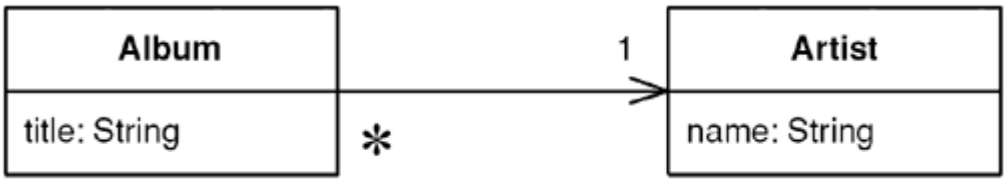


Figura 2-3: Asociación de Clases[8]

La asociación expresa una conexión bi-direccional entre objetos. Una asociación es una abstracción de la relación existente en los enlaces entre los objetos. Puede determinarse por la especificación de multiplicidad (mínima...máxima)[9]

- Uno y sólo uno
- 0..1 Cero o uno
- M..N Desde M hasta N (enteros naturales)
- Cero o muchos
- 0..* Cero o muchos
- 1..* Uno o muchos (al menos uno)

2.2.3. Diagrama de casos de uso

La utilidad principal de un diagrama de casos de uso es la de capturar o levantar los requerimientos de un proceso de negocio o un sistema. Mediante elementos gráficos se puede describir cómo se desea que un sistema trabaje,

[8] Martin Fowler, *op. cit.*
[9] http://www.creangel.com/uml/clases.php

cuáles son los entes internos o externos que van a interactuar con el sistema y los límites del mismo.

La importancia de estos casos de uso es que están descritos en un lenguaje natural y pueden ser captados por usuarios que pueden o no estar relacionados con aspectos técnicos.

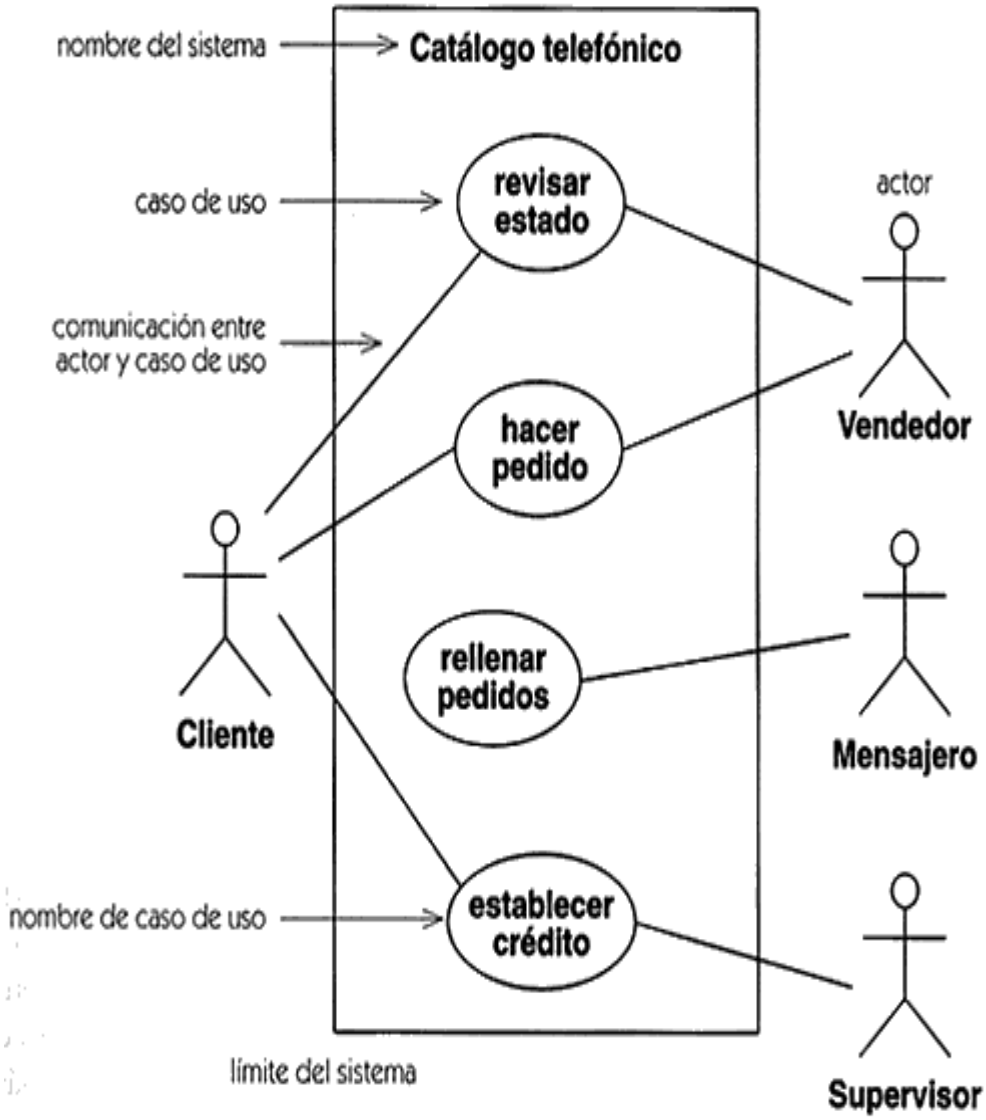


Figura 2-4: Diagrama de casos de uso[10]

[10] http://www.creangel.com/uml/casouso.php

Actores

El actor es todo aquel ente que interactúa con el sistema, puede representar a una persona o a un rol o a otro sistema o componente. El actor envía a o recibe del sistema mensajes o intercambia información. Los tipos de actores son:

- Principales: personas que usan el sistema.
- Secundarios: personas que mantienen o administran el sistema.
- Material externo: dispositivos materiales imprescindibles que forman parte del ámbito de la aplicación y deben ser utilizados.
- Otros sistemas: sistemas con los que el sistema interactúa.

Generalmente para determinar los actores se debe analizar los casos de uso y plantearse preguntas como: ¿Quién utilizará la funcionalidad principal del sistema? ¿Con qué otros sistemas necesitarán interactuar el sistema a desarrollar?

2.2.4. Diagrama de Componentes

Un diagrama de componentes es un conjunto de módulos de software que muestra la parte física interna de un sistema, puede incluir componentes de código fuente, componentes del código binario, componentes ejecutables, archivos y bibliotecas de clases DLL. Los componentes también pueden contener entidades físicas, documentos, bases de datos, etc.

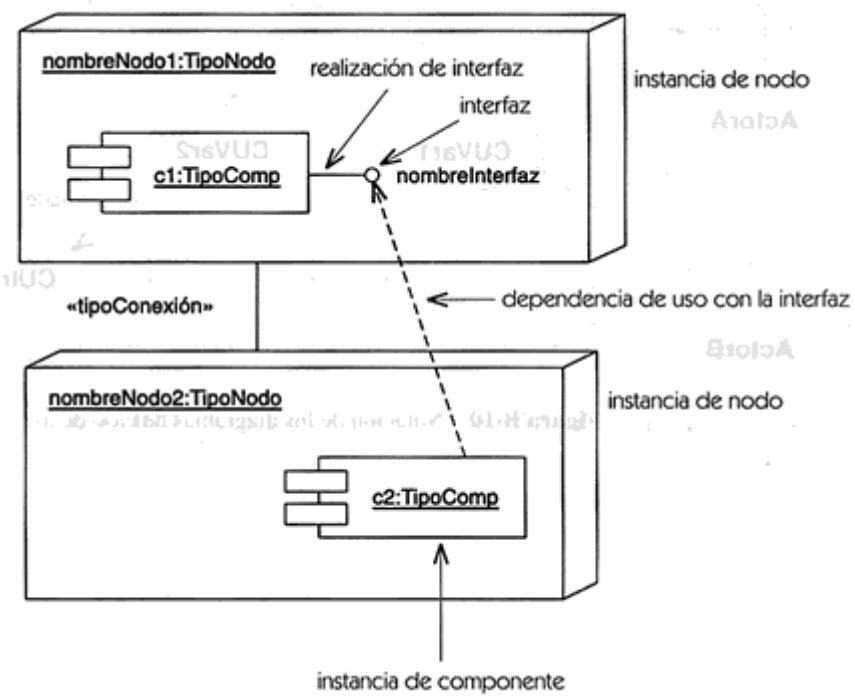


Figura 2-5: Diagrama de componentes[11]

Un componente de software puede tener operaciones e interfaces las cuales se relacionan con otros componentes del sistema; un componente puede mostrar una dependencia o relación con otro componente mediante flechas con líneas discontinuas.

En el diagrama de componentes también se puede mostrar la estructura de implementación de la aplicación como tal, en la cual se establece una correspondencia entre las clases y los componentes de implementación.

[11] http://www.creangel.com/uml/componente.php

2.2.5. Diagramas de Estados

Es un diagrama que representa el estado de un objeto en un determinado momento, su comportamiento en dicho estado junto con los cambios que le permiten pasar de un estado a otro. El estado está dado generalmente por los valores que tienen en esa instancia del tiempo los atributos que conforman el objeto.

Se representa mediante un rectángulo con los bordes redondeados, que puede tener tres compartimientos: uno para el nombre, otro para el valor característico de los atributos del objeto en ese estado y otro para las acciones que se realizan al entrar, salir o estar en un estado (entry, exit o do, respectivamente).[12]

Evento

El paso de un estado a otro del objeto esta dado por la ocurrencia de un evento que puede estar dada en las siguientes ocasiones:

- Recepción de una señal o mensaje de otro objeto del modelo
- Cambios respecto a un periodo de tiempo dado por una hora o fecha.

Para representar en envío de un mensaje se une mediante una línea punteada dirigida al diagrama de estados del objeto receptor del mensaje.

[12] http://www.creangel.com/uml/estado.php

Transiciones de objetos

La transición indica el paso de un estado a otro estado y las operaciones que puede realizar, se representa a través de una línea sólida.

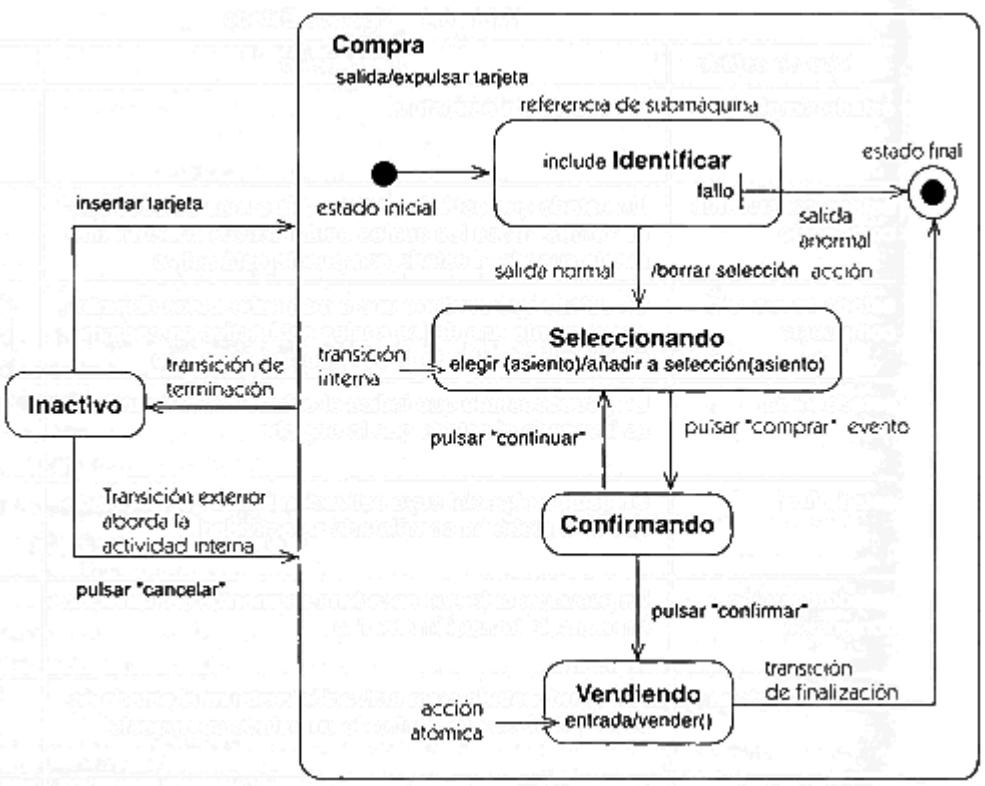


Figura 2-6: Diagrama de estados[13]

2.2.6. Diagrama de Actividad

Un diagrama de actividad representa un estado pero visto desde un punto de vista de acciones. Un estado de actividad representa un flujo de trabajo o la

[13] http://www.creangel.com/uml/estado.php

ejecución de un proceso o tarea. Puede existir un grafo de actividades en el cual una actividad desencadena otra actividad después de una ejecución determinada.

Una actividad desencadena acciones que puede tener parámetros de entrada y salida; así cuando una actividad termina su proceso puede desencadenar un cambio de estado y no necesariamente esperar un evento como se puede notar en el diagrama de estados.

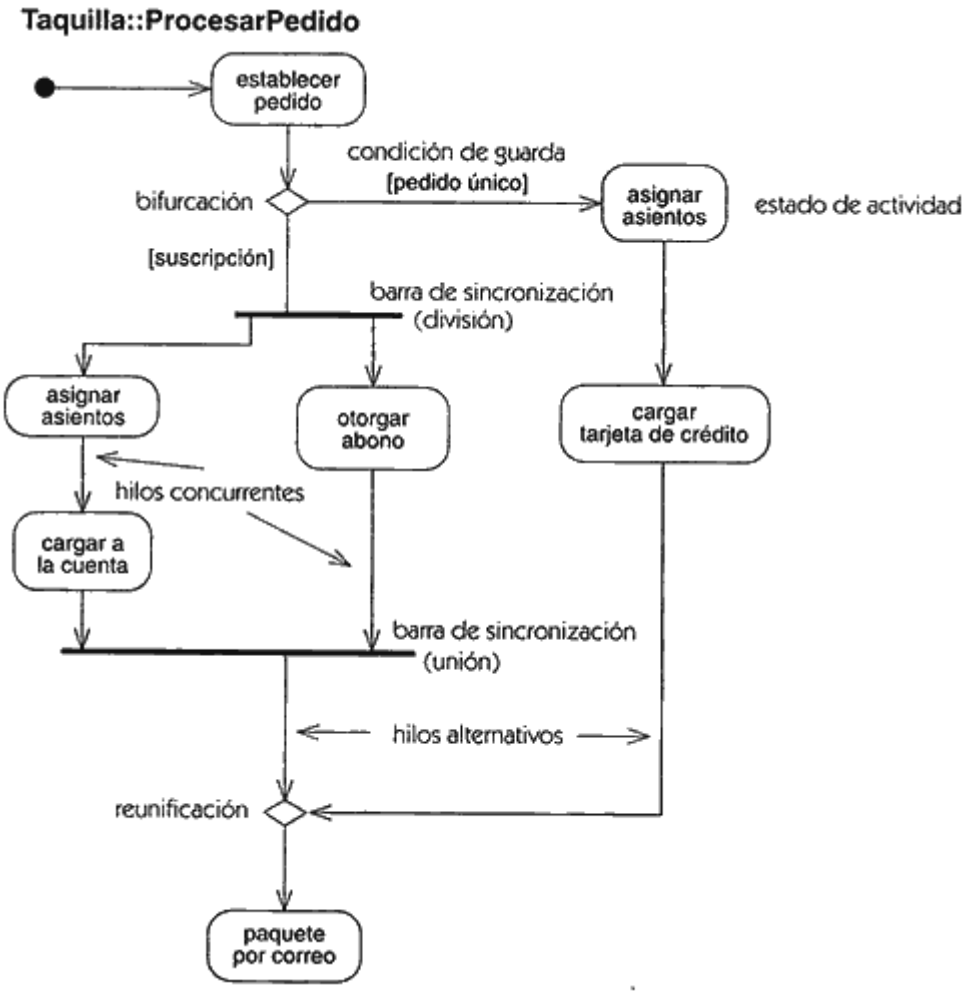


Figura 2-7: Diagrama de actividades[14]

[14] http://www.creangel.com/uml/actividad.php

Un grafo de actividades puede tener puntos de decisión y de control en hilos concurrentes, en los cuales se puede representar varias acciones concurrentes que pueden ser generadas por diversos objetos o personas; esto lo difiere con un diagrama de flujo tradicional ya que se puede tener varios hilos ejecutándose al mismo tiempo.

2.2.7. Diagramas de Interacción

Los diagramas de interacción muestran un patrón de interacción entre objetos y la secuencia de intercambios de mensajes entre ellos. La principal utilidad de estos diagramas es la de mostrar los aspectos dinámicos de un sistema y la relación entre los objetos involucrados en él.

Existen dos diagramas que componen los diagramas de interacción:

- Diagramas de secuencia, enfocados en mostrar cronológicamente o en orden una secuencia temporal de mensajes que participan en una interacción.
- Diagramas de colaboración, enfocados en mostrar la estructura y organización de los objetos que participan en la interacción.

La diferencia entre los diagramas de secuencia y colaboración es que los diagramas de colaboración muestran las relaciones entre los objetos participantes, pero los diagramas de secuencia muestran las secuencias temporales y los flujos de ejecución.

El diagrama de Colaboración puede obtenerse automáticamente a partir del correspondiente diagrama de Secuencia (o viceversa).

2.2.7.1. Diagrama de Secuencia

Un diagrama de secuencia muestra una interacción ordenada de una secuencia explícita de mensajes. Muestra los objetos que participan en la interacción y el intercambio de mensajes en una secuencia de tiempo.

Cada objeto se representa con una barra vertical cuya longitud representa el tiempo de interacción dentro del flujo de comunicación. Los mensajes se muestran como flechas entre líneas de vida. Cuando existe demora entre el envío se puede indicar usando una línea oblicua

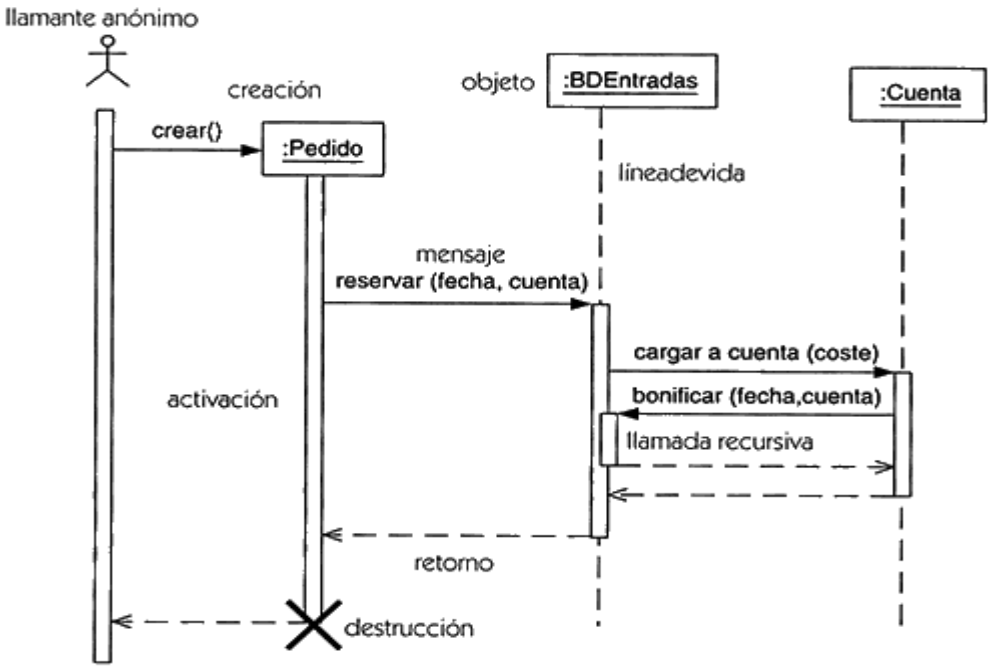


Figura 2-8: Diagrama de secuencia[15]

[15] http://www.creangel.com/uml/secuencia.php

Una secuencia dentro del diagrama posee dos dimensiones o ejes: un eje vertical el cual representa el tiempo, y un eje horizontal que representa los objetos que participan en la interacción; se crea una nueva columna por cada objeto. El tiempo transcurre desde arriba hacia abajo.

Cada objeto genera un mensaje hacia el otro objeto el cual se une con una flecha. Se pueden generar bifurcaciones hacia otros objetos o hacia sí mismo. Cada bifurcación puede enviar y recibir mensajes.

El Diagrama de Secuencia es más adecuado para observar la perspectiva cronológica de las interacciones y son mejores para especificaciones de tiempo real y para escenarios complejos.

2.2.7.2. Diagrama de Colaboración

Representa la estructura de los objetos que interactúan con un cierto propósito dentro un contexto. Cada objeto cumple un rol dentro de una colaboración; los roles pueden estar dados por un rol de asociación el cual representa una descripción de los enlaces que pueden participar en una ejecución de colaboración, y un rol clasificador el cual es una asociación que está limitada por tomar parte en la colaboración.

Una colaboración está dada por dos partes:

- Estructural o estática, la cual describe el conjunto de roles y relaciones entre los objetos.

- Comportamiento, describe el conjunto de mensajes intercambiados por los objetos ligados a los roles, los cuales conforman una interacción.

Figura 2-9: Diagrama de colaboración[16]

Para representar este diagrama cada objeto muestra una serie de objetos con los que se enlaza en conjunto con los mensajes que intercambian entre ellos. Los mensajes se representan con flechas que van junto al enlace y con el nombre del mensaje y los parámetros entre paréntesis. Cada mensaje lleva un número de secuencia que muestra cuál es el mensaje que le precede (excepto el mensaje inicial). Se pueden indicar alternativas con condiciones entre corchetes.

El Diagrama de Colaboración ofrece una mejor visión espacial mostrando los enlaces de comunicación entre objetos, muestra las relaciones entre objetos y son mejores para comprender todos los efectos que tiene un objeto para el diseño de procedimientos.[17]

[16] http://www.creangel.com/uml/colaboracion.php
[17] http://www.creangel.com/uml/interaccion.php

2.3. Frameworks

2.3.1. Definición

Actualmente la capacidad de procesamiento de las computadoras, y el ancho de banda disponible en las redes de datos, se han incrementado muy dramáticamente, pero el diseño de grandes y complejos sistemas de software se mantiene susceptible a errores. Mucho esfuerzo y también recursos, se dedica a redescubrir y rehacer componentes a lo largo de la industria de software. Además el creciente número de arquitecturas de hardware y la diversidad de sistemas operativos y plataformas de comunicaciones hacen difícil el desarrollar aplicaciones portables y eficientes desde cero.

En el diccionario[18] encontramos la palabra framework: se refiere a un armazón, una estructura que mantiene o soporta algo que está siendo construido en base a él, es decir un esqueleto, o también como una plataforma de trabajo para el desarrollo en base a él.

En software el concepto de un framework mantiene la misma idea, se lo toma como un conjunto de clases abstractas, que buscan realizar alguna funcionalidad común para un cierto tipo de aplicación, este diseño debe ser reutilizable y es considerado como una micro-arquitectura. Usualmente no es una aplicación completa, la mayoría de veces no provee la funcionalidad específica de

[18] Websters II, New Riverside University Dictionary, pag 503

una aplicación. Al contrario una aplicación deberá ser construida a partir de uno o más frameworks agregando la funcionalidad específica que la aplicación requiere.

El framework sobre el que una aplicación está desarrollada es el que determina la arquitectura de una aplicación, definiendo la forma en la que las clases interactúan, el flujo de los datos y el control, de esta manera se separan claramente las responsabilidades de cada uno de los bloques de la aplicación, haciendo que los desarrolladores puedan enfocarse de una mejor manera a realizar tareas específicas que la aplicación requiere. Es decir un framework provee la infraestructura de base necesaria para la interacción de los diferentes componentes de una aplicación.

Un framework está orientado a un cierto "Application Domain" o dominio de aplicaciones, esto significa que un framework define funcionalidad que es aplicable para un cierto tipo de aplicaciones, por ejemplo, un framework que esté diseñado para aplicaciones distribuidas será diferente a un framework que se encuentre diseñado para ser implementado en aplicaciones Cliente-Servidor.

2.3.2. Características[19]

Un framework es un diseño orientado a objetos, aunque no necesariamente su implementación esté realizada en un lenguaje orientado a objetos. Generalmente se diferencian dos partes de un framework, una parte llamada "virtual engine", que es estática y no cambia independientemente de la aplicación,

[19] Object-Oriented Application Frameworks - http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html

mientras que las clases abstractas que son utilizadas como interfaces hacia la aplicación son también llamadas “plug points” o “hot spots”, estas clases son implementadas usualmente utilizando delegados o polimorfismo, hacen que el framework pueda ser extendido y adaptado para satisfacer las necesidades específicas de una aplicación. Aunque también usualmente se incluyen clases concretas que pueden ser instanciadas para poner a trabajar el framework directamente sin ninguna modificación.

De esta manera un framework no solamente se centra en el concepto de reutilización de código de la orientación a objetos, sino también amplía esto de manera que se busca la reutilización del diseño de un sistema, sobre la reutilización de código.

Las características de un framework dependerán del tipo de aplicación para el que el framework fue desarrollado, pero en general un framework debe tener 2 características básicas:

- Debe ser lo suficientemente simple para que el desarrollador pueda aprender a utilizarlo correctamente.
- Debe proveer suficiente funcionalidad y puntos de entrada para que pueda ser utilizado rápidamente, además de poder ser personalizado sin mucho esfuerzo.

Los principales beneficios de la utilización de un framework en el desarrollo de aplicaciones son:

- Modularidad: Al encapsular la implementación volátil específica de una aplicación detrás de interfaces probadas y estables. La modularidad de un framework ayuda a mejorar la calidad del software, localizando el impacto de cambios en el diseño e implementación de un software. Esto reduce el esfuerzo requerido para entender y mantener las aplicaciones.

- Reusabilidad: Un framework debe proveer interfaces, definiendo de esta manera componentes genéricos que pueden ser aplicados para crear nuevas aplicaciones. La reusabilidad de un framework conlleva un mejoramiento en la calidad, rendimiento, y estabilidad de un software.

- Escalabilidad: Al proveer interfaces estables y probadas, un framework mejora la escalabilidad ya que las aplicaciones pueden agregar funcionalidad rápidamente utilizando estas interfaces como punto de partida, teniendo ya un conjunto de componentes robustos que les ayuda a enfocarse en la parte específica de una aplicación.

2.3.3. Tipos de Frameworks

Aunque los principios en los que el diseño de un framework y los beneficios de su utilización son independientes del dominio de aplicaciones a los que son

aplicados, se puede clasificar a los frameworks tomando en cuenta el ámbito en el que se desenvuelven.

2.3.3.1. System infrastructure frameworks[20]

Frameworks para infraestructura de un Sistema. Estos frameworks simplifican el desarrollo de aplicaciones, ya sea frameworks de comunicaciones, interfaces de usuario, etc. Este tipo de frameworks son internos a la organización y no son vendidos a los consumidores directamente.

2.3.3.2. Middleware integration frameworks[21]

Frameworks de Integración. Estos frameworks son comúnmente utilizados para integrar aplicaciones distribuidas y componentes. Son diseñados para mejorar la habilidad de los desarrolladores de software de modularizar, reutilizar, y extender la infraestructura de las aplicaciones para trabajar en ambientes distribuidos. Estos frameworks pueden ser vendidos a los consumidores, que usualmente son otras organizaciones dedicadas al desarrollo de software.

[20] Object-Oriented Application Frameworks - http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html

[21] Íd., Ibíd.

2.3.3.3. Enterprise application frameworks[22]

Frameworks para Aplicaciones Empresariales. Estos frameworks tratan con muchos dominios de aplicaciones, como telecomunicaciones, aeronáutica, finanzas. En comparación con los frameworks de infraestructura o integración, son costosos de desarrollar o de adquirir. Los frameworks empresariales soportan el desarrollo de aplicaciones de usuario directamente, en contraste a los frameworks de integración e infraestructura, que se enfocan en el desarrollo interno a una organización.

Independientemente del ámbito, los frameworks pueden ser clasificados dependiendo de las técnicas que se utilicen para extenderlos, o personalizarlos.

2.3.3.4. White Box, Frameworks de Caja Blanca[23]

Este tipo de frameworks descansan fuertemente en las características de la programación orientada a objetos, como son herencia y polimorfismo, más la habilidad de sobrecargar funcionalidad genérica con específica para adquirir las características necesarias de extensibilidad. Esto genera un acoplamiento muy fuerte entre la superclase y su derivada, en términos de un íntimo conocimiento de su implementación y además que al instanciar una subclase automáticamente se crea una instancia de su superclase. Este fuerte acoplamiento es un factor que afecta mucho a la flexibilidad y extensibilidad, y estas son las dos razones principales para el desarrollo de un framework.

---

[22] Object-Oriented Application Frameworks - http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html

[23] Framework Evolution - http://www.cbd-hq.com/articles/2000/000401je_frameworks2.asp

Un framework de caja blanca implica que el consumidor del framework debe tener acceso al código fuente, o en su defecto utilizar herencia para agregar la funcionalidad requerida. De cualquier manera se necesita un conocimiento amplio acerca de la implementación del framework. Este nivel de conocimiento requerido hace que los frameworks de caja blanca tengan curvas de aprendizaje bastante largas, además se requiere documentación muy detallada, y ejemplos de su utilización, para utilizarlos de una manera eficiente y efectiva.

Usualmente se dice que un framework de caja blanca está en la "adolescencia", ya que se está separando la parte estática del framework de los puntos de ingreso para su extensión.

En las empresas de desarrollo, se suele decir que a partir de este punto un framework tiene vida por sí mismo, y debería tener su propio equipo de desarrollo y manejo. De esta manera con un nuevo proyecto, durante las fases de análisis, diseño e implementación, los frameworks serán extendidos para cumplir con las características específicas del proyecto, y el equipo de desarrollo del framework puede ser notificado de cualquier error que se encuentre. Lo óptimo sería que uno de los miembros del equipo encargado del desarrollo del framework se incluya en todas las implementaciones, para promover un bucle de retroalimentación continuo.

Luego de que un proyecto se ha finalizado, deberá empezar una fase en la que se determine las partes del framework que deberán ser cambiadas, para una mejora continua del framework.

2.3.3.5. Black Box, Frameworks de Caja Negra[24]

Los frameworks de caja negra soportan la extensión de sus funciones definiendo interfaces para componentes que pueden ser enlazados al framework vía composición de objetos. La funcionalidad existente es reutilizada definiendo componentes que utilicen una interfase particular, e integrando estos componentes en el framework.

Este modelo de composición actualmente es muy aplicado. Los grandes componentes de negocios, son desarrollados al rededor de los conceptos de interfaces, composición y delegación. La mayor parte de los modelos de componentes hoy en día son desarrollados de esta manera. Además de agregar una gran ventaja ya que los componentes desarrollados de esta manera son independientes del lenguaje de programación.

Los frameworks de caja negra son tomados como frameworks “maduros” al contrario de frameworks de caja blanca, ya que una vez que el framework es lo suficientemente estable, se lo puede volver estático, esto es que no se requiere herencia para agregar funcionalidad. Cualquier configuración es hecha a partir de interfaces, y “proveedores” pueden agregar cualquier nuevo comportamiento.

Los frameworks de caja blanca además requieren que los desarrolladores tengan un conocimiento muy amplio acerca de la estructura interna del framework, aunque los frameworks de caja blanca son ampliamente utilizados, se

[24] Framework Evolution - http://www.cbd-hq.com/articles/2000/000401je_frameworks2.asp

tiende a producir sistemas que están fuertemente acoplados a detalles específicos del framework. En contraste con los de caja blanca los frameworks de caja negra son estructurados usando composición de objetos y delegación en reemplazo a la herencia. Como resultado los frameworks de caja negra son generalmente más fáciles de utilizar y extender que los frameworks de caja blanca. Pero los frameworks de caja negra son más difíciles de desarrollar ya que requieren que sus desarrolladores definan interfaces y puntos de ingreso que anticipen un rango muy amplio de usos potenciales.

2.3.3.6. Grey Box, Frameworks de Caja Gris[25]

Realizando un análisis de la evolución de los framework, las cajas negras y blancas son los extremos del diseño y utilización de un framework. Muchos frameworks probablemente vivan en la región entre estos dos extremos, como frameworks de caja gris.

Los frameworks de caja gris intentan sacar provecho de los beneficios de ambos tipos de framework, mientras intentan evitar las limitaciones de ambos. Realizando un análisis del ciclo de vida de los distintos tipos de frameworks los frameworks de caja negra representan la etapa de madurez de un framework, pero muchas veces hay limitaciones tanto de tiempo como de dinero que impiden alcanzar esta etapa, por lo que el desarrollo del Framework queda “a medio camino” como un Framework de caja gris.

[25] Framework Evolution - http://www.cbd-hq.com/articles/2000/000401je_frameworks2.asp

2.3.3.7. Visual Builders y Soporte de Lenguaje[26]

Cuando un framework alcanza la más alta madurez, es decir el tope de su evolución. Las aplicaciones son “armadas” como un rompecabezas, con una herramienta que permita el diseñar el sistema de una manera visual, y un lenguaje a nivel macro de la aplicación. Con este nivel de soporte el framework está listo para ser movido fuera de las manos de un desarrollador de aplicaciones, hacia un usuario final del sistema.

Un ejemplo de un visual builder y un framework que esté completamente maduro sería el que un experto en el tema de seguros, en lugar de interactuar con un desarrollador, lo haga como una herramienta como Visual Basic. El podría crear una aplicación de seguros, basada en un framework específico para el dominio. Seleccionar un consumidor, una póliza de seguros, que proveen las interfaces necesarias, escribir unas pocas líneas en un lenguaje natural para procesar los eventos deseados, y luego un botón de Generar y Distribuir. Si esto suena difícil de creer sería comparable al hablar de Delphi a un programador de ensamblador de los años 60.

2.3.4. Frameworks y Patrones de Software

Los patrones de Software pueden ser empleados tanto en el diseño como en la documentación de un framework. Un solo framework típicamente utiliza muchos de ellos. De hecho, un framework puede ser visto como una implementación de un sistema de patrones de software. Aún con el hecho de que los patrones de

[26] Framework Evolution - http://www.cbd-hq.com/articles/2000/000401je_frameworks2.asp

software y los frameworks están relacionados de esta manera, es importante el reconocer que son dos temas diferentes: un framework es software ejecutable, en cambio los patrones de software representan el conocimiento y la experiencia acerca del software. Es decir los frameworks son de una naturaleza “física”, mientras los patrones son de una naturaleza “lógica”. Los frameworks son la realización física de una o más soluciones dadas por patrones, mientras que los patrones son las instrucciones de como implementar dichas soluciones.

En el libro de “Gang of Tour, Elements of Reusable Object Oriented Software”[27], se describen las diferencias que existen entre los frameworks y una parte de los patrones de software como son los patrones de diseño.

Los patrones de diseño son más abstractos que los frameworks. Los frameworks pueden ser creados con código, pero solamente ejemplos de patrones pueden ser traducidos a código. Una de las fortalezas de los frameworks es que pueden ser escritos en lenguajes de programación, y no solamente estudiados, sino además ejecutados y reutilizados directamente. En cambio, los patrones de diseño deben ser implementados cada vez que son utilizados. Los patrones además explican la intención, problemas y consecuencias de un diseño.

2.4. Descripción de la Tecnología .net

.NET es una arquitectura tecnológica desarrollada por Microsoft que ofrece grandes herramientas, servicios y soporte para la creación y distribución de

[27] E. Gamma, R. Helm, R. Johnson, J. Vlissides., Design Patterns: Elements of Reusable Object Oriented Software

software, brindando soluciones a los problemas de programación en ambientes distribuidos, clientes inteligentes, servicios y aplicaciones de Internet. Esta arquitectura está compuesta por componentes esenciales que son los siguientes:

- **La plataforma .net framework**, representa la infraestructura para la creación de aplicaciones y es el ambiente de ejecución de las mismas.
- **Las herramientas de desarrollo** que van desde el sistema operativo Windows hasta los IDES de desarrollo como Visual Studio .net y un conjunto de herramientas integradas.
- **Servidores Empresariales** para repositorio de datos (SQL Server), herramientas que interactúan con las aplicaciones generadas y dan soporte a tareas comunes como integración y comunicación (Biztalk Server, Commerce Server, etc.).

2.4.1. El .net Framework

El .net framework es una infraestructura que reúne un conjunto de lenguajes y servicios, librerías de clases y un ambiente común de ejecución altamente distribuido y constituye la plataforma y elemento principal sobre el que se fundamenta la tecnología .NET. Se tiene como pilar fundamental las herramientas y tecnologías para la generación de interfaces de usuario: aplicaciones de escritorio (Windows Forms), aplicaciones para Internet (ASP.net y Web Services), aplicaciones ligeras para dispositivos móviles, clientes inteligentes, entre otros.

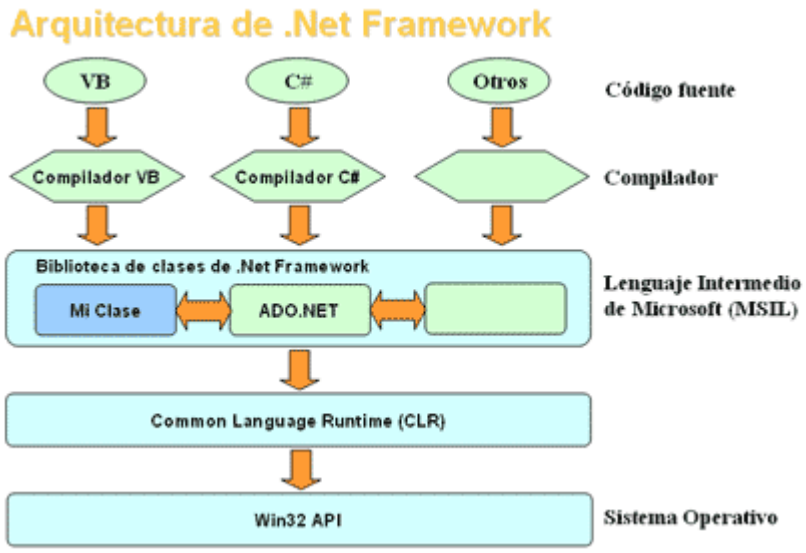


Figura 2-10: Arquitectura de .Net Framework[28]

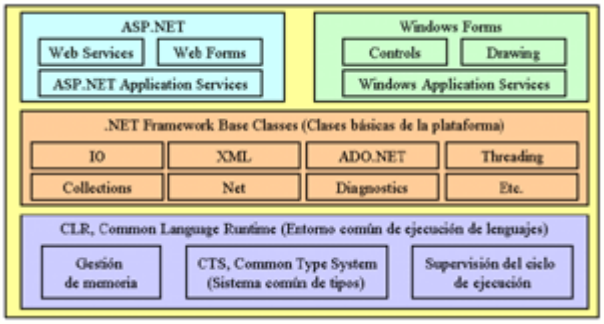


Figura 2-11: Componentes de .Net Framework[29]

2.4.2. El CLR Common Language Runtime[30]

El CLR es uno de los principales componentes del .NET Framework. Provee el ambiente de ejecución del código y varios servicios para las aplicaciones.

[28] Luis Miguel Blanco. Framework .net
[29] Íd., Ibíd.
[30] Íd., Ibíd.

Algunas de sus características y diseño son:

Administración de memoria: desde la carga y disposición del código en memoria hasta su finalización y descarga que es realizada por el proceso de Garbage Collector. El CLR detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente, liberando al programador de realizar esta tarea.

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

Sistema común de tipos de datos: un modelo de tipos orientado a objeto, que soporta múltiples lenguajes.

Class Loader: Carga las clases dentro del ambiente de ejecución.

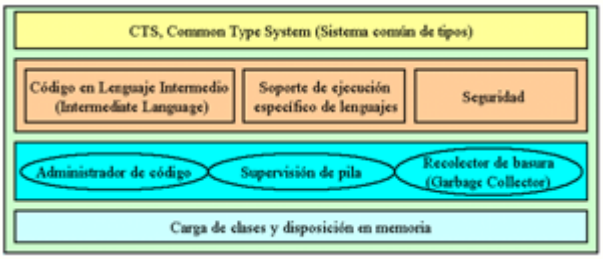


Figura 2-12: CLR Common Language Runtime[31]

[31] Luis Miguel Blanco, *op. cit.*

2.4.2.1. Ejecución de Código dentro del CLR

Existen varios elementos y técnicas que intervienen dentro éste proceso como carga del código en memoria, traducción a un lenguaje común de ejecución, compilación y depuración.

2.4.3. IL Intermediate Language (MSIL Microsoft Intermediate Language)

A diferencia de tecnologías anteriores, los compiladores de código de .net no traducen el código fuente a código binario sino a un lenguaje intermedio IL (Intermediate Language); de esta forma, indistintamente del lenguaje utilizado, el código generado es siempre el mismo, ya que el IL es el único lenguaje que entiende directamente el CLR.

Adicionalmente, este lenguaje es independiente del sistema operativo y/o procesador en el que se pretende ejecutar la aplicación.

2.4.4. Compilador Just In Time

Ya que el IL no representa un código de máquina se necesita un paso adicional que consiste en un compilador JIT (Just in time), el cual genera el código máquina real que se ejecuta en la plataforma que tenga la computadora, consiguiendo cierta independencia de plataforma, ya que cada plataforma puede tener su compilador JIT y crear su propio código máquina a partir del código IL.

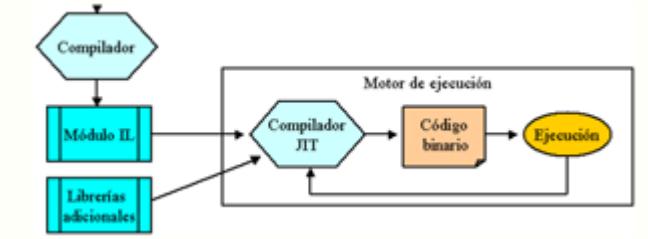


Figura 2-13: Compilador Just in time[32]

Para que este proceso tenga mejor rendimiento, al ejecutar la aplicación no se compila todo su IL, sino que sólo se compila el código según se va necesitando y se almacena el código de máquina resultante.

Ejemplo de un código transformado en MSIL

Utilizando la herramienta ILDASM que viene incorporada con el framework de .net podemos visualizar el código IL compilado de un método de una clase de ejemplo.

Tabla 2-2: Ejemplo de un código transformado en MSIL[33]

Ejemplo de Código en Lenguaje C#	***Código Compilado en MSIL***
public string Ejemplo() *{* *int i = 1;* *i++;*	*.method Public hidebysig instance string* *Ejemplo() cil managed* *{* *// Code size 38 (0x26)*

[32] Luis Miguel Blanco, *op. cit.*
[33] Elaboración Propia

string str = "Hola Mundo";
str += " " + i.ToString();
return str;
}
	.maxstack 3
	.locals init ([0] int32 i,
	[1] string str,
	[2] string CS$1$0000)
	IL_0000: nop
	IL_0001: ldc.i4.1
	IL_0002: stloc.0
	IL_0003: ldloc.0
	IL_0004: ldc.i4.1
	IL_0005: add
	IL_0006: stloc.0
	IL_0007: ldstr "Hola Mundo"
	IL_000c: stloc.1
	IL_000d: ldloc.1
	IL_000e: ldstr " "
	IL_0013: ldloca.s i
	IL_0015: call instance string [mscorlib]System.Int32::ToString()
	IL_001a: call string [mscorlib]System.String::Concat(string,
	string,
	string)
	IL_001f: stloc.1

	IL_0020: ldloc.1 *IL_0021: stloc.2* *IL_0022: br.s IL_0024* *IL_0024: ldloc.2* *IL_0025: ret* *} // end of method Class1::Ejemplo*

2.4.5. CTS (Common Type System)

El sistema común de tipos provee los tipos de datos necesarios, valores y tipos de objetos, los mismos que son necesarios para el desarrollo de aplicaciones en diferentes lenguajes. Todos los lenguajes de .NET comparten el mismo Common Type System, esto implica que un String en Visual Basic .NET es el mismo String en Visual C# o en cualquier lenguaje .NET. Todos los tipos de datos tienen su equivalente dentro del CLR y se puede mostrar en la siguiente tabla:

Tipo de dato (Nombre de clase)	Descripción
Byte	Entero sin signo de 8 bit
SByte	Entero sin signo de 8 bit (tipo no acorde con el CLS)
Int16	Entero con signo de 16 bit
Int32	Entero con signo de 32 bit
Int64	Entero con signo de 64 bit
UInt16	Entero sin signo de 16 bit (tipo no acorde con el CLS)
UInt32	Entero sin signo de 32 bit (tipo no acorde con el CLS)
UInt64	Entero sin signo de 64 bit (tipo no acorde con el CLS)
Single	Numero con coma flotante de precisión simple, de 32 bit
Double	Numero con coma flotante de precisión doble, de 64 bit
Boolean	Valor lógico
Char	Carácter Unicode de 16 bit
Decimal	Valor decimal de 96 bit

Figura 2-14: Tipos de datos en .Net Framework[34]

Dentro de .NET Framework, todos los tipos de datos están implementados como clases, cuando se declara una variable, ésta representa un objeto de la clase relacionada con el tipo de dato que contiene.

Existen dos categorías de tipos de datos:

Tipos Por valor: representan los tipos de datos nativos del framework, enumeradores, tipos definidos por el usuario, o cualquier tipo de dato que pueda ser accedido en forma directa.

[34] Luis Miguel Blanco, *op. cit.*

Tipos por referencia: representan las clases del .net framework, los clases definidas por el usuario, delegados, o cualquier tipo de datos que necesite ser instanciado o llamando a su dirección de memoria o referencia.

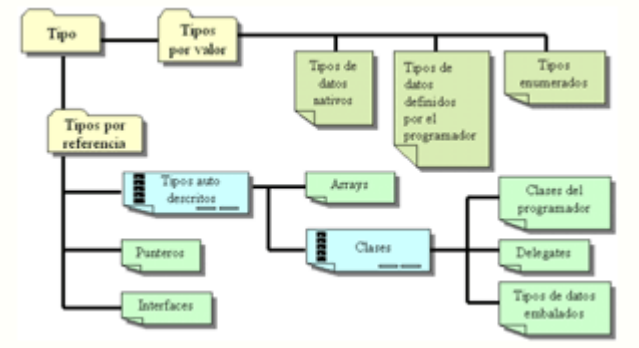


Figura 2-15: Categorías de tipos de datos[35]

2.4.6. CLS (Common Language Specification)

El Common Language Specification pretende dar los lineamientos y reglas que deben cumplir todos los lenguajes de .net para lograr interoperabilidad. El objetivo principal es poder brindar al desarrollador una independencia del lenguaje, que se puedan realizar componentes en cualquier lenguaje y poder utilizarlos en otro.

Adicionalmente se pueden seguir desarrollando más lenguajes e ir adaptándolos al framework, como ya se lo ha hecho con Cobol .net, Delphi .net, Perl .net.

[35] Luis Miguel Blanco, *op. cit.*

2.4.7. Namespaces

Un namespace es la organización lógica y jerárquica de las clases dentro de .net. Una clase pertenece a un namespace. Un ejemplo puede ser una clase llamada “Factura” y “Orden”, ambas clases estarían en el namespace denominado “Ventas”. Como resultado tenemos lo que se denomina el nombre calificado (fully qualified name) de una clase, que para nuestro ejemplo, sería Ventas.Factura.

Generalmente en el desarrollo de un sistema, además de jerarquizar las clases dependiendo del punto funcional, se compone al Namespace con el nombre de la compañía que lo desarrolla y el layer o capa en la que éste se encuentra. Ej, “G5Corp.CapaNegocio.Ventas.Factura”

2.4.8. Assemblies

Un ensamblado o assembly, consiste en un conjunto de clases reunidos para formar la unidad más elemental de código que puede ejecutar el entorno de .NET Framework. [36]

Además de clases un assembly contiene recursos, tipos de datos y el descriptor o manifiesto el cual contiene toda la información que describe al assembly, esta información incluye:

- Nombre. Una cadena con el nombre del ensamblado.

[36] Luis Miguel Blanco, *op. cit.*

- Versión. Número de versión.
- Cultura. Información sobre idioma y otros elementos de globalización.
- Nombre seguro. En el caso de ensamblados compartidos, este nombre permite identificar al ensamblado a través de una clave o firma.
- Lista de archivos. Los nombres de los archivos que forman el ensamblado.
- Ensamblados referenciados. Lista de los ensamblados con los que el actual mantiene dependencias.

Un ensamblado puede estar físicamente en uno o varios archivos DLL aunque lógicamente se trate de un solo ensamblado. Los ensamblados pueden ser compartidos o únicos para cada aplicación. En el caso de los ensamblados compartidos éstos se alojan en una zona compartida denominada Global Assembly Cache y para que éste pueda ser registrado en esta zona debe tener una firma o clave que lo identifique de manera única en el Assembly Cache.

Durante la ejecución de un programa, cuando sea necesario el uso de un ensamblado compartido, el CLR realizará la búsqueda de dicho ensamblado empleando su clave de identificación y el número de versión, para localizar el correcto ensamblado en la caché global.[37]

[37] Luis Miguel Blanco, *op. cit.*

2.4.9. Librería de clases

Figura 2-16: Librería de clases .Net Framework[38]

El .net framework nos provee de un conjunto bastante amplio de clases que se encargan de realizar las tareas más comunes que requieren las aplicaciones: seguridad, encriptación, manipulación de archivos, acceso a datos, entre otros.

Las clases están organizadas en namespaces, y para utilizar éstas clases dentro de la aplicación solo se debe hacer la referencia al namespace y declarar la clase.

Las clases de esta biblioteca no solo son clases comunes que implementan funcionalidad sino también nos brindan una plataforma tecnológica para construir aplicaciones Web más funcionales, ricas en interfase gráfica y fáciles de mantener; podemos destacar las siguientes tecnologías:

- ASP.NET para construir aplicaciones y servicios Web.
- ADO.NET para la conexión hacia fuente una fuente de datos.
- Windows Forms para desarrollar interfaces de usuario.

[38] Microsoft – MSDN, Cursos desarrollador 5 estrellas

2.4.10. Ado.net

ADO .NET es una tecnología de acceso a datos pensada principalmente para ambientes desconectados; esto quiere decir que al momento de recuperar los datos de una fuente de información se realiza la conexión, se transfieren los datos hacia la memoria del cliente mediante una consulta y se cierra la conexión. Los datos de la consulta pueden ser alterados y una vez que se termina de manipular dichos datos son enviados de vuelta hacia la fuente de datos para reflejar los cambios realizados.

Las estructuras de datos que permiten almacenar las consultas recibidas de la fuente de información se denominan DataSets, éstos son una representación en forma de tabla de tal como se ve la tabla en la base de datos. Un DataSet puede o no contener un esquema de validación similar al de la tabla en la base; este tipo de DataSet se denomina DataSet con tipo.

Los objetos encargados de actualizar y cargar los datos en un DataSet son los DataAdapters. Un DataAdapter actualiza los registros alterados mediante la lectura del estado de cada fila de la tabla del DataSet, aquellas filas que están marcadas como “nuevas”, “eliminadas” o “actualizadas” son sincronizadas con la base de datos para reflejar los cambios.

2.4.11. ASP.NET

ASP.NET es un marco de trabajo de programación generado en Common Language Runtime que puede utilizarse en un servidor para generar eficaces aplicaciones Web. ASP.NET ofrece varias ventajas importantes acerca de los modelos de programación Web anteriores:

Mejor rendimiento. ASP.NET es un código de Common Language Runtime compilado que se ejecuta en el servidor. A diferencia de sus predecesores, ASP.NET puede aprovechar las ventajas del enlace anticipado, la compilación just-in-time, la optimización nativa y los servicios de caché desde el primer momento. Esto supone un incremento espectacular del rendimiento antes de siquiera escribir una línea de código.

Compatibilidad con herramientas de primer nivel. El marco de trabajo de ASP.NET se complementa con un diseñador y una caja de herramientas muy completos en el entorno integrado de programación (Integrated Development Environment, IDE) de Visual Studio. La edición WYSIWYG, los controles de servidor de arrastrar y colocar y la implementación automática son sólo algunas de las características que proporciona esta eficaz herramienta.

Eficacia y flexibilidad. Debido a que ASP.NET se basa en Common Language Runtime, la eficacia y la flexibilidad de toda esa plataforma se encuentra disponible para los programadores de aplicaciones Web. La biblioteca de clases de .NET Framework, la Mensajería y las soluciones de Acceso a datos

se encuentran accesibles desde el Web de manera uniforme. ASP.NET es también independiente del lenguaje, por lo que puede elegir el lenguaje que mejor se adapte a la aplicación o dividir la aplicación en varios lenguajes. Además, la interoperabilidad de Common Language Runtime garantiza que la inversión existente en programación basada en COM se conserva al migrar a ASP.NET.

Simplicidad. ASP.NET facilita la realización de tareas comunes, desde el sencillo envío de formularios y la autenticación del cliente hasta la implementación y la configuración de sitios. Por ejemplo, el marco de trabajo de página de ASP.NET permite generar interfaces de usuario, que separan claramente la lógica de aplicación del código de presentación, y controlar eventos en un sencillo modelo de procesamiento de formularios de tipo Visual Basic. Además, Common Language Runtime simplifica la programación, con servicios de código administrado como el recuento de referencia automático y el recolector de elementos no utilizados.

Facilidad de uso. ASP.NET emplea un sistema de configuración jerárquico, basado en texto, que simplifica la aplicación de la configuración al entorno de servidor y las aplicaciones Web. Debido a que la información de configuración se almacena como texto sin formato, se puede aplicar la nueva configuración sin la ayuda de herramientas de administración local. Esta filosofía de "administración local cero" se extiende asimismo a la implementación de las aplicaciones ASP.NET Framework. Una aplicación ASP.NET Framework se implementa en un servidor sencillamente mediante la copia de los archivos necesarios al servidor.

No se requiere el reinicio del servidor, ni siquiera para implementar o reemplazar el código compilado en ejecución.

Escalabilidad y disponibilidad. ASP.NET se ha diseñado teniendo en cuenta la escalabilidad, con características diseñadas específicamente a medida, con el fin de mejorar el rendimiento en entornos agrupados y de múltiples procesadores. Además, el motor de tiempo de ejecución de ASP.NET controla y administra los procesos de cerca, por lo que si uno no se comporta adecuadamente (filtraciones, bloqueos), se puede crear un proceso nuevo en su lugar, lo que ayuda a mantener la aplicación disponible constantemente para controlar solicitudes.

Posibilidad de personalización y extensibilidad. ASP.NET presenta una arquitectura bien diseñada que permite a los programadores insertar su código en el nivel adecuado. De hecho, es posible extender o reemplazar cualquier sub-componente del motor de tiempo de ejecución de ASP.NET con su propio componente escrito personalizado. La implementación de la autenticación personalizada o de los servicios de estado nunca ha sido más fácil.

Seguridad. Con la autenticación de Windows integrada y la configuración por aplicación, se puede tener la completa seguridad de que las aplicaciones están a salvo.

2.5. Aplicaciones Distribuidas en .net [39]

Antes de que apareciera la plataforma de .net, las aplicaciones distribuidas estaban desarrolladas sobre COM/DCOM, con los problemas que esto acarrea, como son la recolección de basura distribuida, en la que se mantienen referencias a los objetos para determinar su tiempo de vida, de una manera en la que el cliente envía mensajes de actividad periódicamente, esta es la única manera de lograr que los objetos remotos sean destruidos de la memoria del servidor, y el protocolo binario de comunicaciones de DCOM. Además, debido a que se utiliza un protocolo de comunicaciones completo, usualmente es necesario mucho cuidado en la configuración de firewalls, si es que se va a implementar la aplicación sobre redes de área extendida, en las que se incluyan computadores de otras plataformas.

En la plataforma de .net se introdujeron nuevas formas de comunicación para el desarrollo de aplicaciones distribuidas. Debido a que dependiendo de las características específicas de la aplicación, se desarrollaron varias maneras, así cada aplicación puede utilizar la opción que más se ajuste a sus requerimientos. Las opciones que el .net framework ofrece a los desarrolladores son: .net remoting, Web services, y ASP.net.

Una de las formas más importantes de componentes distribuidos en la versión 1.1 de .net Framework es .net Remoting, que en realidad es el reemplazo para DCOM. Remoting permite que aplicaciones cliente creen instancias de

[39] http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp

objetos en computadores remotos, y sean utilizados por el cliente como componentes locales. En remoting, se puede configurar un componente remoto vía, código o utilizando archivos XML de configuración. Remoting puede comunicarse utilizando mensajes compactos binarios, o utilizando SOAP, para conexiones independientes de plataforma, además de ser completamente personalizable, si se desea se puede escribir métodos propios de comunicación.

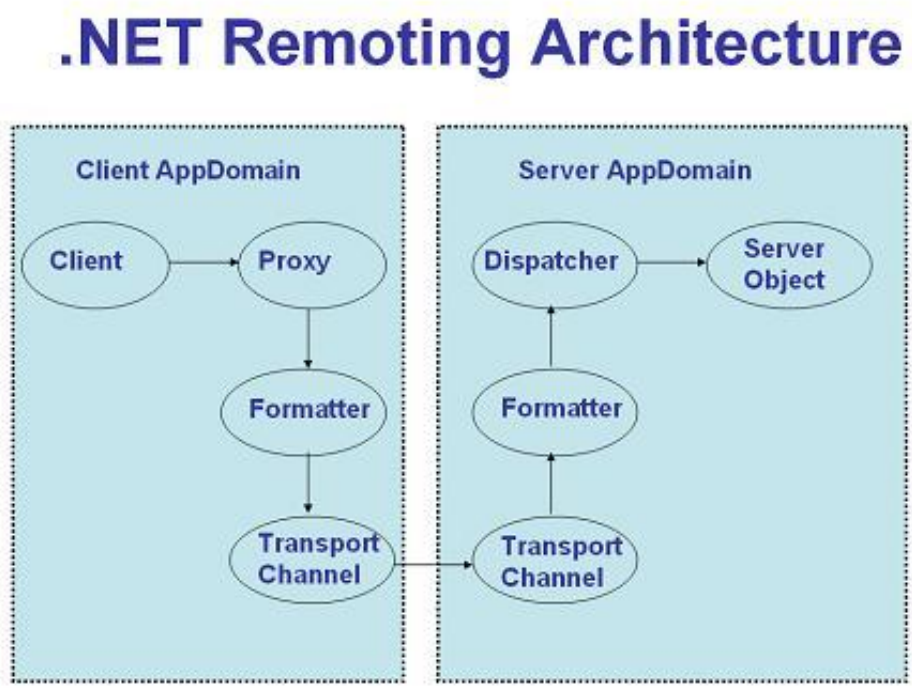


Figura 2-17: Arquitectura .Net Remoting[40]

Normalmente remoting es una solución para implementaciones en intranets, debido a que ofrece el mejor rendimiento si se utilizan canales de comunicación TCP y formatos binarios de transporte, además de flexibilidad, especialmente en

[40] http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp

casos en los que se necesita que los objetos remotos guarden los estados, o en caso de aplicaciones punto a punto.

La segunda forma y una de las más extendidas hoy en día son los XML Web Services, los servicios Web fueron concebidos para ambientes mucho más extensos que una intranet local, fueron diseñados para permitir la comunicación de componentes remotos en el Internet, además de buscar la interoperabilidad entre varias plataformas, lenguajes y sistemas operativos, es por esta razón que es relativamente sencillo el invocar a un servicio Web, desde un cliente de otra plataforma distinta a .net.

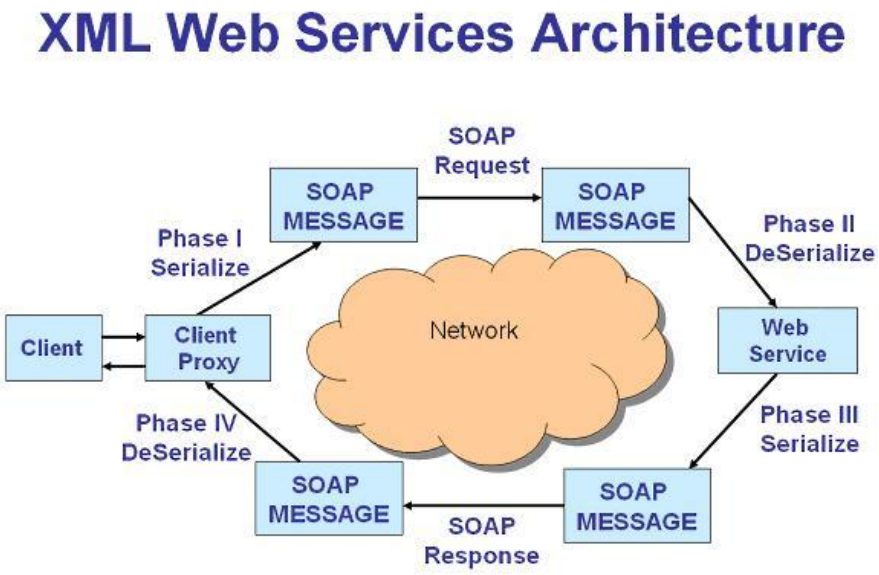


Figura 2-18: Arquitectura Web Services .Net[41]

El diseño de una aplicación distribuida implica la toma de decisiones sobre su arquitectura lógica y física, así como sobre la tecnología e infraestructura que

[41] http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp

se emplearán para implementar su funcionalidad. Para tomar estas decisiones, debe tener un conocimiento claro de los requisitos funcionales de la aplicación, así como los niveles de escalabilidad, disponibilidad, seguridad y mantenimiento necesarios.

2.5.1. Componentes y niveles en aplicaciones y servicios

Actualmente un principio ampliamente aceptado en el diseño de aplicaciones distribuidas es la división de la aplicación en componentes que ofrezcan servicios de presentación, empresariales y de datos. Los componentes que realizan tipos de funciones similares se pueden agrupar en capas, que en muchos casos están organizados en forma de apilamiento para que los componentes que se encuentran por "encima" de una capa determinada utilicen los servicios proporcionados por ésta, y un componente específico utilizará la funcionalidad proporcionada por otros componentes de su propia capa, y otras capas "inferiores", para realizar su trabajo.

Es importante tener en cuenta que las capas son simplemente agrupaciones lógicas de los componentes de software que conforman la aplicación, que ayudan a diferenciar entre los distintos tipos de tareas que realizan los componentes, facilitando la reutilización en la solución, además de que permiten a los equipos de desarrollo el enfocarse en tareas específicas.

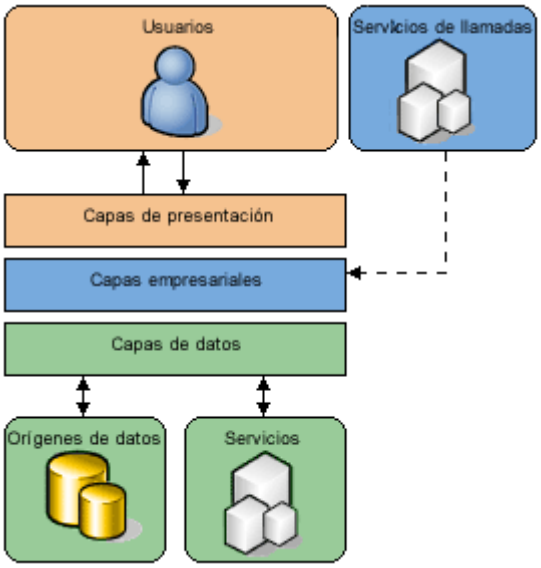


Figura 2-19: Componentes separados en capas según sus funciones[42]

Una solución distribuida posiblemente necesite trabajará con varias organizaciones o niveles físicos, en cuyo caso tendrá sus propias directivas en relación a la seguridad, administración y comunicaciones.

Aunque la lista que se muestra en la siguiente figura no es completa, representa los tipos de componentes de software más comunes encontrados en la mayoría de las soluciones distribuidas. A lo largo de este capítulo describiremos en profundidad cada uno de estos tipos.

[42] http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp

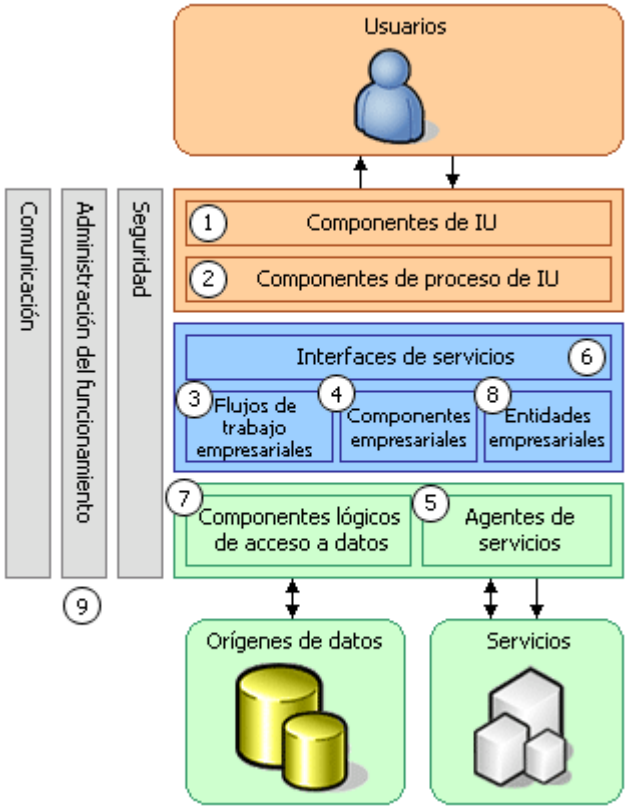


Figura 2-20: Componentes y niveles en aplicaciones y servicios[43]

2.5.1.1. Componentes de interfaz de usuario (IU)

Son los componentes responsables de ofrecer al usuario las interfaces necesarias para la interacción con la aplicación. En .net las interfaces de usuario se implementan ya sea en Windows Forms, o páginas ASP.NET usualmente, en los que se utiliza controles u otro tipo de tecnología que permita procesar y dar formato a los datos de los usuarios, así como adquirir y validar los datos entrantes procedentes de éstos.

[43] http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp

2.5.1.2. Componentes de proceso de usuario

En un gran número de casos, la interacción del usuario con el sistema se realiza de acuerdo a un proceso predecible. Para facilitar la sincronización y organización de las interacciones con el usuario, resulta útil utilizar componentes de proceso de usuario individuales. De este modo, el flujo del proceso y la lógica de administración de estado no se incluyen en el código de los elementos de la interfaz de usuario, por lo que varias interfaces podrán utilizar el mismo "motor" de interacción básica.

2.5.1.3. Flujos de trabajo empresariales

Una vez que el proceso de usuario ha recopilado los datos necesarios, éstos se pueden utilizar para realizar un proceso empresarial. Usualmente los procesos empresariales requieren la realización de varios pasos, los cuales se deben organizar y llevar a acabo en un orden determinado. El tiempo que este proceso puede tardar en completarse es indeterminado, por lo que sería preciso administrar las tareas necesarias, así como los datos requeridos para llevarlas a cabo.

2.5.1.4. Componentes empresariales

Independientemente de si el proceso empresarial consta de un único paso o de un flujo de trabajo organizado, la aplicación requerirá probablemente el uso de componentes que implementen reglas empresariales y realicen tareas empresariales. Los componentes empresariales implementan la lógica empresarial de la aplicación.

2.5.1.5. Agentes de servicios

Cuando un componente empresarial requiere el uso de la funcionalidad proporcionada por un servicio externo, se puede hacer uso de código para administrar este tipo de procesos. Los agentes de servicios permiten el separar la implementación específica del servicio externo, y de esta manera mantener a la aplicación independiente.

2.5.1.6. Interfaces de servicios

Para exponer lógica empresarial como un servicio, es necesario crear interfaces de servicios, que definen la forma en la que los componentes de la aplicación se comunican como son mensajes, formatos, protocolos, seguridad y excepciones, entre otros. Las interfaces de servicios también se denominan fachadas empresariales.

2.5.1.7. Componentes lógicos de acceso a datos

Casi siempre una aplicación necesita acceder a un almacén de datos en alguna parte del proceso. Es por esto que es necesario el aislar la lógica de acceso a datos en una capa independiente. Esto permite que la aplicación tenga centralizado el acceso en un solo punto lo que simplifica el mantenimiento o la migración hacia distintas fuentes de datos.

2.5.1.8. Componentes de entidad empresarial

Todas las aplicaciones requieren el paso de información entre las distintas capas y componentes. Las entidades empresariales que se utilizan de forma interna en la aplicación suelen ser estructuras de datos, como conjuntos de datos,

DataReader o secuencias de lenguaje de marcado extensible (XML), aunque también se pueden implementar utilizando clases orientadas a objetos personalizadas que representan entidades del mundo real necesarias para la aplicación.

2.5.1.9. Componentes de seguridad, administración operativa y comunicación

La aplicación probablemente utilice también componentes para realizar la administración de excepciones, autorizar a los usuarios a que realicen tareas determinadas y comunicarse con otros servicios y aplicaciones.

2.6. Arquitectura orientada a servicios (SOA)

La arquitectura orientada a servicios define un modelo de arquitectura para el desarrollo de aplicaciones distribuidas basado en la utilización de servicios que, a diferencia de la orientación a objetos, son débilmente acoplados y altamente interoperables sobre estándares tecnológicos abiertos.

Los servicios son pequeñas partes o porciones de funcionalidad que tienen sus propias reglas de negocio, datos, procedimientos de administración y operación relevantes a la solución del problema para el cual fueron diseñados.

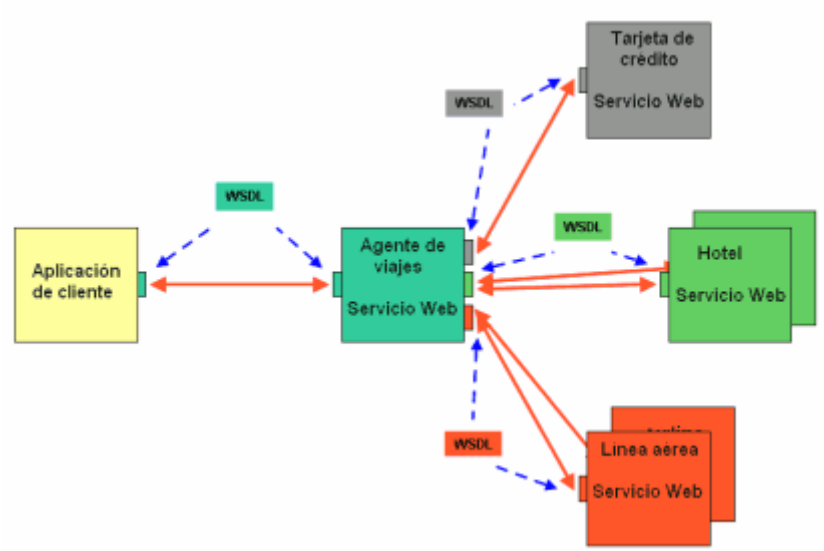


Figura 2-21: Interacción de servicios[44]

Un servicio debe ser autónomo e independiente, carece de una interfaz de usuario pero puede interactuar con otros servicios mediante una interfaz basada en mensajes. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación; con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio hecho en .net podría ser usado por una aplicación Java.

Elementos de la arquitectura orientada a servicios[45]

- **Operación**: Es la unidad de trabajo o procesamiento en una arquitectura SOA.
- **Servicio**: Es un contenedor de lógica. Estará compuesto por un conjunto de operaciones, las cuales las ofrecerá a sus usuarios. Los servicios son

[44] http://md2.dei.inf.uc3m.es:8000/PA/Practicas/Exposiciones/SOA.ppt
[45] http://arquitecturaorientadaaservicios.blogspot.com

auto-contenidos e independientes entre sí, con una tarea claramente definida cada uno.

- **Mensaje**: Para que un servicio pueda ejecutar una determinada operación, es necesario un conjunto de datos de entrada. A su vez, una vez ejecutada la operación, esta devolverá un resultado. Los mensajes son los encargados de encapsular esos datos de entrada y de salida que se envían los servicios unos a otros.
- **Proceso de negocio**: Son un conjunto de operaciones ejecutadas en una determinada secuencia (intercambiando mensajes entre ellas) con el objetivo de realizar una determinada tarea.

Estándares y conceptos de la arquitectura orientada a servicios[46]

- **XML**: Es un lenguaje de marcado capaz de describir distintos tipos de datos. Es un estándar ampliamente aceptado y utilizado medio de descripción de datos.
- **SOAP**: Es un protocolo de comunicación entre procesos basado en el intercambio de mensajes en formato XML dentro de una red. A su vez SOAP esta basado en XML y es completamente independiente de la plataforma y del lenguaje en el que estén implementados los procesos que se comunican.
- **WSDL**: Es un lenguaje basado en XML que permite describir servicios Web. Un documento WSDL especifica, entre otras cosas, dónde se

[46] http://md2.dei.inf.uc3m.es:8000/PA/Practicas/Exposiciones/SOA.ppt

encuentra el servicio así como las operaciones que pone accesibles a otros servicios.

- **UDDI**: Es un directorio, basado en XML, en el que las distintas empresas dan de alta servicios Web que ponen al servicio de otra empresas.

Implementación de una arquitectura SOA en .net

Una arquitectura SOA puede ser implementada usando cualquier tipo de infraestructura cliente/servidor, pero lo más común es que se implemente con servicios Web. Muchas veces se confunde una arquitectura orientada a servicios con servicios Web, sin embargo SOA es una visión de arquitectura tecnológica evolutiva, mientras que los servicios Web son elementos que permiten su implementación, no obstante se puede implementar SOA en cualquier protocolo de mensajería.

Cuadro 2-1: Protocolos de mensajería implementables en SOA[47]

Protocolo	Pros	Contras:
Remoting	Tiene el mejor rendimiento si se utiliza un serializador binario sobre TCP.	A menos que se utilice HTTP y un serializador SOAP, utiliza un protocolo propietario. Aunque existen productos como Ja.NET que permiten interoperabilidad con Java 1.2+. No tiene las capacidades de mensajería confiable, seguridad y enrutamiento de mensajes requeridas por SOA. Deben ser implementados manualmente. Promueven una arquitectura mas acoplada.
Web Services XML	Protocolo abierto (independiente de plataforma) Independiente de ubicación Fácil de implementar Promueven una arquitectura desacoplada.	Su rendimiento sobre la red no es el mejor (en comparación con un protocolo binario). Debido a que todo el flujo es texto. La serialización/deserialización en los extremos también contribuye a su pobre desempeño. Aunque cuenta con una manera de hacer llamados asincrónicos, no tiene todas las capacidades de mensajería requeridas por SOA.
Web Services XML + Web Services Enhancements (WSE) 2.0	Permite configurar un canal binario sobre el cual se va toda la mensajería SOAP. Provee muchas de las características requeridas por SOA implementando las especificaciones WS-Security, WS-Routing, WS-ReliableMessaging, entre otras.	Aunque basado en estándares, aun no existen muchas implementaciones diferentes a la de Microsoft. Esto complica la interoperabilidad. Aun falta madurez en las implementaciones actuales.
MSMQ	Por ser un protocolo de mensajería asincrónica basado en colas de mensajes, ya incluye las características que brindan confiabilidad y desacoplamiento a la transmisión de los mensajes.	Aunque existen puentes con otros sistemas de mensajería basada en colas (como MQSeries de IBM), es un protocolo cerrado.

[47] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art143.asp

CAPÍTULO III

3. ESTUDIO DE LOS PATRONES DE SOFTWARE

3.1. Clasificación de los patrones de software

Uno de los posibles criterios para la clasificación de los patrones de software es según su nivel de abstracción y detalle, en esta clasificación los patrones son divididos en tres grupos principales:

- Patrones Arquitecturales
- Patrones de Diseño
- Patrones de Programación (Idiomas)

Cuadro 3-1: Clasificación de los patrones de software[48]

1. Patrones Arquitecturales	
POSA Patterns oriented software architecture	Distributed Systems Interactive Systems Adaptable Systems
PEAA: Patterns of Enterprise Application Architecture	Domain Logic Patterns Mapping to Relational Databases Patterns Web Presentation Patterns Distribution Patterns Offline Concurrency Patterns Session State Patterns Base Patterns

[48] Elaboración Propia

2. Patrones de Diseño
Creacionales De comportamiento Estructurales
3. Patrón de Programación (Idiomas)

3.1.1. Niveles de Patrones

La clasificación utilizada de los patrones se basa en el nivel de abstracción y detalle:

- Los patrones de software de mayor abstracción (Patrones de Arquitectura) se enfocan en la estructura del sistema definiendo su organización por completo.
- Los patrones con un menor nivel de abstracción (Patrones de Diseño) definen la forma en la que los pequeños componentes definidos por los patrones arquitecturales deben ser implementados. Aún a este nivel no se utilizan características específicas de un lenguaje de programación, aunque se hace referencia a capacidades como herencia o polimorfismo de un lenguaje para que su implementación sea correcta.
- Los patrones con menor nivel de abstracción (Idiomas) toman ya la sintaxis específica de un lenguaje para la codificación, definiendo como debe ser programado un componente en un lenguaje específico.

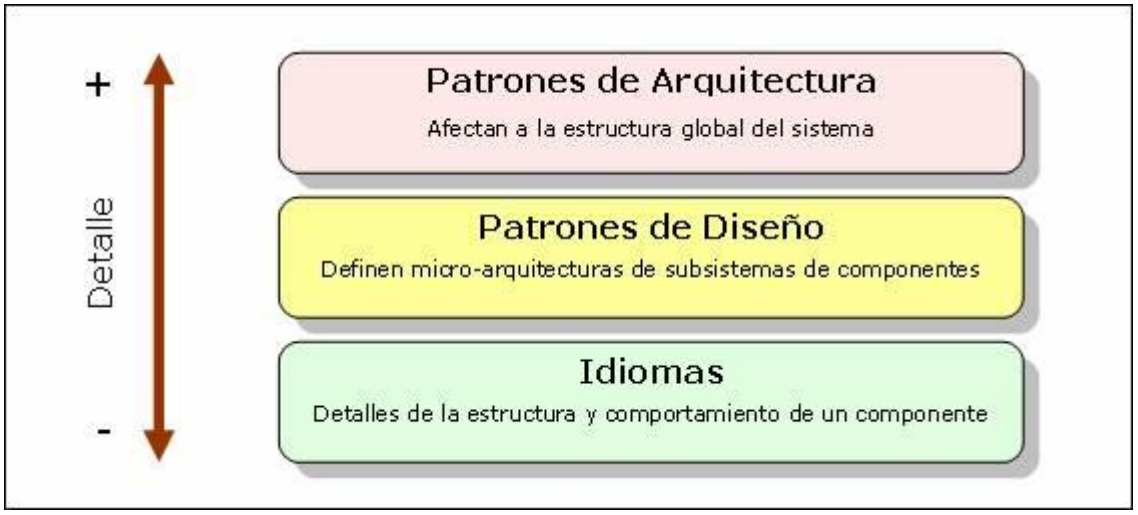


Figura 3-1: Niveles de patrones[49]

3.1.2. Estructura de los Patrones

- **Nombre**: un nombre comprensivo que sea para un vocabulario común.
- **Intención**: sumariza el patrón en una o dos oraciones.
- **Sketch / Representación**: representación gráfica del patrón, generalmente UML pero no siempre.
- **Motivación / Descripción**: problema principal que intenta resolver un patrón.
- **Funcionamiento**: describe la solución.
- **Utilidad**: describe cuándo un patrón debe ser utilizado. Se puede plantear de la forma: Cuando no debe ser utilizado, para tener más alternativas cuando se trata de patrones similares (front y page controller)

[49] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/mtj_2864.asp

- **Ejemplo**: un ejemplo no representa el patrón como tal, es una implementación del patrón para mejor comprensión. El código utilizado no debe ser tratado como un código master que representa el patrón.

3.2. Patrones de Programación de Software

Los patrones de Software para el estudio los podemos dividir en dos categorías, los patrones de diseño, y patrones de programación.

3.2.1. Patrones de Diseño[50]

Diseñar software orientado a objetos es difícil, y diseñar software reutilizable orientado a objetos es más difícil aún. Se deben encontrar los objetos que sean pertinentes, encapsular los objetos en clases con un grado de granularidad correcto, definir las interfaces de las clases, las jerarquías de herencias, y establecer las relaciones entre los diferentes componentes. Un diseño debe ser específico al problema que se está tratando, pero también debe ser lo suficientemente general para poder ser utilizado en futuras ocasiones y requerimientos. Además de buscar eliminar el rediseño o por lo menos reducirlo considerablemente.

[50] E. Gamma, R. Helm, R. Johnson, J. Vlissides., Design Patterns: Elements of Reusable Object Oriented Software

Al reutilizar soluciones que han funcionado en el pasado para resolver cierto tipo de problemas, se encuentran patrones recurrentes de clases y objetos que se comunican entre ellos en muchos sistemas orientados a objetos. Estos patrones resuelven problemas de diseño específicos, y hacen los diseños orientados a objetos, más flexibles, elegantes y por último reutilizables.

Los patrones de diseño tienen 4 partes fundamentales, que son:

Nombre del Patrón, de alguna manera el nombre del patrón describe un problema de diseño, sus soluciones, y consecuencias en una o dos palabras.

El problema, describe cuando se debe aplicar el patrón. Explica el problema de diseño y su contexto. Algunas veces describe también las condiciones que se deben cumplir antes de que se pueda aplicar el patrón.

La solución describe los elementos que contienen el diseño, sus relaciones, y responsabilidades. La solución no describe un diseño concreto particular, ni su implementación.

Las consecuencias son el resultado de la utilización de la aplicación de un patrón, estos resultados pueden ser favorables o también desfavorables, estas consecuencias son críticas para evaluar alternativas de diseño y para comprender los costos y beneficios de la aplicación de un patrón. Las consecuencias en el software usualmente se refieren a velocidad, y espacio. Debido a que la reutilización es un factor en los diseños orientados a objetos, las consecuencias

de un patrón incluyen el impacto dentro de la flexibilidad, extensibilidad, o portabilidad de un sistema.

3.2.2. Clasificación de los Patrones de Diseño[51]

Para este estudio de los patrones de diseño se tomará como base la clasificación dada por GoF, en la que se dividen a los patrones de diseño en tres categorías relacionadas con el propósito del patrón, estas categorías son: Creacionales, Estructurales y de Comportamiento.

El segundo criterio de clasificación es el ámbito, y se refiere a si el patrón es aplicable principalmente a clases u objetos. Los patrones de clases manejan las relaciones existentes entre las clases y las subclases. Estas relaciones son establecidas mediante herencia, entonces son relaciones estáticas definidas en tiempo de compilación. Los patrones de objetos manejan las relaciones entre objetos, que pueden ser cambiadas en tiempo de ejecución y son más dinámicas.

- **Creacionales**: Abstraen el proceso de creación de instancias de objetos. Ayudan a hacer a un sistema independiente de cómo se crean, se componen y se representan sus objetos.
- **Estructurales**: Tratan con la composición de clases u objetos. Se ocupan de cómo las clases y objetos son utilizados para componer estructuras de mayor tamaño.

[51] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- **De Comportamiento**: Caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad. Atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

En la siguiente tabla se muestra la distribución de los 24 patrones en las categorías comentadas anteriormente:

Cuadro 3-2: Clasificación de los patrones de diseño[52]

		Creacionales	**Estructurales**	**De Comportamiento**
Ámbito	Clase	• Factory method	• Adapter	• Interpreter • Template method
	Objeto	• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

[52] http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp

3.2.2.1. Patrones Creacionales

3.2.2.1.1. Factory method [53]

Descripción

Define una interfase para la creación de un objeto, pero deja decidir a las subclases que clase es la que se va a instanciar. Factory method, permite a una clase delegar la instanciación a subclases. Es también conocido como Constructor Virtual. El nombre Factory method está dado ya que la clase que delega la creación de objetos tiene un método que deberá ser sobrecargado a fin de que sean creados a través de este método los objetos correctos.

Los diferentes componentes que participan en este patrón de diseño son los siguientes

- Producto: Define la interfase de los objetos que el Factory method crea.
- Producto Concreto (ConcreteProduct): Implementa la interfase de producto.
- Creador: Declara el Factory method, que retorna un objeto de tipo Producto, el creador además puede definir una implementación por defecto del Factory method que retorna un Producto Concreto por omisión.

[53] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- Creador Concreto (ConcreteCreator): Sobrecarga el Factory method para retornar una instancia del producto concreto.

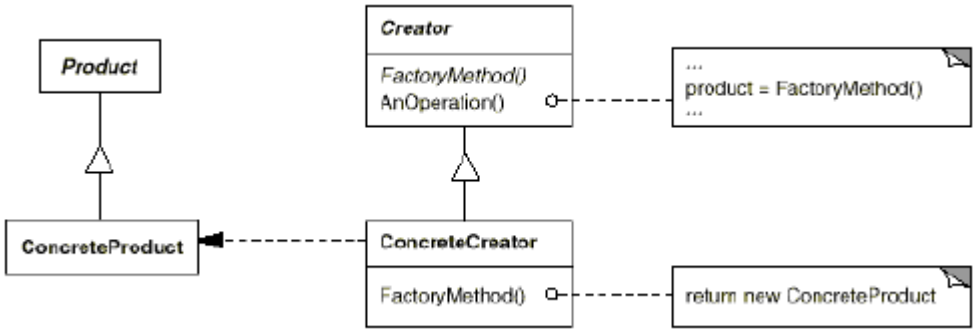


Figura 3-2: Factory method[54]

Los Factory Methods eliminan la necesidad de agregar clases específicas de una aplicación dentro del marco de trabajo. El código solamente maneja una interfase de Producto, y a partir de esto puede manejar cualquier implementación específica de un Producto.

Utilidad

El patrón de Factory method se puede usar cuando:

- Una clase no puede anticipar el tipo de objetos que debe crear
- Una clase quiere que sus subclases sean las que especifican los objetos que deben crear
- Una clase delega responsabilidades a una o más subclases de apoyo, para la creación de objetos específicos.

[54] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

3.2.2.1.2. Abstract Factory [55]

Descripción

Abstract Factory provee una interfase para crear familias de objetos relacionados o dependientes, sin especificar sus clases concretas, este patrón también es conocido como “Kit”, debido a que muchas veces se denominan de esta manera las implementaciones de este patrón, como por ejemplo WidgetKits, son utilizados para crear interfaces de usuario de temas específicos, en las que se necesita que todos los elementos de la interfase sean de un tipo en particular, pero existen varios tipos.

Abstract Factory, elimina el problema de codificación específica para un tipo, y hace posible que se pueda crear cualquier tipo de objetos relacionados, utilizando una interfase consistente para todos los tipos, sin conocer las clases específicas que se deben instanciar

Los componentes que participan en este patrón son:

- AbstractFactory: Declara una interfase con operaciones que crean productos abstractos.
- ConcreteFactory: Implementa las operaciones de crear productos concretos.

[55] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- AbstractProduct: Declara una interfase para un tipo de objeto producto.
- ConcreteProduct: Define un objeto producto que será creado por el ConcreteFactory correspondiente
- Client: Utiliza solamente las interfaces declaradas por AbstractFactory y AbstractProduct.

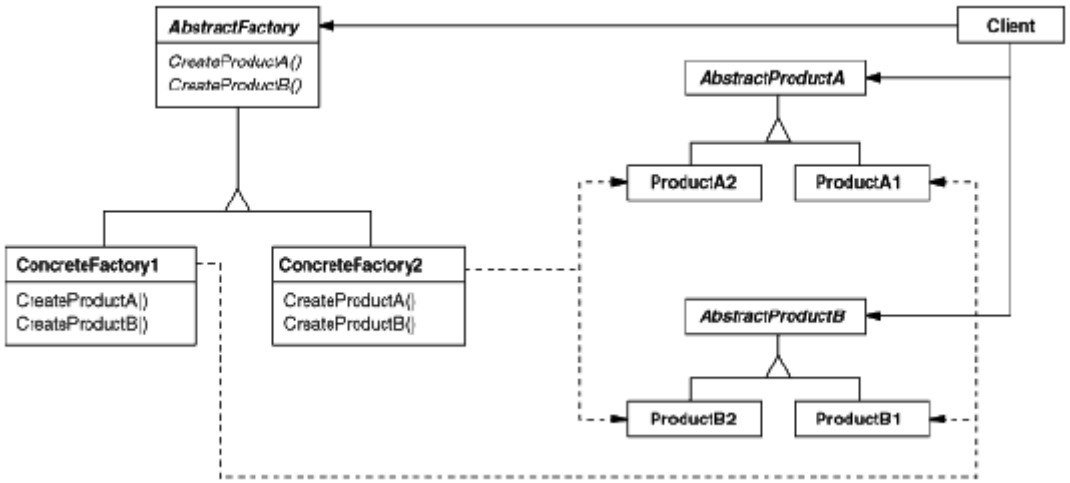


Figura 3-3: Abstract factory[56]

Utilidad

El patrón de Abstract Factory puede ser utilizado cuando:

- Un sistema debe ser independiente de cómo son creados sus productos, de cómo son compuestos o representados.
- Un sistema debe ser configurado con una o más familias de productos
- Una familia de objetos relacionados, está diseñada para ser utilizada conjuntamente, y es necesario que se cumpla esta condición.

[56] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- Se quiere proveer una librería de clases de productos, y se necesita revelar solamente sus interfaces, no las implementaciones.

El patrón de Abstract Factory tiene los siguientes beneficios y desventajas:

Aísla las clases concretas. Ayuda a controlar las clases de los objetos que una aplicación crea. Debido a que se encapsula la responsabilidad y el proceso de crear objetos, se aísla del cliente la implementación de las clases. Los clientes únicamente manipulan las instancias por medio de sus interfaces abstractas. Los nombres de las clases reales son completamente separados en la implementación de un ConcreteFactory, y nunca aparecen estos nombres en el código cliente.

Hace fácil el cambiar familias de objetos, debido a que en el código la clase de un ConcreteFactory solo está en una línea de código, donde es instanciada, hace fácil el cambiar una familia de productos ya que solamente es necesario cambiar el ConcreteFactory, y todos los objetos relacionados cambian inmediatamente.

Muchas veces se diseñan objetos para que trabajen juntos, es importante que una aplicación utilice objetos de una sola “familia” a la vez, este patrón de diseño hace que esta condición sea cumplida ya que las fábricas crean todos los objetos de una familia.

Una desventaja del patrón es la complejidad para cambiar la familia de objetos que un ConcreteFactory maneja, ya que se debe cambiar todas las clases abstractas y sus interfaces, además de todas las subclases.

3.2.2.1.3. Builder[57]

Descripción

El patrón Builder, tiene como objetivo el separar la construcción de un objeto complejo de su representación, de manera que el mismo procedimiento para la creación de objetos pueda ser utilizado para crear distintas representaciones. De esta manera se trata de que un mismo proceso para una determinada información pueda crear diferentes representaciones de la información. Por ejemplo en un convertidor de textos, en el que se necesita convertir el texto a varios formatos, este patrón de diseño hará que la misma manera de leer un texto produzca como resultado muchos formatos distintos, dependiendo de las necesidades del usuario, sin alterar la manera en la que la información es leída, y sin importar tampoco lo complicada que sea la traducción a un formato específico.

Los componentes que este patrón de diseño utiliza son:

- Builder (Constructor): Especifica una interface abstracta para la creación de las partes de un objeto producto.

[57] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- ConcreteBuilder (Constructor Concreto): Construye y ensambla las partes del objeto producto implementando la interfaz de Builder, además define y mantiene registro de la representación que este constructor crea, y provee una interface para recuperar el producto creado.
- Director: Construye un objeto por medio de la interface que Builder provee.
- Producto: Representa un objeto que se encuentra bajo construcción, la representación y la forma de ensamblado del producto está definida por ConcreteBuilder.

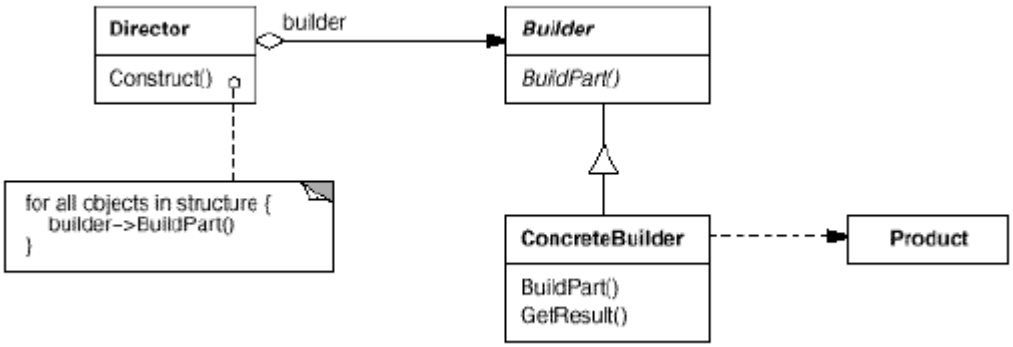


Figura 3-4: Builder[58]

Utilidad

Los usos conocidos para este patrón de diseño se encuentran en convertidores de texto, compiladores, parsers de scripts. De esta manera permite que se varíe la representación del producto y si se desea cambiar esta

[58] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

representación o añadir nuevas, lo que se debe hacer es cambiar el constructor concreto, o implementar uno nuevo.

Aísla el código de construcción del código de representación, de esta manera se mejora la modularidad de un sistema, encapsulando la manera en la que un objeto complejo es creado y representado, no es necesario que los clientes conozcan como esta implementado el producto ya que estas clases no aparecen en la interface proporcionada por el constructor.

Cada uno de los constructores concretos contiene todo el código necesario para crear y ensamblar una clase particular de producto. El código es escrito una vez, a partir de esto varios directores pueden reutilizar el constructor para construir variantes del producto.

3.2.2.1.4. Prototype[59]

Descripción

Especificar los objetos que se deben crear a partir de una instancia prototipo, y los siguientes objetos se crear a partir de realizar copias de este prototipo. De esta manera se puede definir los tipos de objetos que se deben crear a partir de un solo prototipo, cuando el tipo de objetos que se deben crear es definido en tiempo de ejecución y no en tiempo de compilación.

[59] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Si en una aplicación se crea un objeto prototipo, y este objeto soporta operaciones de copia, a partir de este objeto creado se puede crear más objetos iguales y luego modificarlos. De esta manera es fácil incluir en un programa nuevos tipos de objetos prototipo.

Los participantes de este patrón son:

- Prototype (Prototipo): Declara una interface para clonarse a si mismo.
- ConcretePrototype (Prototipo Concreto): implementa la operación para copiarse a sí mismo.
- Client (Cliente): Crea un objeto nuevo por medio de la operación de copia del prototipo.

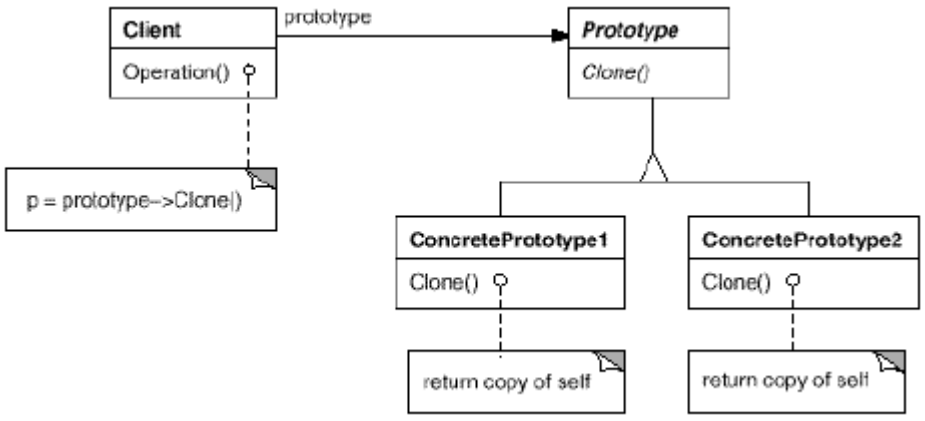


Figura 3-5: Prototype[60]

[60] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Utilidad

El prototipo se utiliza principalmente en casos en los que se necesita agregar o remover productos en tiempo de ejecución, de esta manera se puede cargar clases dinámicamente en un programa para su utilización. Y crear estructuras complejas de objetos utilizando la composición de objetos registrando derivados de un prototipo en un cliente, que proporcione nueva funcionalidad al cliente cuando se le delega el control.

3.2.2.1.5. Singleton[61]

Descripción

Singleton sirve para asegurarse que en cualquier momento de la ejecución de un programa, exista solamente una instancia de una clase, y proveer un punto de acceso global a esa instancia. Es importante en algunos casos que solamente exista una instancia de una clase para poder mantener los datos que se manejan completamente consistentes.

Este patrón de diseño es ampliamente utilizado, incluso en la tecnología remoting de .net existe un soporte nativo para este tipo de objetos. Se puede utilizar este patrón de diseño cuando debe existir exactamente una instancia de una clase dada, y esta instancia debe ser accesible globalmente por los clientes

[61] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

desde un punto de acceso conocido, además cuando la clase puede ser extendida mediante herencia, y todos los clientes deben ser capaces de utilizar la instancia extendida sin modificar el código del cliente.

Los componentes participantes en este patrón de diseño son:

- Singleton: Define una método de instancia, que permite a los clientes acceder a la única instancia de la clase.

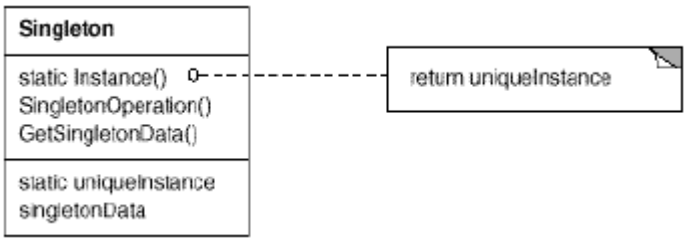


Figura 3-6: Singleton[62]

Utilidad

Los beneficios de la utilización del patrón Singleton son:

Acceso controlado a una instancia única: Debido a que la clase de Singleton encapsula el acceso a una instancia de la clase, se puede mantener un control estricto acerca de cómo, cuando, y quien accede a los recursos de la clase.

[62] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Reduce los espacios de nombres, es una mejora con respecto a las variables globales debido a que se reduce el nombre de variables que se utilizan en un espacio de nombres.

Permite el mejorar las operaciones, debido a que una clase singleton puede ser extendida utilizando mecanismos como herencia, se puede configurar una aplicación para que utilice las clases derivadas, incluso se puede llegar a configurar las clases que se utilizan en tiempo de ejecución. Permitiendo así mayor flexibilidad.

Permite un número variable de número de instancias, este patrón permite cambiar el número de instancias que se permiten, incluso se puede utilizar variaciones de este patrón de diseño para controlar el número de instancias de una clase en una aplicación.

3.2.2.2. Patrones Estructurales

Los patrones estructurales se preocupan de cómo las clases y objetos se componen para formar estructuras más grandes. Los patrones estructurales de clase utilizan herencia para componer los objetos, por ejemplo la herencia múltiple logra que 2 o más clases den como resultado una con las características combinadas de sus clases padre, esto es muy utilizado cuando se necesita que librerías de clases distintas, y desarrolladas independientemente trabajen conjuntamente.

Los patrones estructurales de objetos a su vez, describen la forma en la que los objetos se componen para ofrecer funcionalidad nueva, la flexibilidad de la composición de objetos viene de la habilidad de cambiar esta composición en tiempo de ejecución, lo que es imposible de lograr con patrones estructurales de clase, ya que las relaciones se dan mediante herencia y no por composición.

3.2.2.2.1. Adapter[63]

Descripción

El propósito de este patrón de diseño como es definido en el libro de Patrones de Diseño de GoF, es utilizado para convertir una interface de una clase en otra interface que los clientes esperan. El adaptador permite que clases trabajen conjuntamente que de otra manera no sería posible debido a la incompatibilidad de las interfases. Este patrón de diseño es también llamado wrapper.

Muchas veces componentes que fueron pensados para ser reutilizados no pueden serlo debido a que las interfases no son compatibles con la nueva aplicación. Una solución es modificar las interfases del componente, pero no es correcto, ya que el componente no debería ser cambiado para adaptarse a una aplicación específica, en cambio lo que se puede hacer es definir una interface que adapte la existente del componente a lo que el cliente requiere, esto puede ser implementado de dos maneras: primero, utilizando herencia múltiple entre la

[63] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

interface que el cliente espera y el componente, y segundo usando composición para implementar la interface del cliente en términos del componente. Estas dos alternativas corresponden al adaptador de clase, y de objeto respectivamente.

Cuando se utiliza la versión del adaptador de objeto además se tiene la ventaja que se puede reutilizar varias clases sin tener que crear clases derivadas por cada una.

Los componentes que participan en este patrón de diseño son los siguientes:

- Target (Objetivo): Define la interface específica del dominio de la aplicación que los clientes utilizan.
- Client (Cliente): Utiliza los objetos que corresponden con la interface objetivo.
- Adaptee (Adaptado): Define una interface existente que necesita ser adaptada.
- Adapter (Adaptador): Adapta la interface del adaptado a la interface objetivo.

Adaptador de Clase

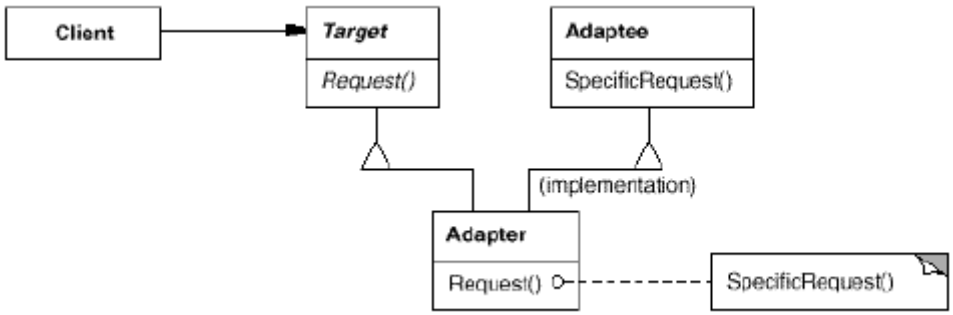


Figura 3-7: Adaptador de clase[64]

Adaptador de Objeto

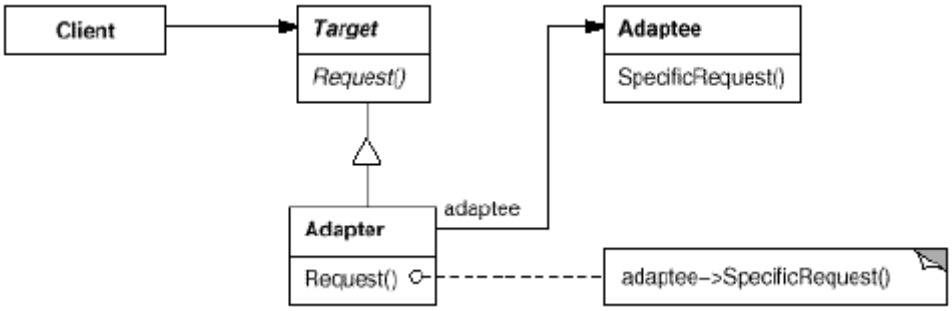


Figura 3-8: Adaptador de objeto[65]

Utilidad

Existen diferentes consecuencias en la utilización del adaptador de objetos y de clases. Para el adaptador de clases debido a que se crea una clase derivada, una clase adaptador por ejemplo no será utilizable cuando se desee adaptar una

[64] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*
[65] Íd., Ibíd.

clase y todas las clases derivadas. Debido a que el adaptador es una clase derivada, es posible modificar parte del comportamiento del componente original.

Al contrario un solo adaptador de objetos puede trabajar con muchos componentes, como por ejemplo todas las clases derivadas, incluso puede agregar funcionalidad a todas las clases con una sola implementación. En cambio al contrario del adaptador de clase hace más difícil el modificar el comportamiento del componente original, esto requeriría que se cree una clase derivada del componente original y en el adaptador hacer referencia a la clase derivada y no a la original.

3.2.2.2.2. Bridge[66]

Descripción

El patrón de diseño bridge o puente intenta separar la abstracción de un objeto de cómo está implementado de tal manera que ambas partes puedan variar independientemente. Usualmente una abstracción puede tener varias implementaciones, y la manera usual de manejar esto es utilizando herencia, de esta manera una clase abstracta define la interface de la abstracción, y las clases derivadas implementan esto de la manera correcta. Pero esta forma no siempre es lo suficientemente flexible, la herencia une permanentemente a las dos partes y esto hace que sea complicado el modificar las partes libremente.

[66] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

A continuación se describe los diferentes componentes que forman el patrón de diseño bridge.

- Abstraction (Abstracción): Define la interface del objeto abstracto, y mantiene una referencia al objeto de tipo Implementación.
- RefinedAbstraction (Abstracción Refinada): Extiende la interface definida por la abstracción.
- Implementor (Implementación): Define la interface de las clases de implementación, la interface no tiene que corresponder exactamente con la interface de la abstracción. Usualmente en la implementación se definen solamente operaciones primitivas, y la abstracción define operaciones de alto nivel basadas en estas primitivas.
- ConcreteImplementor (Implementación Concreta): Implementa la interface definida por Implementor y define una clase concreta.

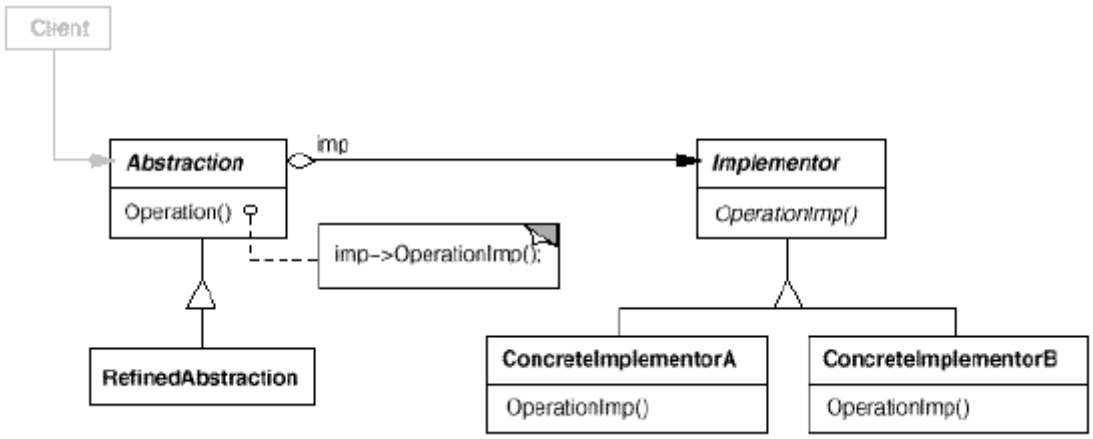


Figura 3-9: Bridge[67]

[67] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Utilidad

Se puede utilizar este patrón de diseño cuando se necesite que una abstracción y su implementación no estén ligadas estáticamente, por ejemplo cuando se requiera cambiar esta relación en tiempo de ejecución. Uno de los beneficios es que cuando se realizan cambios en la implementación, este cambio no debe tener impacto en el cliente, es decir el código del cliente no debería tener que recompilarse para que los cambios tengan efecto. Esta es una característica esencial cuando se necesita asegurarse de que exista compatibilidad binaria entre diferentes versiones de una librería de clases.

3.2.2.2.3. Composite[68]

Descripción

El objetivo de este patrón de diseño es el ocultar al cliente como está compuesto un objeto, y tratar a objetos compuestos o individuales de la misma manera. Este patrón se basa en la composición de objetos recursiva de manera que el cliente manipule de igual manera un objeto simple que uno compuesto.

[68] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Los componentes participantes en este patrón de diseño son:

- Component (Componente) Declara la interface para los objetos que realizan la composición, implementa además el comportamiento por defecto de la interface, que será común para todas las clases, incluye una interface para acceder y manejar los componentes hijos. Opcionalmente puede declarar una interface para acceder a su componente padre, en la estructura recursiva.
- Leaf (Hojas): Representan los objetos situados en los extremos de la composición. Una hoja no tiene elementos hijos, define también el comportamiento para los elementos básicos de la composición.
- Composite (Objeto Compuesto): Define el comportamiento de los objetos que tienen hijos, almacena las referencias a los objetos hijos, implementa operaciones relacionadas a los hijos en la interface de componente (Component).
- Client (Cliente): Manipula los objetos de la composición a través de la interface del componente.

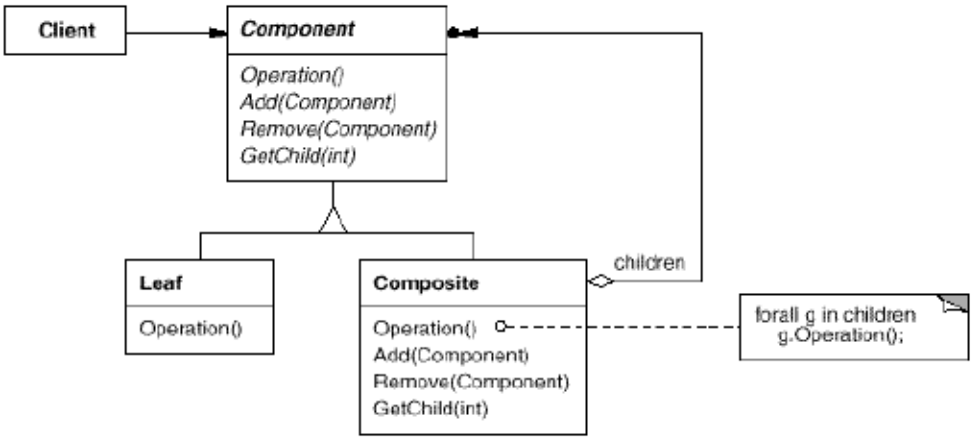


Figura 3-10: Composite[69]

[69] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Una estructura típica de un objeto compuesto utilizando Composite se podría ver de la siguiente manera.

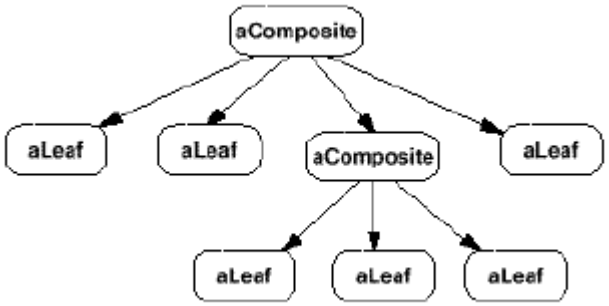


Figura 3-11: Estructura de un objeto compuesto usando composite[70]

Utilidad

Este patrón de diseño es especialmente útil cuando se necesita representar jerarquías de objetos de la forma parte-todo, y cuando se necesita que el cliente pueda manipular un objeto sin importar la diferencia entre los objetos simples y los compuestos.

3.2.2.2.4. Decorator[71]

Descripción

El propósito de este patrón de diseño es el lograr agregar funciones adicionales a un objeto de forma dinámica. Decorator, provee una alternativa

[70] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*
[71] Íd., Ibíd.

flexible a la herencia para extender funcionalidad es conocido también como Wrapper.

Muchas veces se desea agregar funcionalidad a objetos individuales de una clase determinada, no siempre se desea hacer esto con toda la clase. Una forma de agregar funcionalidad a un objeto determinado es utilizando herencia, pero el problema de esto es que de esta manera las relaciones o la funcionalidad agregada al objeto está ligada de una forma estática, y el objeto no puede “elegir” que tipo de funcionalidad quiere, ni cuando la quiere. La segunda opción es encapsular el componente en otro que agregue la funcionalidad deseada.

El objeto que encapsula al componente original es llamado Decorador, el decorador cumple con la interface del componente que encapsula de manera que sea transparente para el cliente, de esta manera el decorador reenvía las peticiones al componente y realiza tareas adicionales cuando es necesario. Esta transparencia permite que sean agregados decoradores a un componente de manera recursiva, de manera que se puedan agregar no solo una sino muchas funciones extras al componente.

Las distintas partes de este patrón son:

- Component(Componente): Define la interface de los objetos que pueden tener funcionalidad agregada dinámicamente.
- ConcreteComponent (Componente Concreto): Define un objeto al que se pueden agregar funciones.

- Decorator (Decorador): Mantiene una referencia al objeto componente y define una interface que cumple con la interface del componente.
- ConcreteDecorator (Decorador Concreto): Agrega funcionalidad al componente.

La estructura del patrón de diseño se muestra en la siguiente figura.

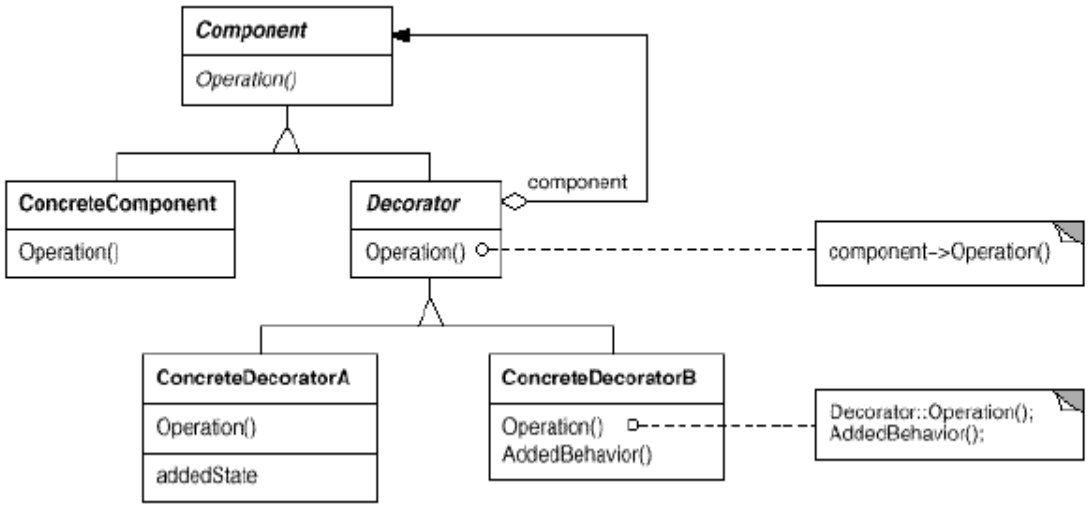


Figura 3-12: Decorator[72]

Utilidad

La mayor utilidad de este patrón de diseño es el agregar funcionalidad a objetos individuales, dinámicamente, transparentemente, sin cambiar interfases de clientes, y sin ligar permanentemente a los objetos. Otra opción para la utilización de este patrón es cuando se necesita introducir un gran número de

[72] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

nuevas funciones, por lo que significaría demasiadas clases si se utiliza herencia, además de ser extensiones independientes que no siempre van a ser utilizadas.

3.2.2.2.5. Facade[73]

Descripción

La intención de Facade es el proveer una interface unificada a un conjunto de interfases para que el sistema sea más simple de utilizar.

El estructurar un sistema y separarlo en bloques de procesamiento más pequeños ayuda a reducir la dificultad del desarrollo del mismo, usualmente se intenta eliminar lo más posible los puntos de interconexión de sistemas para hacerlos más simples, Facade intenta alcanzar esta meta, simplificando las interfases de un sistema, en una sola interface de manejo más simple.

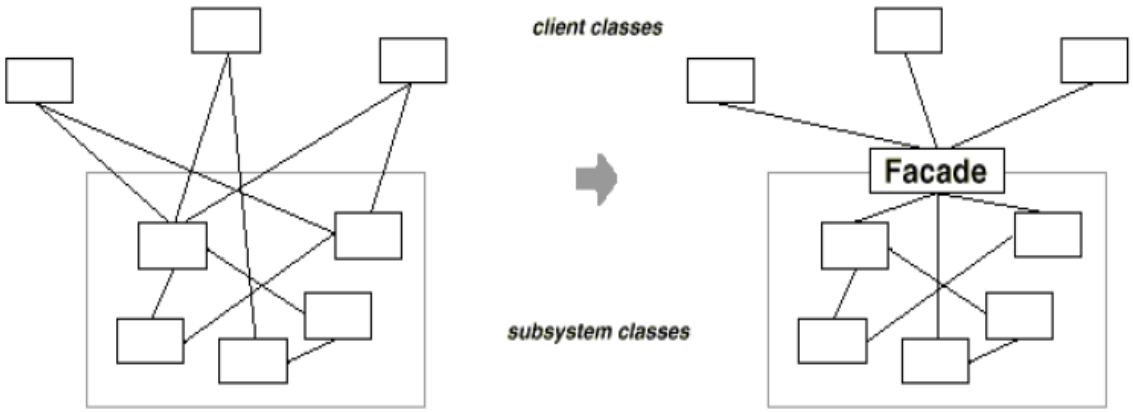


Figura 3-13: Facade[74]

[73] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

De esta manera lo que se intenta hacer es que los usuarios de una librería de clases, tengan acceso a un punto centralizado en el que se encapsule toda la funcionalidad que normalmente es requerida, para simplificar las llamadas, mientras que para usuarios que necesiten realizar tareas más especializadas, se mantenga toda la funcionalidad por medio de las demás interfases.

Componentes Participantes

- Facade (Fachada): Conoce qué clases de un subsistema son responsables de una petición, y delega las peticiones de los clientes a los objetos apropiados del subsistema.
- Subsystem Classes (Clases del Subsistema): Implementan la funcionalidad del subsistema, pero no mantienen ninguna referencia a la fachada, es decir la existencia de una fachada debe ser transparente para estas clases.

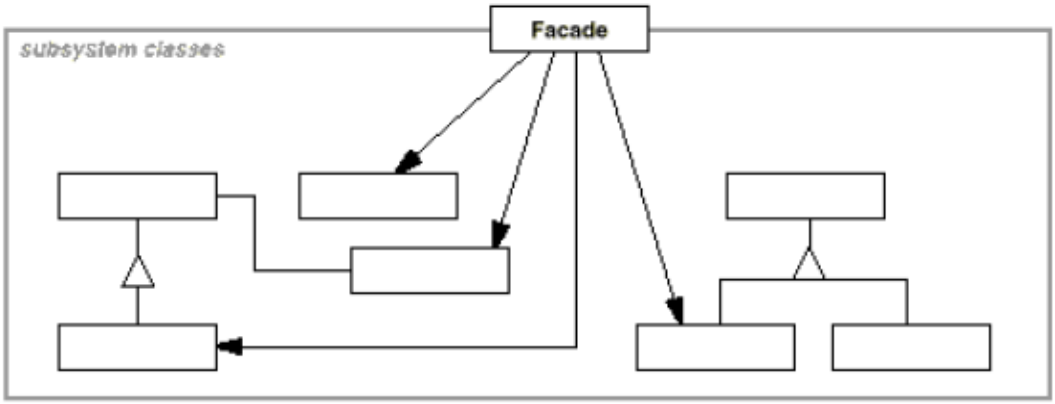


Figura 3-14: Facade (2)[75]

[74] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*
[75] Íd., Ibíd.

Utilidad

Se debe utilizar Facade cuando se necesita proveer a los clientes de una interface simple de manejo de un subsistema, sin afectar a la flexibilidad del subsistema, en caso de que un cliente desee realizar tareas que no son comunes. Usualmente los subsistemas se vuelven cada vez más complejos mientras siguen evolucionando. Incluso la aplicación de Patrones de Diseño en el desarrollo de un subsistema implica que se va a tener un mayor número de clases, esto provoca que un sistema sea más escalable o extensible, sin embargo esto hace también que los clientes que no necesitan extenderlo tengan que manejar un mayor número de interfases haciéndolo más complicado.

Cuando hay muchos puntos de dependencia entre los clientes y las clases de implementación. Al introducir este patrón de diseño, se logra eliminar la dependencia que existe entre el cliente y los subsistemas.

De esta manera también ayuda a eliminar dependencias de compilación, lo que provoca en sistemas grandes minimizar la recompilación de todo el sistema, teniendo que compilar solo las partes afectadas por cambios.

3.2.2.2.6. Flyweight[76]

Descripción

Muchas veces para realizar una representación correcta de un caso de la vida real se necesitarían demasiados objetos, lo que provocaría que el rendimiento de la aplicación, y la utilización de memoria sea demasiado alta.

Flyweight describe cómo se pueden compartir objetos para permitir el que sean utilizados para proveer un control con una granularidad muy alta, sin que el costo en memoria y rendimiento de esto sea demasiado alto.

Un objeto “flyweight” es un objeto compartido que puede ser utilizado en varios contextos simultáneamente, y actúa como un objeto independiente en cada uno de los contextos en los que es utilizado, y es indistinguible de una instancia del objeto que no es compartida. Un objeto compartido no puede asumir el contexto en el que se encuentra.

La base de esto es la distinción entre los estados del objeto, se tienen dos estados uno interno del objeto que es independiente del contexto en el que se encuentre, este estado es almacenado en el objeto, y el estado externo del objeto que es dependiente del contexto y varía con él. Este estado no es compartido entre los contextos desde los que el objeto es llamado, mientras el estado interno

[76] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

es compartido entre los contextos. Los objetos clientes son responsables de proveer al objeto “flyweight” datos sobre su estado externo cuando el objeto los necesita.

¿Qué tan efectiva es la utilización de este patrón de diseño? depende mucho de cómo es aplicado, según el libro de GoF[77], se debe aplicar este patrón de diseño cuando todas estas condiciones se cumplen.

- Una aplicación usa un gran número de objetos.
- Los costos de almacenamiento son altos debido al gran número de objetos.
- La mayor parte del estado del objeto puede ser externo.
- Muchos grupos de objetos pueden ser reemplazados por relativamente pocos una vez que el estado externo es removido.
- La aplicación no depende de la identidad de los objetos, ya que los objetos “flyweight” pueden ser compartidos, los tests de identidad pueden retornar un valor verdadero en objetos conceptualmente distintos.

Los componentes de este patrón de diseño son:

- Flyweight: Define una interface por medio de la que los objetos pueden recibir y actuar en un estado externo.

[77] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- ConcreteFlyweight: Implementa la interface de Flyweight y agrega la capacidad de almacenamiento para el estado interno del objeto, este objeto debe ser compartible, esto es que todo el estado que almacene debe ser interno al objeto.
- UnsharedConcreteFlyweight: No todas las subclases de un objeto flyweight deben ser compartidas.
- FlyweightFactory: Crea y maneja los objetos flyweight, se asegura que los objetos flyweight sean compartidos correctamente, cuando un cliente solicita un objeto flyweight, este objeto proporciona una instancia existente, o crea una nueva si es necesario.
- Client: Mantiene una referencia a los objetos flyweights, calcula o almacena el estado externo de los objetos.

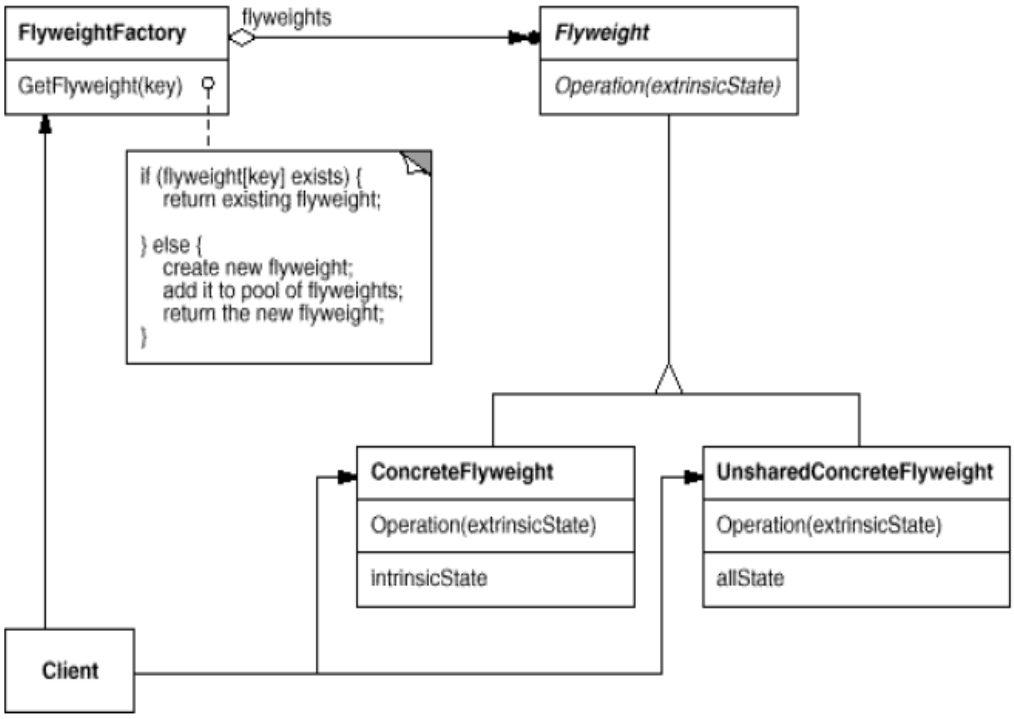


Figura 3-15: Flyweight[78]

[78] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

El siguiente diagrama muestra como los objetos Flyweight son compartidos.

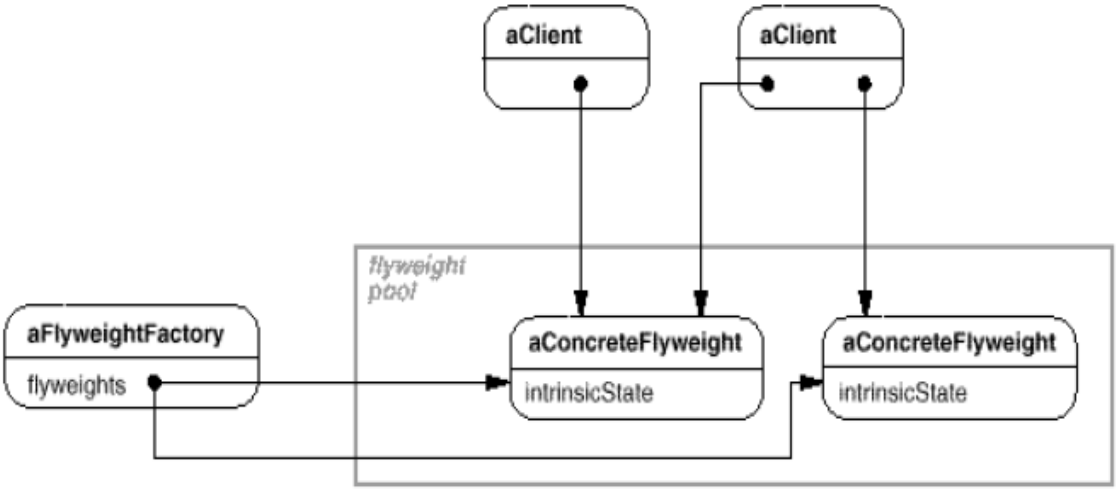


Figura 3-16: Flyweight compartidos[79]

Utilidad

Con la utilización de objetos flyweight se introducen nuevos costos asociados con el cálculo del estado externo de los objetos, así como la transferencia, localización de este estado para enviarlo al objeto que lo requiere. Sin embargo este costo es pequeño en relación al costo relacionado con el almacenamiento de objetos en memoria. Mientras más objetos son compartidos el ahorro en almacenamiento es cada vez mayor.

[79] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

3.2.2.2.7. Proxy[80]

Descripción.

La intención de este patrón de diseño es el tener un objeto que se encargue de controlar el acceso a otro objeto, de esta manera se puede controlar los procesos de creación y inicialización, hasta que en realidad el objeto sea necesario, pero esto debe ser hecho de una manera transparente de forma que no tenga impacto en el resto del sistema.

Un Proxy tiene varias formas de aplicación, algunos de los tipos de Proxy son:

- Proxy Remoto.- Provee una representación local de un objeto que se encuentra en un espacio o incluso aplicación distinta.

- Proxy Virtual.- Maneja la creación de objetos que son costosos de crear bajo demanda, es decir los crea cuando son necesarios en vez de crearlos cuando son declarados.

- Proxy de protección.- Controla el acceso al objeto original. Estos son utilizados cuando un objeto tiene diferentes privilegios de acceso que deben ser comprobados.

---

[80] Íd., Ibíd.

- Smart Reference.- Son utilizados como reemplazos para los punteros, estos objetos implementan funcionalidad adicional cuando se accede al objeto que representan, esta funcionalidad es usualmente:

    - Cuenta del número de referencias del objeto, de manera que pueda ser destruido automáticamente cuando no existen más referencias activas.
    - Cargar un objeto persistente a memoria cuando es llamado la primera vez, no cuando es declarado.
    - Verifica que el objeto real esté bloqueado antes de ser utilizado para evitar que otro objeto pueda realizar cambios mientras se encuentra en uso.

Los componentes participantes de este patrón de diseño se detallan a continuación.

- Proxy: Mantiene una referencia que permite al Proxy el acceso al objeto real, además de proveer una interface igual a la del objeto real, para que pueda ser reemplazado por un Proxy. Controla el acceso al objeto real de manera que es responsable por su creación y eliminación. Además de realizar todos los chequeos necesarios relacionados con un tipo específico de Proxy.
- Subject: Define una interface común para el Proxy y un objeto real, de manera que el Proxy pueda ser utilizado en cualquier punto en el que un objeto real sea esperado.

- RealSubject: Define el objeto que el Proxy representa.

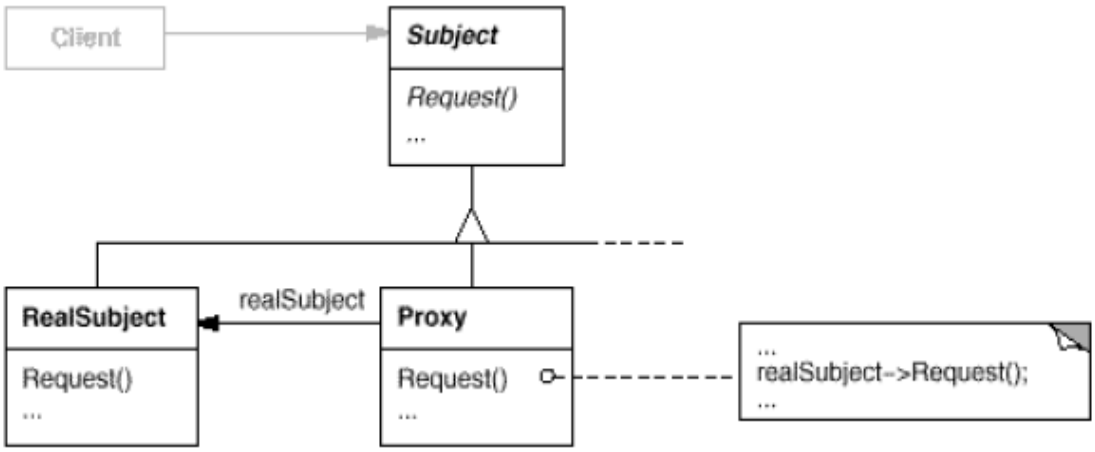


Figura 3-17: Proxy[81]

Utilidad

La utilidad más clara de este patrón de diseño es el hecho de que los objetos pueden realizar muchas tareas adicionales, como son verificar seguridad, creación bajo demanda, transparencia de localización del objeto creado, para simplificar el acceso, optimizar la utilización de recursos y cumplir con requerimientos de seguridad.

Ejemplo de Proxy

Un Proxy puede ser utilizado en una aplicación en la que sus componentes se conecten a una base de datos. Si se mantuviera las conexiones a la base de

[81] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

datos abiertas durante todo el tiempo o se mantuviera muchas conexiones abiertas a la base, además de influir en el rendimiento de la base de datos y la aplicación, posiblemente se mantengan conexiones abiertas que no se encuentren en uso.

En este caso un Proxy puede ser utilizado para centralizar la gestión de las conexiones a la base de datos, ésta tarea resulta siempre pesada para la aplicación y para la base. El Proxy puede encargarse de abrir la conexión solamente cuando es necesaria y destruirla automáticamente cuando ya no es requerida. Esta sería una de las posibles aplicaciones de un Proxy virtual. Puede ser utilizado también como un Proxy de Protección, agregando características como verificación de permisos de usuarios, de manera que usuarios no autorizados no tengan acceso a la base de datos, sin modificar políticas de seguridad directamente en la base de datos.

3.2.2.3. Patrones de Comportamiento

3.2.2.3.1. Interpreter[82]

Descripción

Si un grupo de problemas se repiten de manera muy común, posiblemente es posible definir un intérprete para que el problema pueda ser expresado en función de sentencias simples en un lenguaje más natural.

[82] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

De esta manera el intérprete se encargaría de definir la gramática para expresar el problema en función de sentencias más simples. Por ejemplo la forma de manejar las expresiones regulares en muchos lenguajes es similar, utilizando, expresiones que luego de ser interpretadas pueden ser utilizadas para determinar patrones de cadenas por ejemplo, y de esta manera no es necesario el escribir un algoritmo específico determinar uno por uno los tipos de cadenas que se desea manejar.

Este patrón de diseño define como se debe realizar esta interpretación de un parámetro específico, basándose en un modelo genérico. De esta manera para manejar o verificar si una cadena cumple con un patrón por ejemplo, no es necesario el reescribir el algoritmo de validación de la cadena, sino que simplemente se cambia este "modelo" de validación.

La estructura para este patrón de diseño se muestra en la siguiente figura.

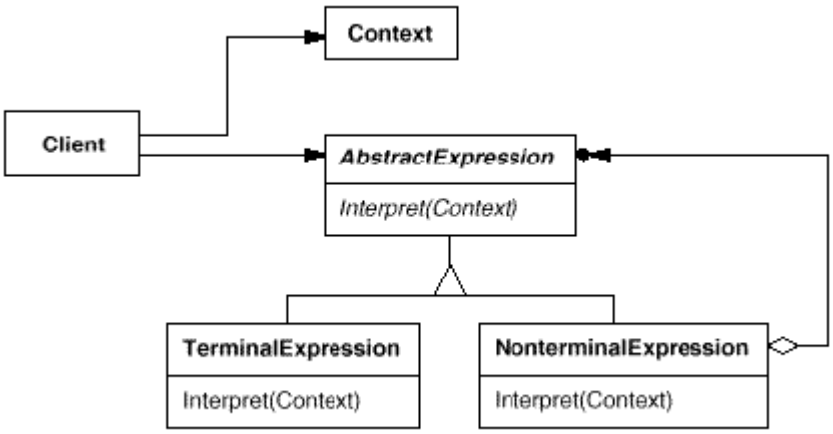


Figura 3-18: Interpreter[83]

[83] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- AbstractExpression: Define una operación interpret, que es común para todos los nodos en el árbol.
- TerminalExpression: Implementa una operación interpret asociada con símbolos terminales en la gramática, una instancia de esta clase es necesaria por cada uno de los símbolos terminales de la gramática.
- NonterminalExpression: una clase de este tipo es requerida para cada una de las reglas que están definidas dentro de la gramática, que pueden ser interpretadas como símbolos no terminales.
- Context: Contiene información que es global para el intérprete.
- Client: utiliza la operación interpret para generar un árbol que representa una frase en la gramática definida, este árbol está formado por instancias de TerminalExpression y NonterminalExpression.

Utilidad

Este patrón de diseño es útil cuando la gramática que se quiere implementar es simple, debido a que el árbol generado por las estructuras cuando una gramática es muy compleja y requiere de muchas reglas y símbolos se vuelve demasiado complicado para mantenerlo.

Por otro lado es fácil el cambiar la forma en la que una expresión es evaluada, agregar elementos de la gramática o cambiarlos utilizando herencia para extender las clases existentes, además el implementar una gramática es relativamente sencillo, ya que las clases son simples de implementar.

3.2.2.3.2. Template method[84]

Descripción

Template method define el esqueleto de un algoritmo en una operación definida, delegando algunos de los pasos del algoritmo a subclases. Además permite el redefinir ciertos pasos de un algoritmo sin cambia la estructura del algoritmo para la resolución de un cierto problema.

Un template method contiene un algoritmo definido en términos de operaciones abstractas que las subclases sobrecargan para obtener la funcionalidad concreta deseada. Definiendo los pasos a seguir para la realización de una operación, de esta manera se fija el orden que se debe seguir pero permite a las subclases de aplicación y documento el definir la forma de la implementación.

La estructura general de este patrón de diseño es la siguiente:

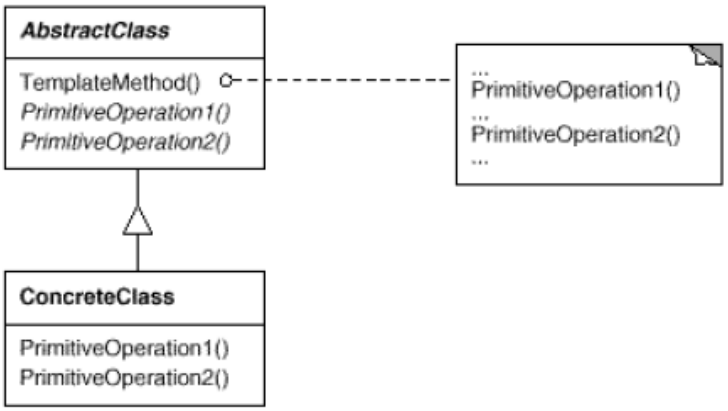


Figura 3-19: Template method[85]

[84] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Donde los componentes participantes son:

- AbstractClass: Define las operaciones primitivas abstractas que las subclases deben implementar para definir los pasos de un algoritmo. Además implementa un template method con el esqueleto del algoritmo, este template method realiza las llamadas a las operaciones primitivas.
- ConcreteClass: implementa las operaciones primitivas definidas en la clase abstracta, para definir la funcionalidad específica del algoritmo.

Utilidad

Este patrón es especialmente útil cuando se necesita tener un comportamiento común entre las subclases, y este comportamiento debe estar en un punto centralizado, para evitar duplicación de código. Además de esta forma se puede implementar las partes invariantes de un algoritmo y dejar a subclases el implementar partes específicas que pueden necesitar variación.

Otra forma de utilización de este patrón de diseño es para controlar los puntos en los que una subclase puede extender la funcionalidad de un algoritmo, se puede definir un template method que incluya llamadas a operaciones específicas en puntos específicos, permitiendo extensiones solamente en esos puntos.

---

[85] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

3.2.2.3.3. Chain of Responsability[86]

Descripción

La intención de este patrón de diseño es el evitar la unión entre un objeto que envía una petición y el objeto que la recibe, dando la posibilidad de manejar la petición a más de un objeto, encadenando los objetos receptores y pasando la petición por esta cadena hasta que un objeto la maneje.

El principio es que un objeto recibe la petición, y puede manejarla si es capaz de hacerlo o pasa esta petición hacia el siguiente objeto en la cadena, que hace lo mismo. De esta manera el objeto que envía la petición no tiene un conocimiento explícito de que objeto es el que maneja la petición. Es de esta manera que se puede generar jerarquías de objetos desde los más específicos hacia los más generales, así una petición puede viajar desde un objeto con propiedades específicas hacia un objeto más general hasta que alguno de los objetos en la cadena pueda manejar correctamente esta petición.

Para reenviar la petición por la cadena, cada objeto en la cadena debe compartir una interface común para manejar las peticiones y para reenviarla hacia el siguiente objeto en la cadena.

Los objetos y sus relaciones e este patrón de diseño se muestran en la siguiente figura.

---

[86] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

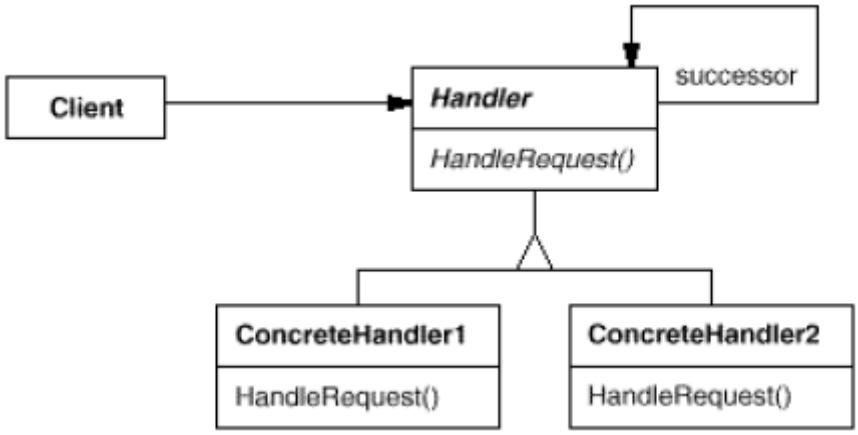


Figura 3-20: Chain of Responsability[87]

Donde:

- Handler: Define la interface para el manejo de las peticiones, además puede implementar un link hacia el sucesor en la cadena.
- ConcreteHandler: Maneja las peticiones que le corresponde, en caso de que sea necesario puede acceder a su sucesor en la cadena.
- Client: Envía la petición a un objeto ConcreteHandler de la cadena.

Utilidad

Se puede utilizar este patrón de diseño cuando más de un objeto puede manejar una petición y el objeto que va a manejar esta petición no es conocido, de esta manera se permite enviar la petición sin especificar un receptor definido. Además de esta manera el set de objetos que pueden manejar una petición

[87] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

puede ser definida dinámicamente brindando mucha flexibilidad en las operaciones.

De esta forma este patrón de diseño agrega una flexibilidad adicional al agregar, o cambiar responsabilidades de los objetos, cambiando la cadena en tiempo de ejecución. Uno de los problemas es que cuando una cadena no está bien configurada, es posible que una petición se mantenga sin manejarse, debido a que no existe un manejador explicito para una petición.

3.2.2.3.4. Command[88]

Descripción

La intención de este patrón de diseño es el encapsular una petición como un objeto de manera que sea posible el pasar esta petición, parametrizar clientes con diferentes peticiones, guardar un registro de las peticiones o ponerlas en cola, además de soportar operaciones que sean reversibles.

Muchas veces no es posible conocer qué operación va a ser realizada a partir de una petición, entonces si la petición es encapsulada como un objeto y luego pasada a un receptor cualquiera como un parámetro, el objeto que origina la petición puede no tener el conocimiento de la operación que se va a realizar y tampoco que objeto es el que va a realizarla.

[88] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Por ejemplo cuando se implementa un botón o menú, el botón debe responder a acciones, pero el programa que utiliza este botón es el responsable de realizar las operaciones por lo que el botón no puede tener en él una implementación de respuesta a una acción del usuario, de esta manera se puede utilizar el patrón Command para pasar esta petición encapsulada del botón hacia el objeto responsable de su manejo.

Los Comandos según el libro de GoF, es la forma de reemplazar las funciones llamadas Callback, en una forma orientada a objetos.

Debido a la encapsulación del comando como un objeto se puede guardar el estado para revertir los efectos de la operación, soportando de esta manera el deshacer acciones realizadas, la interface del comando tendrá entonces operaciones para ejecutar y una opción de deshacer también.

La estructura del patrón de diseño Command se muestra a continuación.

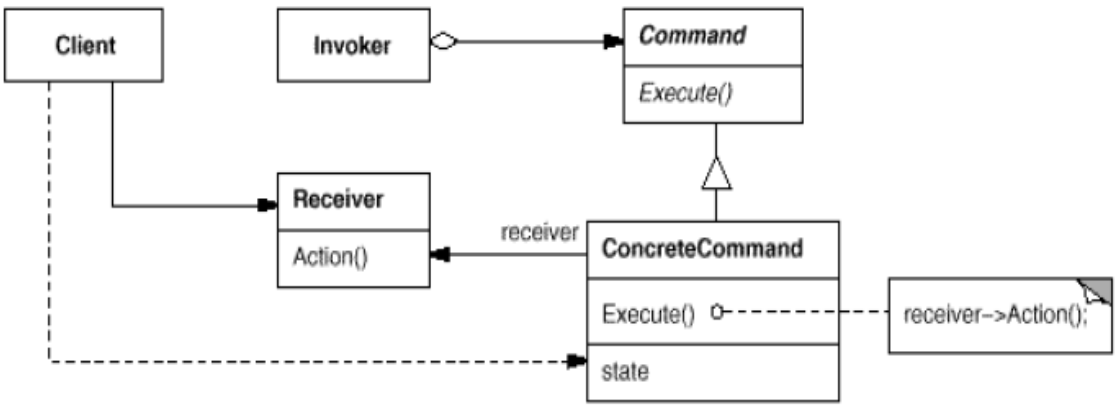


Figura 3-21: Command[89]

[89] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- Command: Declara una interface para ejecutar una operación
- ConcreteCommand: Define un enlace entre un objeto receptor y una acción, implementa la ejecución de una operación haciendo la llamada correspondiente en el receptor.
- Client: Crea un Comando concreto y define en el comando el objeto receptor.
- Invoker: Es el responsable de pedir al comando que lleve a cabo alguna petición específica.
- Receiver: Es quien implementa las operaciones asociadas a una petición específica.

Utilidad

El patrón de diseño Command logra que se desacoplen los objetos que realizan la petición de una operación de los objetos que mantienen la implementación de una operación dada. Incluso se pueden componer objetos command en un objeto compuesto de manera que se puedan ejecutar secuencias de comandos, incluso en receptores diferentes.

3.2.2.3.5. Iterator[90]

Descripción

Un iterador provee una forma para acceder a un objeto compuesto secuencialmente, sin exponer la implementación del objeto. La idea principal de este patrón de diseño es el presentar una forma de acceso a un objeto agregado, sin necesidad de agregar estas operaciones en el objeto en sí, un iterador mantiene el registro acerca del elemento actual, y es responsable de definir una interface por medio de la cual los clientes puedan recorrer un objeto.

Separar la forma de navegación por un objeto permite definir iteradores para varias políticas, sin agregar estos en la interface del objeto, por ejemplo se puede crear iteradores que contengan filtros de datos, de manera que un iterador solo mostraría los datos consistentes con un criterio definido.

El iterador y el objeto compuesto están unidos, de ahí que es necesario que el cliente conozca qué tipo de objeto es el que se está recorriendo con el iterador. Lo mejor en este punto sería el poder cambiar el objeto compuesto sin cambiar el código del cliente, esto se logra generalizando el concepto de iterador, para soportar algo llamado Iteración Polimórfica.

[90] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

De esta manera se pueden definir clases abstractas que definan una interface común para manipular una familia de objetos definida, de la misma manera se debe definir una clase abstracta de un iterador, que defina una interface común de iteración, a partir de estas clase abstractas se pueden derivar las clases concretas de implementación, de forma que los mecanismos de iteración se vuelven independientes de las clases concretas.

La estructura de los componentes de este patrón de diseño es la siguiente:

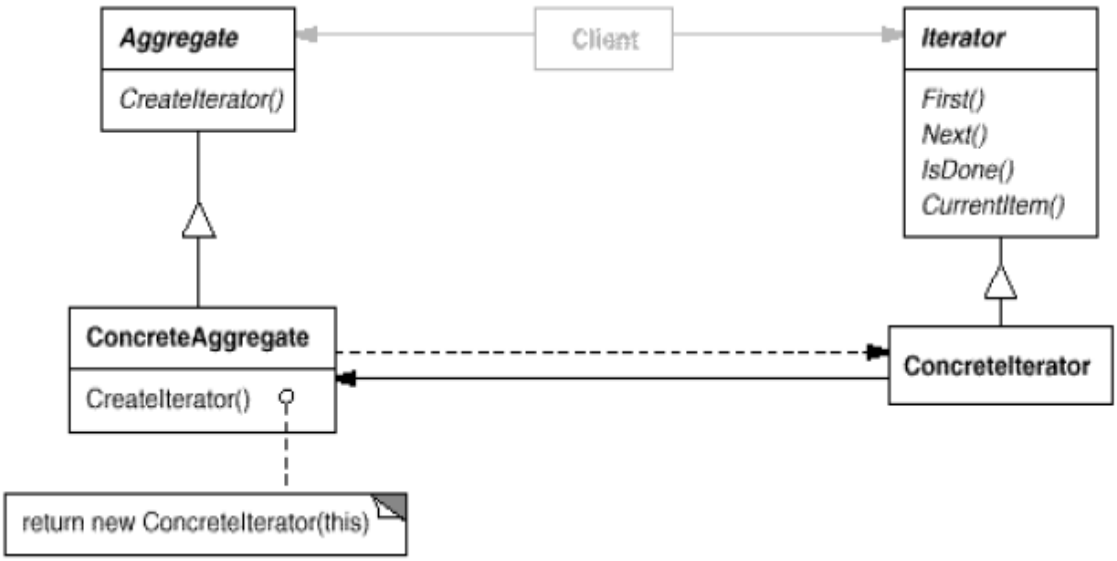


Figura 3-22: Iterator[91]

- Iterator: Define una interface para acceso y recorrido de los elementos.
- ConcreteIterator: Implementa la interfaz definida por Iterator, además mantiene un registro de la posición actual dentro del objeto compuesto.

[91] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- Aggregate: Define una interfaz para la creación de un objeto Iterator.
- ConcreteAgregate: Implementa la creación de el objeto iterador, para retornar una instancia del Iterador Concreto correspondiente.

Utilidad

Al utilizar iteradores, la forma de manejo y recorrido de objetos compuestos es mucho más flexible debido a que un iterador soporta variaciones en la forma de este recorrido, además se puede tener varios iteradores actuando sobre un objeto compuesto, debido a que el iterador guarda datos sobre el estado actual de su iteración, se puede tener operaciones pendientes de varios iteradores sobre el mismo objeto compuesto.

3.2.2.3.6. Mediator[92]

Descripción

En el desarrollo orientado a objetos, la encapsulación de los componentes nos permite mejorar las características de reusabilidad, mientras que el aumento de las interconexiones entre los objetos tiende a reducir esta reusabilidad. El mediator es un objeto que encapsula un grupo de objetos que interactúan entre ellos, de manera que los objetos no tienen que referirse a los demás de una manera explícita, y permite el variar la interacción independientemente.

[92] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Los componentes que participan en este patrón de diseño se detallan a continuación.

- Mediator: Define una interface para la comunicación de los objetos del grupo manejado.
- ConcreteMediator: Implementa la cooperación de los objetos coordinando los objetos del grupo manejado, además de guardar referencias hacia estos objetos.
- Colleage Classes: Son las clases manejadas por el mediator, cada una de estas clases tiene relacionado un objeto Mediator. Estas clases envían y reciben peticiones al objeto mediador, y se comunican con el en vez de comunicarse con los otros objetos.

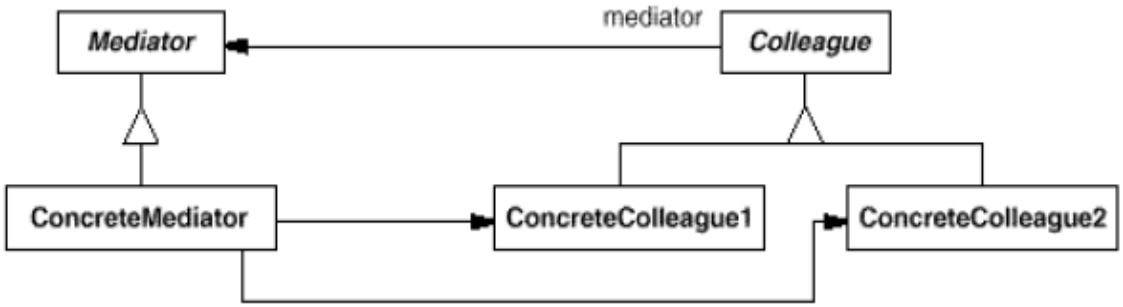


Figura 3-23: Mediator[93]

[93] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Utilidad

Un mediador debe ser utilizado cuando un grupo de objetos se comunican entre ellos de una forma bien definida, pero muy compleja con mucha dependencia entre los objetos participantes, además la reutilización de un objeto es difícil debido a que se comunica con muchos otros objetos.

También es aplicable cuando el comportamiento se encuentra distribuido entre varias clases y debe ser personalizable de una manera simple y sin muchas complicaciones.

Como consecuencia de la utilización de este patrón de diseño tenemos que: los protocolos de comunicación entre objetos se simplifican ya que en vez de tener interacciones de varios a varios, se reduce a interacciones de uno a varios donde el centro de las interacciones es el mediador.

Otra consecuencia es la centralización del control del comportamiento de los objetos, se cambia la complejidad entre la comunicación de los objetos por la complejidad en la implementación del mediador.

3.2.2.3.7. Memento[94]

Descripción

La intención de este patrón de diseño es el poder restaurar un objeto a un estado definido almacenado exteriormente, sin perjudicar la encapsulación de los objetos.

Muchas veces es necesario el registrar el estado interno de un objeto, por ejemplo cuando se necesita implementar puntos de control o acciones que puedan ser revertidas. De esta manera se debe guardar el estado interno de un objeto para que pueda ser restaurado posteriormente, pero normalmente un objeto encapsula su estado haciéndolo inaccesible para otros objetos, y por lo tanto imposible de registrar externamente.

Un memento es un objeto que almacena el estado de un objeto, llamado originador, el originador inicializa el memento con información que caracteriza el estado del objeto. Solamente el originador puede almacenar y recuperar datos almacenados en un memento.

Es muy importante que no se rompa la encapsulación del objeto utilizando interfaces directas para obtener el estado del mismo, ya que esto puede comprometer la implementación exponiéndola innecesariamente.

---

[94] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

La estructura de este patrón de diseño se muestra a continuación.

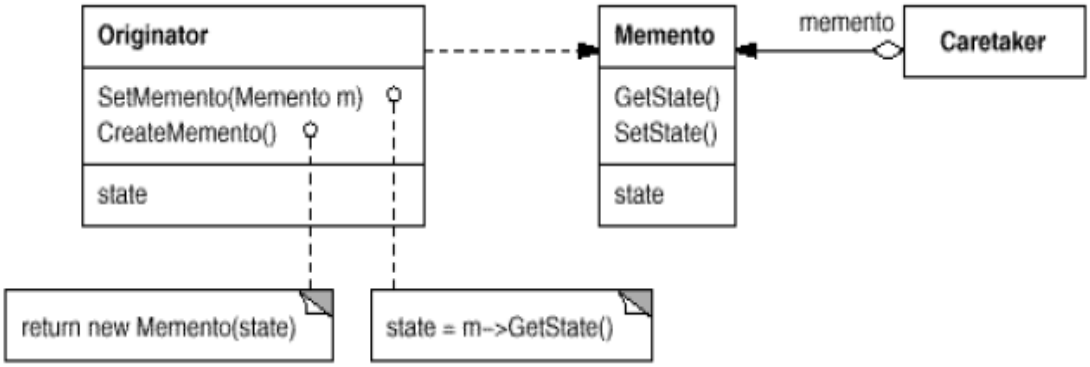


Figura 3-24: Memento[95]

- Memento: Almacena el estado interno de un originador, protege contra el acceso de otros que no sean el originador, un memento tiene dos interfaces, Caretaker que es una interface reducida que solamente puede pasar el memento a otros objetos. El originador al contrario tiene acceso a una interface extendida del memento que le permita recuperar y almacenar todos los datos necesarios.
- Originator: Crea el memento con la información del estado interno del objeto, y utiliza el memento para restaurar su estado interno.
- Caretaker: es responsable de la seguridad del acceso del memento, y nunca puede examinar o realizar operaciones sobre el contenido del memento.

[95] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Utilidad

Al utilizar mementos se preserva la encapsulación de los objetos, evitando que información que solamente debe ser accesible por un objeto sea accesible por otro, además simplifica el originador, en otras implementaciones el originador es el responsable de implementar toda la lógica para soportar el almacenamiento de los estados, haciéndolo muy complicado a veces, pero en cambio el costo del manejo, almacenamiento de los mementos puede ser alto.

3.2.2.3.8. Observer[96]

Descripción

El observar sirve para formar relaciones de uno a varios objetos, de manera que cuando un objeto cambia su estado, todos los objetos que dependen de este son notificados y se actualizan automáticamente.

Un efecto común en el particionamiento de un sistema en componentes y clases es la necesidad de mantener la consistencia entre varias clases, una forma de mantener esta consistencia es acoplar las clases pero de esta manera se hace más difícil la reutilización de estas clases, y ya no es posible reutilizar las clases independientemente.

[96] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

La implementación básica de este patrón de diseño consta de un sujeto y un observador, un sujeto puede tener varios observadores, de manera que el sujeto notifica cuando sucede un cambio a los observadores, y como respuesta cada uno de los observadores sincroniza sus datos con el sujeto.

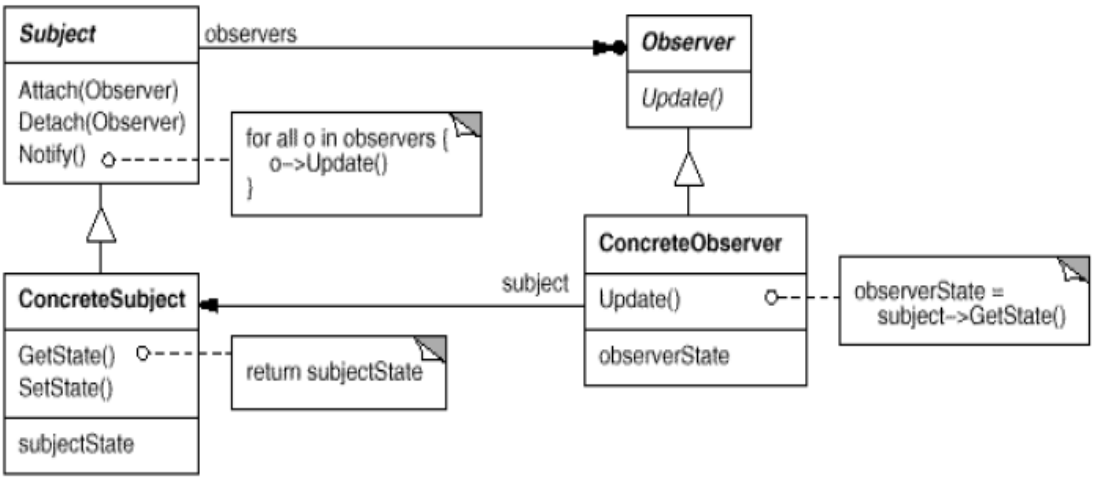


Figura 3-25: Observer[97]

En el diagrama se muestran los diferentes componentes de este patrón de diseño.

- Subject: Provee una interface para agregar y remover observadores, además guarda el registro de los observadores registrados.
- Observer: Define una interface para la actualización de los objetos que deben ser notificados.
- ConcreteSubject: Es la implementación del objeto que debe notificar a los observadores de sus cambios de estado.

[97] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

- ConcreteObserver: Mantiene una referencia a un objeto ConcreteSubject, además almacena estado que debe ser consistente con el estado del sujeto, implementa la interface de actualización del observer de manera que se pueda mantener su estado consistente.

De esta manera se puede separar la implementación de distintas formas de presentación por ejemplo para un mismo set de datos, ya que es posible presentar un set de datos de varias maneras, y no siempre se conoce cuantas formas de representación se va a tener, este patrón de diseño soluciona este problema.

Utilidad

Es recomendable utilizar este patrón de diseño cuando un objeto debe cambiar otros objetos, pero no se tiene un conocimiento explícito de cuantos son, además cuando un objeto debe ser capaz de notificar a otros de sus cambios sin tener una relación directa con los otros objetos o su implementación.

3.2.2.3.9. State[98]

Descripción

Este patrón de diseño es utilizado para permitir a un objeto cambiar su comportamiento cuando su estado interno cambia, de manera que aparentemente el objeto cambia su clase. De manera que se puede definir el comportamiento de un objeto dependiendo de su estado. Muchas veces los objetos para definir el comportamiento tienen largos condicionales, State pone cada uno de los condicionales referentes a un estado específico en una subclase, lo que permite que los estados sean tratados como objetos independientes y varíen libremente.

Las clases participantes en este patrón de diseño son las siguientes:

- Context: Define la interface de interés para los clientes, y mantiene una instancia de un ConcreteState que define el estado actual del objeto.
- State: Define una interface para encapsular el comportamiento asociado con un estado particular del contexto.
- ConcreteState: Cada una de las subclases implementa el comportamiento específico de un estado del contexto.

[98] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

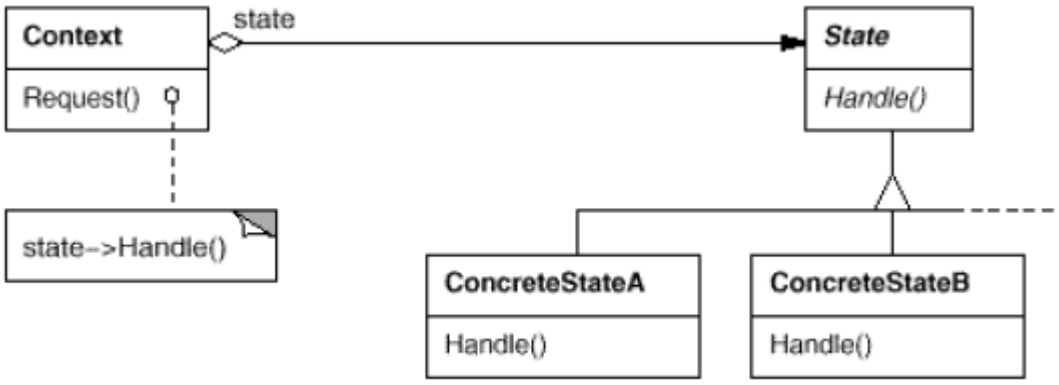


Figura 3-26: State[99]

Utilización

Debido a que este patrón de diseño localiza en subclases, se encapsula todo el comportamiento referente a un estado en una subclase específica, de esta manera es fácil agregar nuevos estados o transiciones definiendo subclases nuevas.

3.2.2.3.10. Strategy[100]

Descripción

El propósito de este patrón de diseño es el definir varios algoritmos para realizar tareas definidas, y encapsular cada uno de ellos en clases independientes. Al realizarlo de esta manera el algoritmo implementado puede variar independientemente de los clientes.

---

[99] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*
[100] Íd., Ibíd..

De esta manera se logra que el cliente pueda utilizar una implementación de un algoritmo sin atarse a él, y simplificando el código del cliente ya que todos los algoritmos variables son implementados externamente en otras clases.

Los componentes que intervienen en este patrón de diseño se detallan a continuación.

- Strategy: Declara una interface común para todos los algoritmos, el contexto utiliza esta interface para realizar la llamada correspondiente algoritmo definido por un objeto ConcreteStrategy.
- ConcreteStrategy: Implementa uno de los posibles algoritmos utilizando la interface definida por Strategy.
- Context: Es configurado con un objeto ConcreteStrategy, es el cliente que hace uso de los algoritmos definidos.

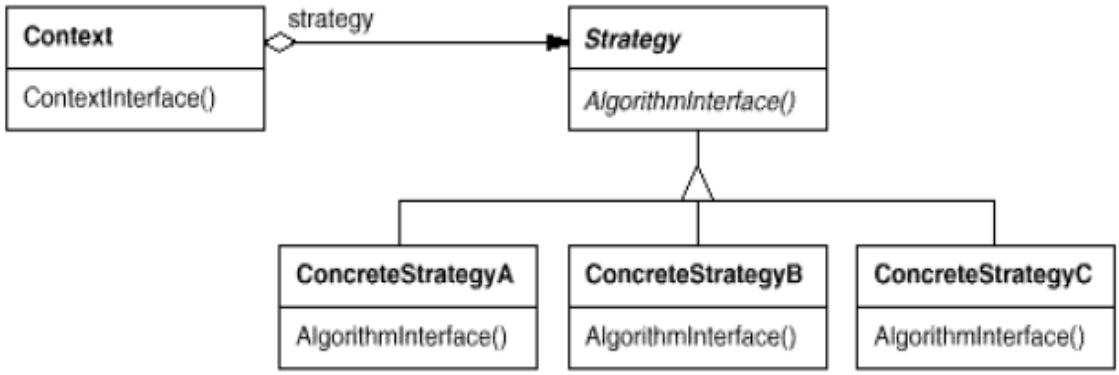


Figura 3-27: Strategy[101]

[101] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

Utilización

Este patrón de diseño puede ser utilizado cuando varias clases solamente se diferencian en su comportamiento, ya que con esto se provee un mecanismo para configurar el comportamiento de una clase. Definiendo muchas variantes diferentes de un algoritmo, o del comportamiento de un objeto. De esto se obtiene que el comportamiento de un objeto pueda ser cambiado dinámicamente si se lo enlaza con otro objeto de comportamiento, permitiendo una mayor flexibilidad a la herencia, donde se enlaza estáticamente un objeto con su comportamiento.

3.2.2.3.11. Visitor[102]

Descripción

Visitor permite que se representen operaciones sobre una estructura de objetos y estas operaciones sean cambiadas dinámicamente sin cambiar las clases de los elementos en los que se aplica la operación.

La idea principal de este patrón es que cuando se quiere aplicar operaciones entre dos objetos que representan una estructura de objetos, estas operaciones sean independientes de los objetos en las que se aplican para poder variar las operaciones libremente, sin tener la necesidad de recompilar al momento que es

[102] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

necesario cambiar la relación entre dos objetos, para esto un objeto que representa la operación que debe ser realizada es pasado entre los objetos.

La estructura de este patrón de diseño es la siguiente:

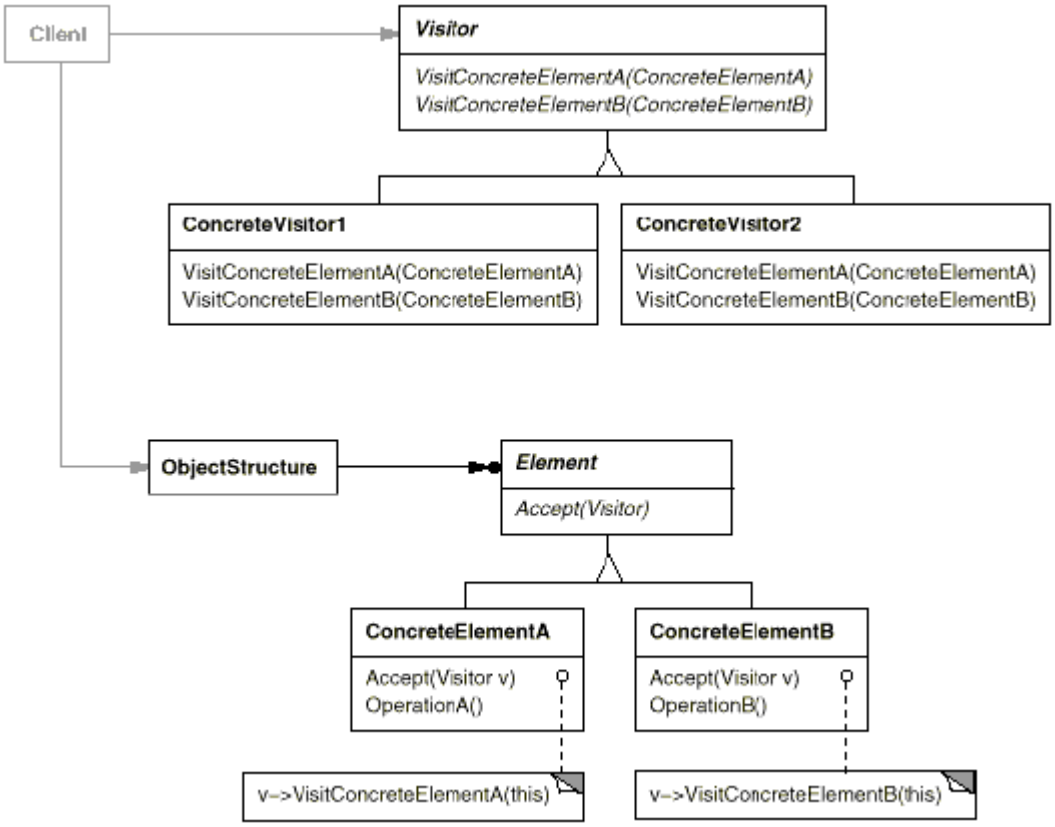


Figura 3-28: Visitor[103]

- Visitor: Define una operación para cada una de las clases de los Elementos Concretos, de manera que el nombre de la operación identifica la clase que envía la petición al visitante, esto permite al visitante determinar la clase concreta del elemento que está siendo

[103] E. Gamma, R. Helm, R. Johnson, J. Vlissides., *op. cit.*

visitado para acceder al elemento a través de su interface correspondiente.

- ConcreteVisitor: Implementa cada una de las operaciones definidas por Visitor, cada una de las operaciones implementa un fragmento del algoritmo definido para un objeto concreto, esta clase provee el contexto para que el algoritmo almacene el estado local, que va cambiando mientras el objeto es pasado a través de la estructura de objetos.
- Element: define una operación accept, que toma el visitor como un argumento.
- ConcreteElement: implementa la operación definida por Element.
- ObjectStructure: Provee la funcionalidad necesaria para que el visitante pueda “viajar” a través de la estructura de objetos.

Utilidad

Parte de las consecuencias de la utilización de este patrón de diseño es que el adicionar operaciones nuevas es relativamente sencillo, pero en cambio el adicionar nuevos elementos a la estructura de objetos se vuelve complicado debido a que se debe realizar la implementación correspondiente a ese nuevo elemento en cada uno de los ConcreteVisitor.

De aquí que este patrón debe ser aplicado cuando se piensa cambiar el algoritmo u operaciones que relacionan dos objetos de una jerarquía pero no se va a cambiar esta jerarquía de objetos muy constantemente.

3.2.3. Patrones de Programación

Son también llamados idiomas, un idioma es un patrón específico a un lenguaje de programación, los idiomas son mucho más específicos, ya que estos describen como se debe realizar la implementación de aspectos particulares de un componente, utilizando las características específicas de un lenguaje de programación dado, y de esta manera no son portables entre lenguajes.

Un idioma se refiere a conjuntos de instrucciones que repiten su organización para ofrecer una funcionalidad determinada, así de esta manera un patrón bastante simple es la forma de implementar bucles de operaciones matemáticas en C++, por ejemplo, el bucle y sus instrucciones siempre tienen la misma organización, y lo que varía es el operador que se utiliza para las operaciones, por ejemplo:

Tabla 3-1: Ejemplo de una operación - patrones de programación[104]

1+2+3+4+5+6+7+8+9+10	1*2*3*4*5*6*7*8*9*10

Int i=1;
Int ac=0;
While (i <= 10)
{
Ac=ac+i;
}
	Int i=1;
	Int ac=1;
	While(i<=10)
	{
	Ac=ac*i;
	}

[104] Elaboración Propia

Como se muestra en la anterior tabla, tanto la estructura de las sentencias, como la organización es la misma aunque la función específica es diferente, así de esta manera este se puede tomar como un patrón de programación de C, ya que su implementación es específica a un lenguaje de programación.

Este patrón que se puede llamar de Acumulación tiene una forma general que se puede expresar de la siguiente manera.

Código 3-1: Forma general patrón Acumulación[105]

```
``tipo'' ``variable'' = ``valor inicial'';
  ...
  while ( ... ) {
    ...
    ``variable'' = ``variable'' ``operador'' ``expresión'';
    ...
  }
```

Debido a que existen muchísimas formas de estos patrones, y son dependientes del lenguaje, no existe una clasificación definida para ellos, como en el caso de los patrones de diseño.

[105] Elaboración Propia

3.3. 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 de patrones, ayudan a especificar la estructura fundamental de una aplicación. Cada patrón de arquitectura ayuda a conseguir una propiedad específica en el sistema global; por ejemplo, la adaptabilidad de la interfaz de usuario. Los patrones que dan soporte a características similares se agrupan en una misma categoría.

Un patrón de arquitectura de software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución.

Los Patrones de arquitectura describen expresan una organización estructural para un sistema de software. Proveen un conjunto de subsistemas predefinidos e incluyen reglas y lineamientos para conectarlos.

3.3.1. Layers (Capas)[106]

Las capas son utilizadas para dividir en partes más pequeñas un software complejo. Una capa brinda servicios a otras capas; una capa superior generalmente utiliza los servicios de una capa inferior, sin embargo en la mayoría de los casos una capa inferior no tiene conocimiento de la existencia de una capa superior. Cada capa en encapsula su propia funcionalidad y cuándo un capa cambia las otras también lo harán en cascada.

Un sistema no debe ser dividido en más capas de las necesarias ya que esto puede incurrir en la disminución del rendimiento de un sistema debido al tiempo y los recursos que se necesitan para ir de una capa hacia otra.

Un layer puede ser una separación lógica o física; varios layers lógicos pueden estar en un mismo servidor físico, en una misma PC o servidor. Cuando se rompe esa barrera de denomina tier. Un tier puede ser el cliente y otro el servidor; pero en un servidor pueden estar varios layers.

[106] Martin Fowler, Enterprise Application Architecture Patterns

3.3.2. Patrones de lógica de dominio

3.3.2.1. Transaction script[107]

Intención

Organiza la lógica de dominio en donde cada procedimiento atiende una única petición de una capa de presentación o cliente.

Representación

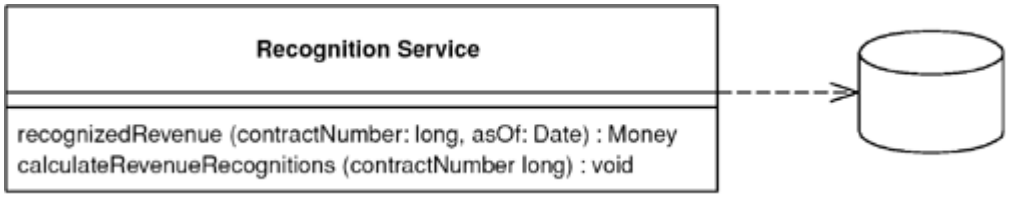


Figura 3-29: Transaction script[108]

Descripción y Funcionamiento

Es un modelo procedural que se refiere a la utilización de procesos que inician con una petición generalmente desde la capa de presentación o en su defecto desde otros procesos transaction script. La petición es recibida: se realizan validaciones, cálculos y otras operaciones como almacenar datos en

[107] Martin Fowler, *op. cit.*
[108] Íd., Ibíd.

bases de datos o se envía a otros transaction script, ya que no solo es un camino o proceso; cuando se termina de procesar se envía una respuesta que puede ser elaborada igualmente con cálculos y validaciones. Este proceso se realiza sin importar el estado o resultado de otras transacciones.

Existen 3 formas de organizar los transaction scripts, la primera forma es colocarlos dentro de una clase, la cual agrupe dependiendo de la funcionalidad que cumpla, la segunda forma es que cada transaction este contenida en su propia clase, usando un patrón de diseño conocido por GOF denominado Command, y la tercera forma es crear transaction script y exponerlos como clases globales dentro de un módulo o procedimiento.

Utilidad

Es un modelo simple de comprender e implementar, se sabe cuando inicia y cuando termina una transacción y se conoce sus barreras.

Es muy recomendable cuando la lógica de dominio no es tan compleja o extensa; cuando se trata de modelos más complejos es mejor optar por un Domain Model y evitando la duplicidad de código y transacciones.

3.3.2.2. Domain model[109]

Intención

Un modelo de objetos que implementa comportamiento y datos.

Representación

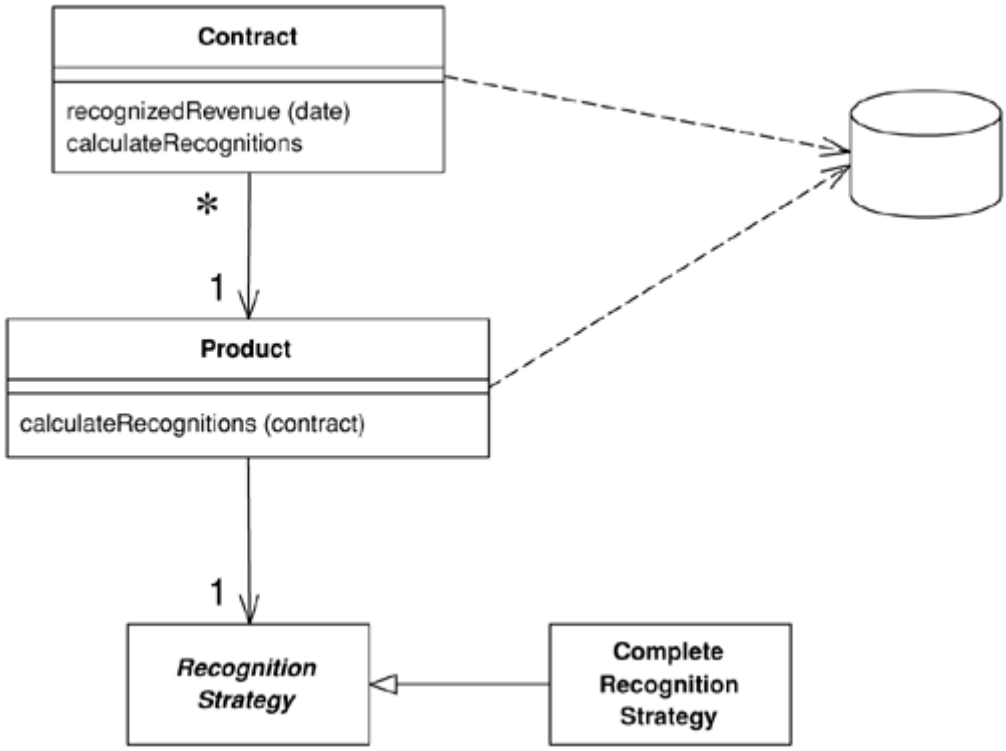


Figura 3-30: Domain model

Descripción y Funcionamiento

Con domain model se incorpora el concepto de orientación a objetos al modelo transaction script. Se pretende incorporar los objetos del mundo real de

[109] Martin Fowler, *op. cit.*

un sistema para representarlos en clases que puedan realizar los procesos de validación, cálculo, etc.

El uso de un domain model puede resultar complejo en un principio y algunos desarrolladores no se familiarizan enseguida pero una vez comprendido este modelo resulta mucho mejor que utilizar transaction script inclusive en proyectos pequeños.

En cuanto a éste patrón existen algunos detalles relacionados que no son fácilmente viables de una solución como es el problema del mapeo de un modelo de objetos a un modelo de tablas relacional y es por esto que al usar este patrón se debe además elaborar una estrategia de solución al problema y diseñar clases que realicen estas tarea como es un DataMapper.

Un domain model puede ser simple y verse como simplemente la representación de objetos de una base de datos o puede complicarse más cuando se incorpora herencia, estrategias u otros patrones de diseño de GOF.

3.3.2.3. Table Module[110]

Intención

Es una simple instancia que maneja la lógica de dominio de una tabla o vista de una base de datos.

[110] Martin Fowler, *op. cit.*

Representación

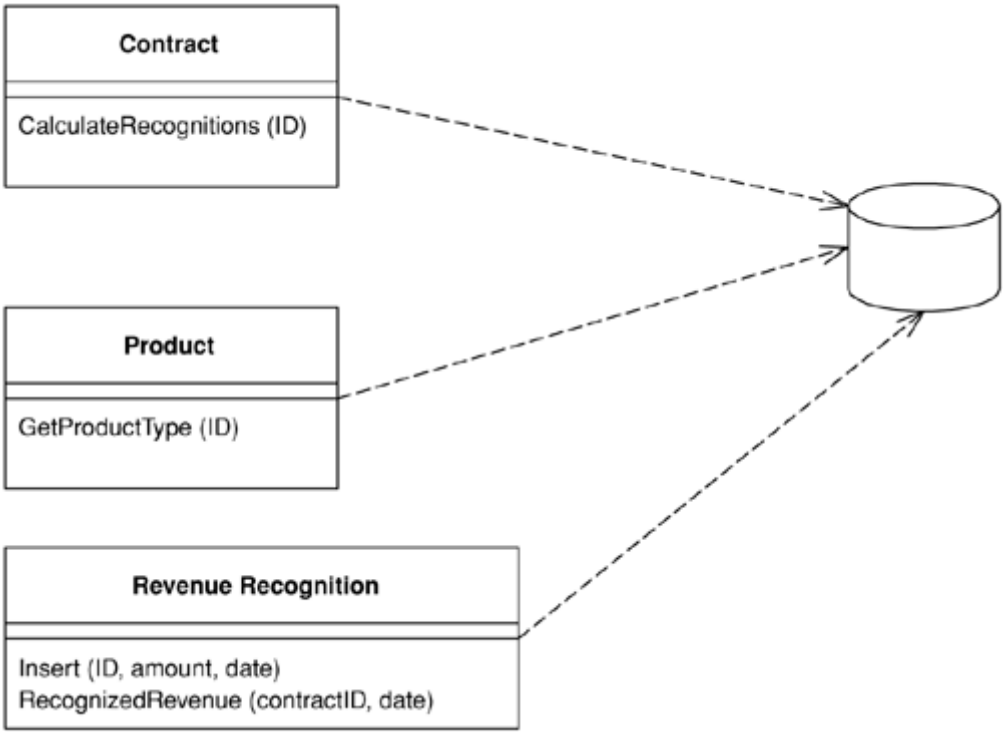


Figura 3-31: Table module[111]

Descripción y Funcionamiento

A diferencia de su similar Domain Model, el table module construye una sola instancia de un objeto por cada registro en la fuente de datos y su lógica de dominio asociada; es decir, si se habla de un objeto factura, domain model creará "n" objetos factura en memoria pero table module solo uno. Esto se logra ya que table module utiliza una estructura (ej. RecordSet) para almacenar a manera de tabla los registros de la base de datos. Una vez que un objeto table module es instanciado, éste consulta la fuente de datos y almacena a manera de tabla los registros en el RecordSet. De igual forma se puede realizar cualquier operación de cálculo pasando por ejemplo el ID del objeto Factura dentro del Table Module.

[111] Martin Fowler, *op. cit.*

Un table module puede ser la representación de una tabla de una base de datos pero esto no quiere decir que únicamente sea una tabla ya que se puede utilizar consultas o tablas virtuales que no necesariamente tengan la misma estructura de una tabla.

Utilidad

Una ventaja es poder utilizarlo en la capa de presentación ya que generalmente se tiene un buen soporte para los RecordSet. Tan solo se debe enviar una consulta, manipular los resultados en el Table Module y pasar estos datos manipulados a la interface gráfica.

Si bien un table module es una buena representación de una estructura de tabla, cuando se trata de objetos que sean más jerárquicos y utilicen herencia será siempre más conveniente utilizar un Domain model que brinde más flexibilidad y soporte a estos esquemas.

3.3.2.4. Service Layer[112]

Intención

Define el límite de una aplicación en el cual se provee una capa en donde se exponen operaciones disponibles y se coordina la repuesta de la aplicación en cada una de las operaciones solicitadas.

---

[112] Martin Fowler, *op. cit.*

Representación

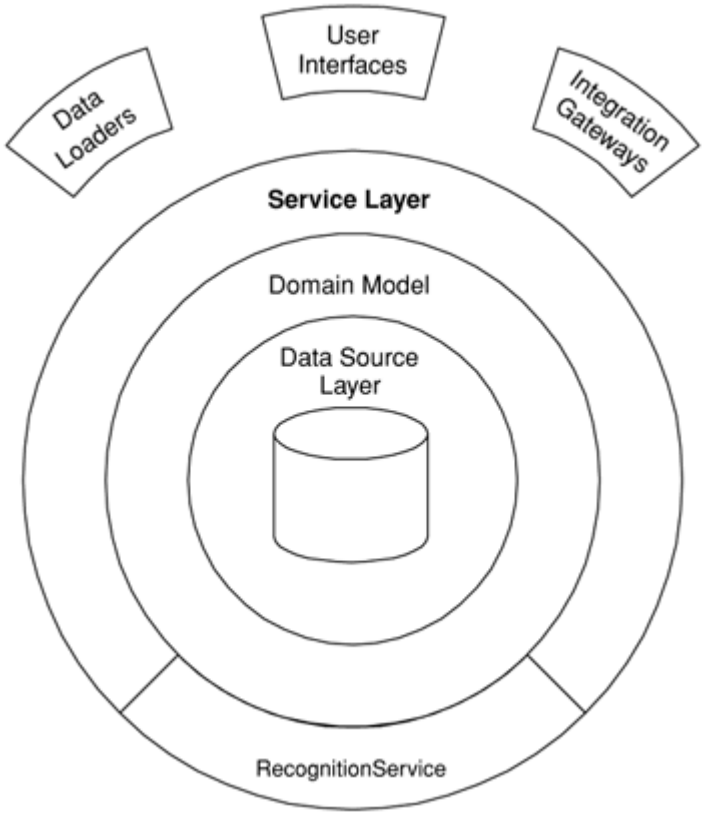


Figura 3-32: Service layer[113]

Descripción / Funcionamiento

Para un mejor manejo de la lógica de dominio de una aplicación se utiliza un service layer el cual divide la capa de lógica de dominio en dos partes. Una capa de service layer es colocada antes de un Table Module o de un domain model para servir de fachada entre una interface de presentación y la lógica como tal. Es muy útil colocar en esta capa servicios de transaccionalidad y seguridad para la aplicación. La separación de las la lógica de dominio en dos partes permite

[113] Martin Fowler, *op. cit.*

también controlar lo que se denomina la lógica de aplicación, la cual coordina el flujo de trabajo, notificación de eventos y servicios de asincronía.

La utilización de un service layer introduce también algunas características de reutilización de código mediante servicios, ya que al dividir la lógica de dominio la aplicación se vuelve más reutilizable entre varios ambientes; es decir si tenemos expuesta la lógica de dominio bajo una capa de servicio varias aplicaciones podrán beneficiarse de la lógica de dominio interna que está atrás del servicio.

Una capa de servicio debe cubrir todas las necesidades de la capa de servicio que viene desde los clientes o capa de interface de usuario. Generalmente esta capa contiene la representación de los casos de uso y el diseño de la aplicación.

Algunas de las operaciones de esta capa de servicio pueden ser simples como administrar una tabla para crear, insertar, actualizar y borrar datos, pero deben soportar transacciones atómicas y coordinación del flujo así como también servicios de asincronía.

Utilidad

El service layer no es tan recomendable usar en un Transaction script ya que éste no es tan complejo como para que represente colocar esta capa adicional.

Además cuando se trata de un solo tipo de cliente que se conecta a la capa de lógica de negocio no es muy aconsejable aumentar esta capa de servicio.

3.3.3. Mapeo a bases de datos relacionales

Desde que las bases de datos relacionales fueron creadas es común encontrar que la capa de datos se conecta a una base de datos relacional, aunque no dejan aún de existir los archivos VSAM e ISAM que utilizan los host y mainframes. El lenguaje de consulta que utilizan estas bases de datos es estándar y se denomina SQL.

Si bien la lógica de dominio tiene una fuerte comunicación con la base de datos, es muy recomendable separar completamente estas dos. Existen muchos desarrolladores que mezclan código con instrucciones SQL, pero sería mejor utilizar clases que utilicen algún mecanismo de conexión hacia la base de datos.

Bajo este esquema el resto de la aplicación no necesita conocer nada acerca de cómo se conecta a la base de datos ni qué está delante de ella, y toda la parte SQL de la aplicación es más organizada y fácil de encontrar. Esto trae beneficios tanto a desarrolladores de la parte de conexión a la base de datos de la aplicación como también a personas de IT que pueden sacar estadísticas, crear índices y lograr un mejor rendimiento de la base ya que toda la parte SQL reside en un mismo lugar.

Es así que nace el concepto de cómo representar los datos de una base de datos en estructuras o fragmentos de memoria, ya que en definitiva en una base

de datos se tienen tablas y registros pero no así en el código de la aplicación. Para esto existe un sinnúmero de posibilidades que van desde instanciar cada registro de la base de datos en una clase en memoria o por otro lado instanciar una sola clase por un conjunto de registros como es el caso de un Recordset e incluso otros patrones recomiendan que en estas clases se debe encapsular funcionalidad y operaciones sobre las filas (insertar, actualizar, borrar, buscar)

Dentro de las clases de acceso a datos se puede usar lenguaje SQL encapsulado en stored procedures o en sentencias parametrizadas. Ambos conducen al mismo resultado pero usar stored procedures puede brindar mejor rendimiento ya que en algunos motores de bases de datos como SQL Server al correr el stored procedure éste ya está pre-compilado, optimizado y con su plan de ejecución programado; pero en sentencias SQL parametrizadas cada vez que se ejecuta la consulta se debe calcular su plan de ejecución y hay cierta pérdida de rendimiento.

La tarea del mapeo del modelo entidad relación a modelo de objetos siempre resulta bastante compleja, existen patrones que detallan cómo debe estar estructurado y construido un mapeador de modelo de objetos a modelo relacional, sin embargo existe software de terceros que realiza esta operación y siempre resultará mucho más poderoso a diferencia de lo que se pueda hacer “a mano”. Una alternativa para evitar el mapeo Objeto / Relacional es utilizar una base de datos orientada a objetos, en la cual no hay que preocuparse por la tarea del mapeo ya que naturalmente los objetos se guardan en la base de datos con sus relaciones, herencia y dependencia con otros objetos. EL usar bases de datos

orientadas a objetos siempre será un riesgo y no se puede tener toda la confiabilidad y el respaldo que ofrecen los múltiples creadores de las bases de datos relacionales utilizados en la mayoría de los proyectos informáticos alrededor del mundo.

Más allá de la parte estructural del manejo de datos en las aplicaciones empresariales existe un problema adicional que es el manejo del estado de cada registro de la base de datos. Una vez leídos los registros de la base de datos se debe crear alguna estrategia para determinar cuándo éstos son modificados y poder plasmar los cambios directo hacia la base, ya sea que cada registro se actualice o exista algún otro componente que guarde su estado y pueda actualizarlos en batch.

Las bases de datos relacionales tienen relaciones entre las tablas y esto ocasiona un nuevo problema dentro del modelo de objetos ya que se debe determinar cómo representar éstas relaciones en memoria. Dentro de un maestro-detalle la tabla secundaria tiene un vínculo hacia la tabla primaria mediante la utilización de una clave foránea, mientras tanto en un modelo de objetos es completamente lo contrario ya que la clase que representa la tabla primaria es la que mantiene el vínculo con las clases dependientes relacionadas. Este vínculo múltiple es representado generalmente por colecciones o arreglos de objetos.

Adicionalmente de las relaciones entre tablas y objetos se debe tomar en cuenta la herencia de objetos. Las bases de datos relacionales no proveen de un mecanismo concreto de cómo plasmar la herencia de los objetos a tablas dentro

del modelo relacional, para esto también existen algunas técnicas dadas por patrones.

3.3.3.1. Table Data Gateway[114]

Intención

Un objeto que actúa como nexo hacia una tabla de una base de datos. Una instancia contiene todos los registros de la tabla.

Representación

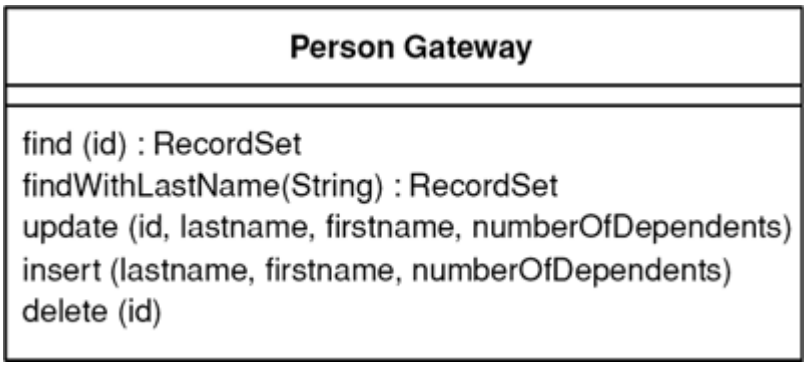


Figura 3-33: Table data gateway[115]

Descripción y Funcionamiento

Contiene una serie de métodos para manipular y obtener registros de una base de datos. Los datos son retornados y almacenados generalmente en una estructura tipo RecordSet.

---

[114] Martin Fowler, *op. cit.*
[115] Íd., Ibíd.

Es muy acoplable cuando se usan los patrones de lógica de dominio: TableModule y DomainModel. En ambos casos el Table Data Gateway provee de la funcionalidad para el envío y recepción de datos de forma tabular.

Se puede definir un Table Data Gateway por cada una de las tablas de una base de datos o se puede hacer una clase genérica en la cual por ejemplo se envía como parámetro la sentencia SQL y los parámetros requeridos.

Las operaciones más usuales que puede tener un Table Data Gateway pueden ser por ejemplo:

- BuscarPorCodigo(string)
- BuscarTodos()
- BuscarPorNombre(string)
- BuscarPor(condición)
- Insertar(param1, param2… param n)
- Actualizar(param1, param2… param n)
- Eliminar(int)

3.3.3.2. Row Data Gateway[116]

Intención

Un objeto que actúa como nexo hacia un solo registro en una fuente de datos. Se crea una instancia por cada registro.

[116] Martin Fowler, *op. cit.*

Descripción y Funcionamiento

Un Row Data Gateway representa un registro o una fila de una fuente de datos, en su estructura tiene los mismos campos de la tabla. Cada que un registro es instanciado en un Row Data Gateway los tipos de datos son convertidos y representados a tipos de datos en memoria; ejemplo un tipo de dato varchar(12) de la base de datos se mapean al tipo de dato string.

En las llamadas a un Row Data Gateway se puede acceder directamente a los datos de la fila, y esto se acopla bastante bien cuando se utiliza un Transaction Script.

Es prácticamente imprescindible crear dos clases para el momento de retornar los datos hacia un Row Data Gateway. Se crea una clase “buscador” o clase para retornar filas, la cual mediante parámetros pasados en el método realiza una consulta, recibe las filas en memoria y crea una clase Row Data Gateway por cada fila retornada en la consulta. Dentro de la clase “buscador” se puede definir un método por cada consulta o generar alguna clase genérica que encapsule un solo método para retornar los registros e instanciar las clases Row Data Gateway. En la siguiente figura se puede notar el funcionamiento de ambas clases.

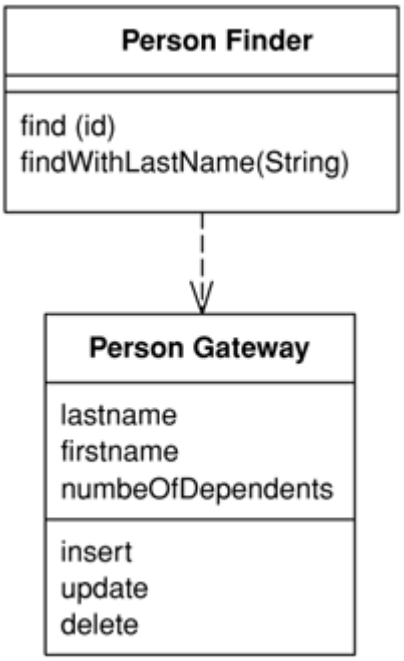


Figura 3-34: Row data gateway[117]

Se debe notar que para realizar las operaciones de inserción, eliminación y actualización se usa directamente el Row Data Gateway el cual contiene los métodos que encapsulan la funcionalidad para realizar dichas tareas. En estos métodos generalmente no se reciben parámetros y se usan directamente las propiedades almacenadas en cada una de las instancias de la clase, con estas variables se ejecutan los procedimientos o consultas que permiten realizar las operaciones en cada registro de la base de datos.

3.3.3.3. Active Record[118]

Intención

Un objeto que encapsula una fila de una fuente de datos, incluye las operaciones básicas de manejo de los registros y también código de lógica de negocio hacia los datos.

---

[117] Martin Fowler, *op. cit.*
[118] Íd., Ibíd.

Descripción y Funcionamiento

Active Record presenta una estructura muy similar a un Row Data Gateway, con las siguientes diferencias fundamentales:

- Se utiliza una sola clase tanto para los métodos de selección de registros o "buscadores" y para la implementación de la estructura de la tabla.
- Implementa lógica simple de negocio a más de incorporar los métodos para la manipulación de registros (Insertar, Actualizar, Borrar)

Estas características se pueden expresar en el siguiente gráfico:

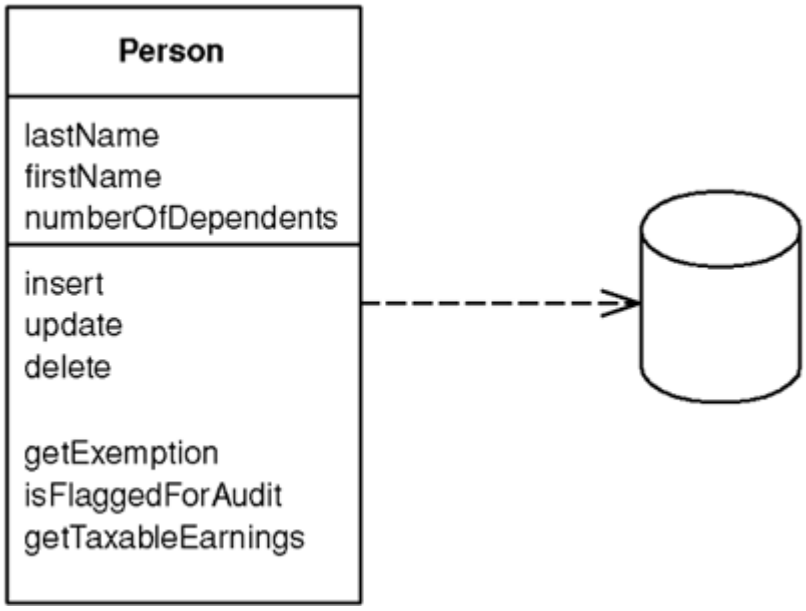


Figura 3-35: Active record[119]

[119] Martin Fowler, *op. cit.*

Entre otras características que se puede notar en un Active Record están:

- Crear instancias de objetos Active Record a partir de un Recordset.
- Los métodos “buscadores” son métodos estáticos que retornan instancias de objetos Active record que cumplen con la condición de la sentencia SQL.
- Los métodos accesores (“get/set”) hacia los campos de la clase, pueden tener lógica de negocio; un ejemplo puede ser que el registro de la base de datos tenga un campo Nombre y Apellido, pero al momento de mapearse al campo “NombreCompleto” de la clase se una Nombre + Apellido. Otro ejemplo es el mejor mapeo de tipo de datos, de tipos de datos de SQL a tipos de datos en memoria.

3.3.3.4. Data Mapper[120]

Intención

Representa una capa intermedia la cual mueve datos desde la base de datos hacia objetos en memoria pero manteniendo independencia entre uno y otro.

[120] Martin Fowler, *op. cit.*

Representación

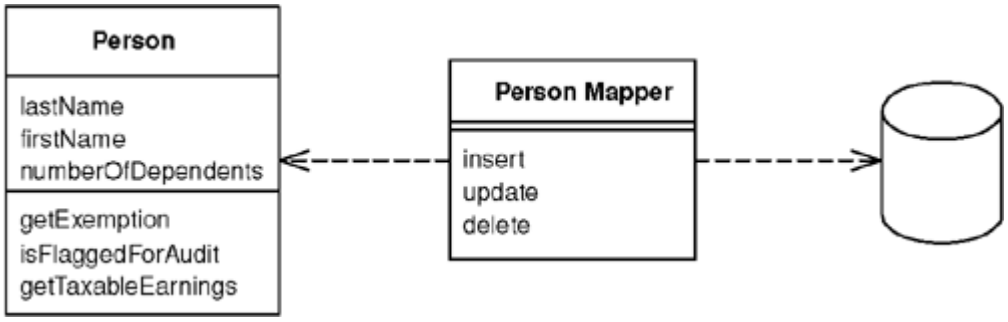


Figura 3-36: Data mapper[121]

Descripción y Funcionamiento

Un DataMapper realiza el mapeo de objetos a modelo relacional y viceversa, pero con la característica de mantener separado e independiente las clases que representan los datos de la base de datos y al mapeador como tal; es decir, el objeto en memoria no conoce realmente que existe una base de datos al otro extremo y su vez el mapeador tampoco necesita conocer qué tipo de objeto se encuentra al otro lado y solo tiene conocimiento de la base de datos.

Existen dos formas de realizar el proceso de mapeo. El primero es la implementación en código de la correspondencia de campos de la base de datos hacia campos de la lógica de dominio, en este caso mediante la utilización de métodos accesores en cada uno de los campos de la lógica de dominio. La segunda forma es utilizar el Mapeo de Metadatos, en el cual se utilizan archivos

[121] Martin Fowler, *op. cit.*

que guardan la información del mapeo como tal. En los archivos se guarda la correspondencia entre el campo de la base de datos y el campo de la lógica de dominio, junto con el mapeo de tipos de datos.

La metadata para el mapeo puede ser guardada utilizando XML, en donde por cada mapeador se define un archivo XML. No solo se puede guardar la información del mapeo en un archivo XML sino también en la propia base de datos; así si el modelo cambia también cambia el mapeo.

El mapeo de matadatos simplifica la tarea de mapeo significativamente, sin embargo se debe diseñar el motor o framework que realiza el parsing de los archivos de mapeo, gestiona la reflexión de objetos y otras tareas que un principio pueden parecer bastante complejas. Por esa razón si se quisiera implementar este tipo de mapeo es mejor adquirir una herramienta de terceros.

La gran ventaja del mapeo de metadatos es que cualquier cambio que se requiera realizar al modelo de base de datos no necesita de cambio en el código del mapeador, solo se modifica el archivo XML asociado para la tabla y el resto sigue funcionando sin problema.

En el siguiente ejemplo se puede ver la utilización de la técnica de Mapeo de Metadatos utilizando la arquitectura de Struts e Ibatis de java. Existen algunos componentes como Hibernate para java que realiza el mismo proceso.

Código 3-2: Mapeo de metadatos[122]

```
<?xml version="1.0" encoding="utf-8" ?>
<sqlMap namespace="Person"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SqlMap.xsd">
    <!-- XML "behind" document for the People service class. -->
    <alias>
        <typeAlias alias="Publishers"
type="IbatisBuraq.WebFace2.Publishers" />
    </alias>
    <resultMaps>
        <resultMap id="SelectAllResult" class="Publishers">
            <result property="Name" column="Name" />
            <result property="CompanyName" column="CompanyName" />
            <result property="Address" column="Address" />
            <result property="City" column="City" />
            <result property="State" column="State" />
        </resultMap>
    </resultMaps>
    <statements>
        <select id="Select" resultMap="SelectAllResult">
            select Name,[Company Name] as
CompanyName,Address,City, State
            from publishers
            Where State='NY'
        </select>
    </statements>
</sqlMap>
```

La gran ventaja de usar Mapeo de MetaDatos es que cualquier cambio en la base se refleja en los mapeadores y no se necesita realizar cambios en el código fuente del mapeador.

[122] http://www.learnasp.com/freebook/learn/ibatis1.aspx

3.3.3.5. Unit of Work[123]

Intención

Mantiene una lista de objetos afectada por una transacción de negocio y coordina que los elementos afectados que hayan sido afectados durante la operación se actualicen en la fuente de datos.

Descripción y Funcionamiento

Cuando se trae datos desde una fuente de datos hacia memoria, ciertos registros de éste conjunto de datos son manipulados e inclusive creados nuevos, y es precisamente el Unit of Work el encargado de guardar una lista o registro de todo lo que ocurrió durante la manipulación de datos en memoria y después actualizar los cambios en un ambiente transaccional y con criterios de concurrencia de datos en el caso de que un mismo registro hubiera sido modificado al mismo tiempo por dos o más clientes.

Dentro del comportamiento y la forma de invocación hacia un objeto Unit of Work están:

- Llamada por parte del cliente, en éste caso el objeto que inicia la consulta también registra al objeto Unit of Work mientras realiza la manipulación de los registros.

[123] Martin Fowler, *op. cit.*

- Llamada por parte del objeto de negocio, en éste caso el objeto de negocio (Ejemplo “Factura”) es el que registra todas las operaciones de manipulación de los registros en el objeto Unit of Work.

- Otra posibilidad es usar un Controlador Unit of Work, en el cual cualquier consulta hacia la fuente de datos es primero procesada por éste controlador. El controlador crea una copia en memoria de los registros almacenados y envía los datos solicitados hacia el objeto que lo invocó. Después el objeto que realiza la manipulación de los registros envía los datos hacia el controlador y éste compara la copia inicial con la copia almacenada y así determina que operaciones se deben realizar para actualizar los registros con modificaciones.

Este ambiente es muy usado en la plataforma .net con el uso de DataSets desconectados. Cada fila de un DataSet tiene un estado (“sin cambio”, “actualizado”, “eliminado” y “nuevo”), luego se realizan los cambios en las filas, y al momento de regresar dichos cambios a la fuente de datos se ejecuta un método como DataSet.GetChanges(), éste devuelve otro DataSet con las filas alteradas y mediante un DataAdapter se actualizan los cambios de vuelta en la fuente de datos.

3.3.3.6. Identity Map[124]

Intención

Asegura que un objeto de negocio se cargado una única vez en memoria guardando en un mapa cada objeto que es cargado durante una consulta.

Descripción y Funcionamiento

Existen varios casos y usos en los que un Identity Map es conveniente, el primero y más obvio es lo relacionado con el rendimiento ya que al asegurar que un objeto de negocio solo se cargue una vez evita que se consulte más de una vez a la fuente de datos; el otro uso importante viene relacionado con la concurrencia, si se cargaría más de una vez un objeto “factura” no se sabría con cual se está trabajando y vendrían problema de inconsistencia de datos.

El funcionamiento básico es que antes de recuperar un registro de una fuente de datos se consulta primero al identity map, si el registro está cargado se devuelve el objeto en memoria caso contrario se lo recupera de la fuente de datos. Puede existir un Identity Map por cada objeto de negocio o se podría implementar uno genérico para todos, el ejemplo de una llamada para cada caso seria:

- Individual: *SeleccionarFactura(int CodFactura)*

[124] Martin Fowler, *op. cit.*

- Genérico: *Seleccionar(string Tabla, int Codigo)*

Es importante determinar correctamente una clave primaria que represente el registro de forma única tanto en memoria como en la base de datos, y para esto se tienen siempre varias opciones, pero si se quisiera implementar un Identity Map genérico es más fácil si todas los objetos de negocio tienen el mismo tipo de clave o identificador.

3.3.3.7. Lazy Load[125]

Intención

Es un objeto de negocio que tiene la estructura de un objeto tradicional pero que solo carga los datos internos cuando se los necesita.

Descripción y Funcionamiento

Cuando se trabaja con objetos de negocio que internamente tienen otros objetos de negocio que dependen del primero, es conveniente no cargarlos por completo para evitar consumir recursos innecesarios. Un ejemplo de esto puede ser un objeto Cliente que internamente tenga un campo que represente una colección de facturas asociadas con el cliente; no sería necesario, en un principio, cargar todos los datos de dichas facturas si solo se necesita saber el Nombre del cliente, pero al momento de requerir las facturas asociadas se accede a la

[125] Martin Fowler, *op. cit.*

propiedad facturas del objeto Cliente y éste determina si debe cargar en memoria los objetos Factura.

Hay algunas posibilidades de implementar ésta funcionalidad:

- Inicialización Lazy: En este caso al instanciarse el objeto de negocio se ponen a todos los campos en Null y con esto se logra determinar cuales no han sido cargados. Esto funciona bien en primer momento pero se tiene problema cuando un campo puede ser nulo, en cuyo caso no se podría determinar si verdaderamente el valor es nulo y si éste no se ha cargado.
- Proxy Virtual: representa al objeto con su estructura tal como se vería realmente pero realmente sus campos internos no contienen información, y ésta se carga realmente cuando se acceden a los métodos accesotes y en ése momento se carga su contenido por primera vez.
- Contenedor de Valor: es un objeto que envuelve o encapsula al verdadero objeto y lo carga al ser llamado por primera vez.
- Ghost: Representa al objeto en un su estado parcial en el cual cada campo de la clase contiene inicialmente solo el identificador único del objeto que aún no se encuentra cargado, una vez que se lo llama por primera vez se carga el resto del objeto. Se puede alterar la inicialización del objeto para que no solo se cargue el identificador de los objetos internos sino también más información que se sabe que se utilizará en un principio o con mayor frecuencia.

3.3.3.8. Identity Field[126]

Intención

Guarda un identificador único para mantener la identidad entre un objeto en memoria y un registro de la fuente de datos.

Descripción y Funcionamiento

Un objeto de negocio debe saber diferenciarse únicamente tanto en memoria como en la fuente de datos. En el segundo caso se lo realiza con la utilización de claves primarias y campos de tipo único, pero en el segundo caso éstos ya están diferenciados simplemente por estar en registros diferentes de memoria; es por esto que se necesita una correspondencia entre ambos tipos de objetos.

El caso más simple es que el objeto en memoria tenga la misma estructura de clave primaria que el objeto en la fuente de datos, y se puede mapear directamente. En otros casos se puede implementar una clase tipo “key” que tenga la correspondencia entre la clave primaria del objeto en memoria y el de la fuente de datos.

3.3.3.9. Foreign Key Mapping[127]

Intención

[126] Martin Fowler, *op. cit.*
[127] Martin Fowler, *op. cit.*

Mapea las relaciones de claves foráneas entre objetos dependientes hacia otros objetos.

Representación

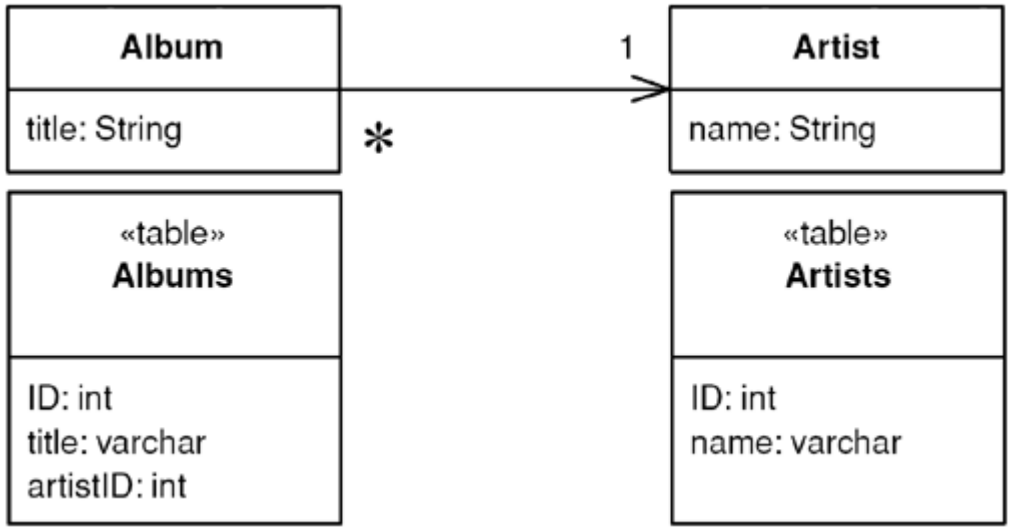


Figura 3-37: Foreign key mapping[128]

Descripción y Funcionamiento

Las bases relacionales se basan estrechamente en la relación entre las tablas y se utilizan técnicas como las claves foráneas para poder determinar qué tabla es dependiente de otra. También se utilizan tablas auxiliares para poder guardar las relaciones de varios a varios entre las tablas participantes.

Una relación en una tabla puede ser representada simplemente colocando el campo primario de la tabla principal en un campo de la tabla secundaria, pero si

[128] Íd., Ibíd.

esta relación la queremos representar en memoria basta con relacionar el objeto con su dependiente directamente; es por esto que el mapeo de relaciones es muy diferente en el modelo de objetos.

En un modelo de objetos el objeto que es dependiente no se mapea únicamente como un identificador de clave foránea sino mas bien se copia al objeto entero dentro del objeto que lo contiene. Esto se puede explicar de mejor con el siguiente ejemplo:

Tabla 3-2: Mapeo objeto – relacional[129]

Mapeo relacional en una base de datos, Tabla Factura	**Modelo de objetos en memoria**
IdFactura - int **IdCliente - int** Monto - decimal Iva - decimal	IdFactura – integer **Cliente (Instancia del Objeto Cliente)** Monto - decimal Iva – decimal

En el ejemplo podemos notar que en el modelo de objetos la relación se puede representar colocando directamente el objeto cliente dentro del objeto factura. En éste caso representamos que una factura pertenece a un cliente. Sin embargo si tomamos el ejemplo anterior pero de forma inversa, es decir que un cliente tiene facturas, tendríamos que un cliente, como parte de sus campos tiene un arreglo o colección de objetos facturas.

[129] Elaboración Propia

Tabla 3-3: Mapeo objeto relacional (2) [130]

Mapeo relacional en una base de datos, Tabla Cliente	**Modelo de objetos en memoria**
IdCliente - int Nombre - varchar Cedula - varchar	IdCliente – integer Nombre - string Cedula - string **Facturas (Colección objetos Factura)**

Igual que en el ejemplo anterior ocurre cuando se trata de relaciones de varios a varios, la representación de la relación es directa: los objetos primarios contienen a los objetos dependientes mediante colecciones. Esto no ocurre en la base de datos pues se requiere de una tabla intermedia ya que en un solo campo no se puede guardar múltiples valores relacionados.

[130] Elaboración Propia

3.3.3.10. Dependent Mapping[131]

Intención

Representa un objeto que realiza el mapeo objeto-relacional de sí mismo y de sus dependientes.

Representación

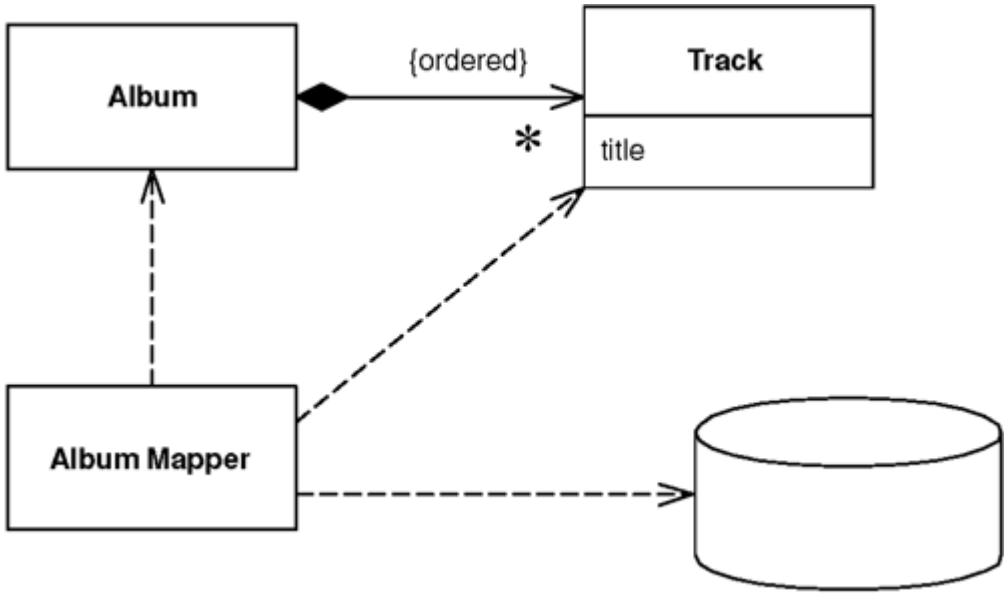


Figura 3-38: Dependent mapping[132]

[131] Martin Fowler, *op. cit.*
[132] Íd., Ibíd.

Descripción y Funcionamiento

Existen objetos que dependen íntimamente de otros objetos, por ejemplo cuando se trata de un maestro/detalle. Ambos objetos generalmente se cargan juntos desde la base e inclusive generalmente no se tiene un Método para cargar el detalle por sí solo por ejemplo: *CargarDetallePorId(int IdDetalle)* sino mas bien *CargarDetallePorPadre(int IDPadre)*. En éste caso se tienen dos objetos el primero es el Maestro y el segundo es el detalle, y justamente la intención de éste patrón es que todas las operaciones de mapeo y de persistencia hacia la base de datos sean manejadas y procesadas por el objeto maestro.

Se necesitan algunas condiciones esenciales para que éste modelo funcione correctamente:

- Se debe notificar al objeto maestro cuando el objeto dependiente cambie, caso contrario no se podría realizar la actualización en la fuente de datos.
- El objeto dependiente debe tener un solo padre o maestro que lo contenga.
- Si se quiere acceder al objeto dependiente se debe pasar primero por el objeto padre para poder tener la referencia al objeto solicitado.
- Pueden existir objetos dependientes que a su vez contengan otros objetos dependientes, se debe guardar la jerarquía y cada maestro debe gestionar las operaciones de sus dependientes.

3.3.3.11. Embedded Value[133]

Intención

Mapea los campos de un objeto o tabla en campos de otro objeto.

Representación

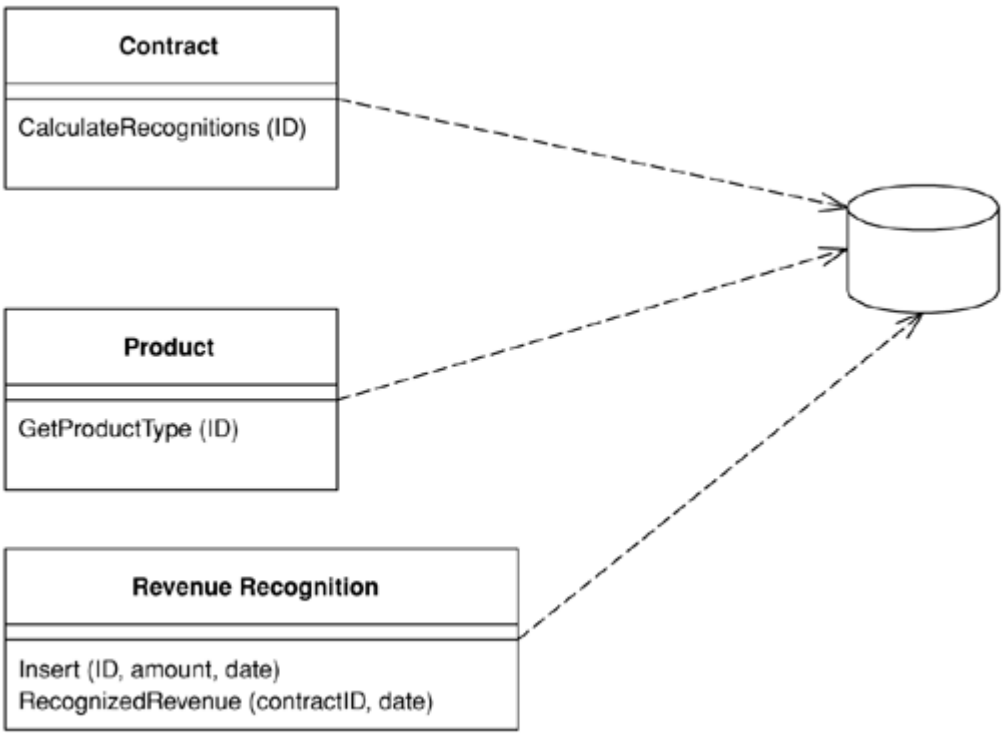


Figura 3-39: Embedded value[134]

Descripción y Funcionamiento

Por lo general cuando se mapea las tablas hacia objetos de memoria se tiende a representar la tabla tal y como es en el modelo de objetos. Esto no

[133] Martin Fowler, *op. cit.*
[134] Íd., Ibíd.

siempre puede ser útil ya que se pueden resumir los campos de las tablas en campos más simples hacia los objetos, por ejemplo uniendo o varios campos en uno solo.

Tabla 3-4: Ejemplo mapeo de campos[135]

Tabla	Objeto
Nombre: varchar Apellido: varchar Titulo: varchar	NombreCompleto: string
totalCompra: decimal moneda: char	Total: money

3.3.3.12. Serialized Lob[136]

Intención

Guarda un objeto de memoria en un campo de una base de datos mediante serialización.

Descripción y Funcionamiento

Cuando un objeto no puede ser representado directamente con tipos de datos o estructuras de una base de datos se necesita algún mecanismo para

[135] Elaboración Propia
[136] Martin Fowler, *op. cit.*

guardar éstos objetos. Para que esto ocurra se debe emplear la serialización de objetos. Existen dos formas de representar los objetos serializados de dos formas:

Serialización Binaria (BLOB)

Es realizado con funciones de la herramienta de programación, se necesita también que la base de datos soporte el tipo de datos de tipo binario o también llamado image. Cuando está ya almacenado no puede ser interpretado ni leído como cualquier otro tipo de dato, y debe convertirse de nuevo al objeto que representa para ser usado. Es la forma más compacta y rápida de serialización tanto para escritura como para lectura.

Serialización de Texto (CLOB)

Es representado por una cadena de caracteres que tienen un cierto formato o reglas. El caso más preciso para representar un objeto serializado es un objeto XML. Para generar la cadena de caracteres XML se puede escribir "a mano" o utilizar un parsers de XML que es bastante común. El XML por ser un lenguaje declaratorio termina ocupando bastante espacio y por ende éste tipo de serialización si bien es más comprensible y explicita que la serialización binaria termina ocupando bastante espacio. Sin embargo trae ventajas importantes ya que se puede editar directamente el contenido almacenado en la base y se puede interpretar son simple lectura.

3.3.3.13. Single Table Inheritance[137]

Intención

Representa una jerarquía de herencia de clases dentro de una tabla de una base de datos la cual contiene todos los campos que existen en las clases de la herencia.

Representación

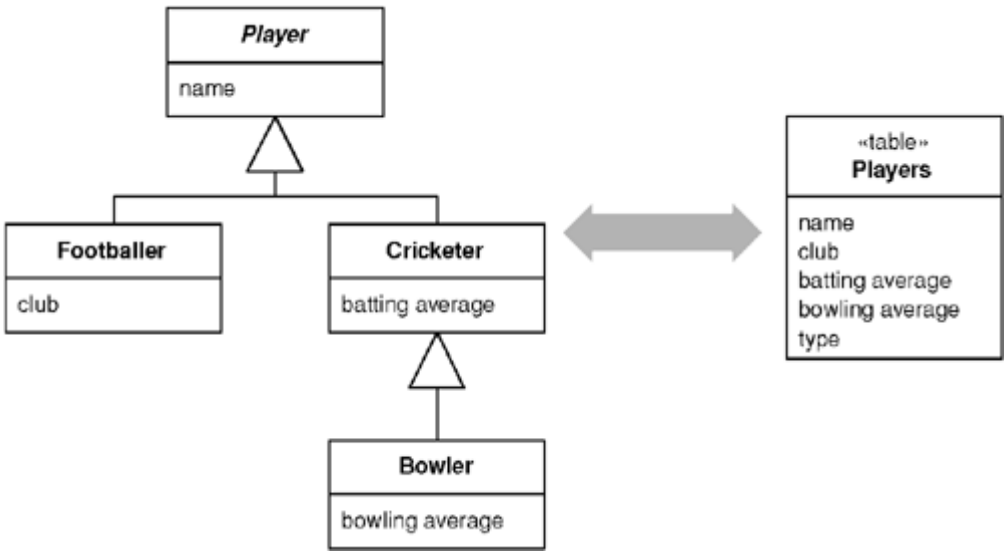


Figura 3-40: Single table inheritance[138]

Descripción y Funcionamiento

Cuando se utiliza bases de datos relacionales se necesita mapear la herencia de objetos en tablas que no tienen ningún soporte para herencia. El

[137] Martin Fowler, *op. cit.*
[138] Íd., Ibíd..

primer caso es la herencia simple de tablas, donde se tiene una sola tabla para poder representar todos los objetos que participan en la herencia.

El funcionamiento básico en éste caso es que cada campo de cada clase en la jerarquía de la herencia se copia a la tabla, y para representar los datos de cada clase se llenan los campos de la base de datos que son utilizados por dicha clase y los demás se dejan vacíos. Adicionalmente se tiene un campo en la tabla en donde se guarda el tipo de objeto que se guarda en la tabla. Se puede guardar el nombre del tipo de objeto o un código identificador.

Ventajas

- Hay que preocuparse de solo una tabla en la base de datos, ya que en ésta se almacena toda la jerarquía de herencia y múltiples tipos de objetos.
- No se necesita relaciones ya que es una sola tabla.

Desventajas

- Se desperdicia espacio al no utilizar todos los campos de la tabla.
- No se puede utilizar el mismo nombre de campo si una o varias tablas se refieren a un campo con el mismo nombre.
- La tabla puede llegar a ser demasiado grande y se necesitaría poner índices para agilitar las consultas.

3.3.3.14. Class Table Inheritance[139]

Intención

Representa una jerarquía de herencia de clases en donde cada clase se representa en una tabla individual.

Representación

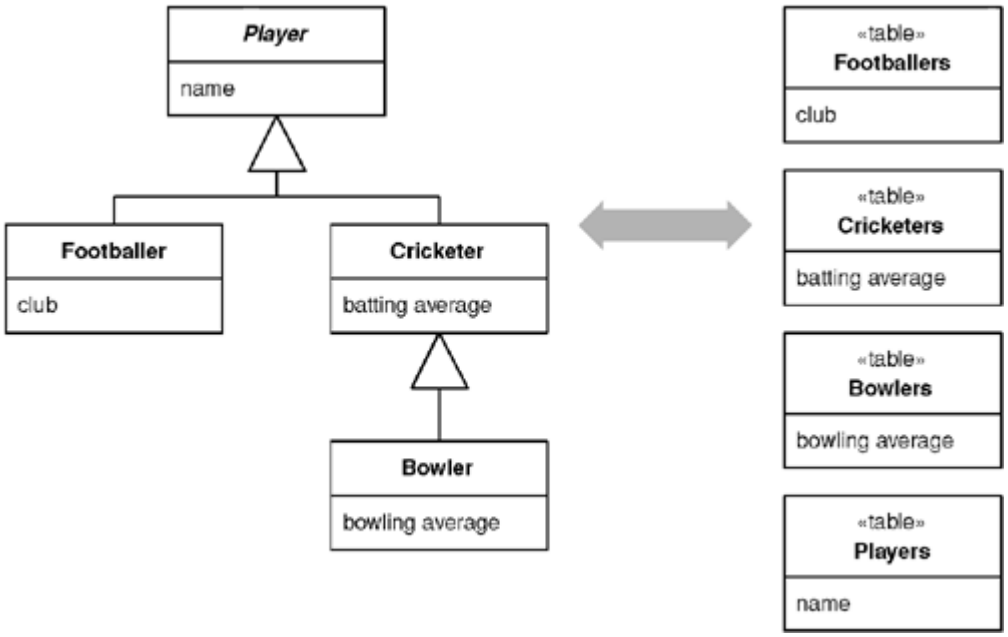


Figura 3-41: Class table inheritance

Descripción y Funcionamiento

Otro tipo de representación de la herencia de tablas en base de datos es la utilización de varias tablas, una por cada clase que participa en la jerarquía de herencia. Las tablas deben tener algún tipo de relación para soportar la herencia y esto se puede realizar de dos formas:

[139] Martin Fowler, *op. cit.*

- Cada fila que se encuentra en la primera tabla padre del modelo tendrá una clave primaria y el resto de las tablas tendrán el mismo valor que la tabla de la cual es dependiente, éste valor será único entre todas las tablas participantes.
- Cada fila en cada tabla tendrá una clave primaria diferente y para relacionarlas se usan claves foráneas.

Ventajas

- No se desperdicia espacio en los campos de las tablas ya que cada tabla tiene la información que le pertenece.
- Se pueden utilizar nombres iguales de campos a través de las tablas.

Desventajas

- La carga de datos puede ser compleja y requerir de muchas llamadas a la base de datos, ya que por cada tabla del modelo se tiene que ir cargando de una en una para respetar la jerarquía.
- Las tablas de los niveles superiores pueden convertirse en cuellos de botellas ya que para acceder a los últimos niveles siempre se tendrá que pasar por una la tabla superior.

3.3.3.15. Query Object[140]

Intención

Un objeto que representa una consulta hacia una base de datos.

Representación

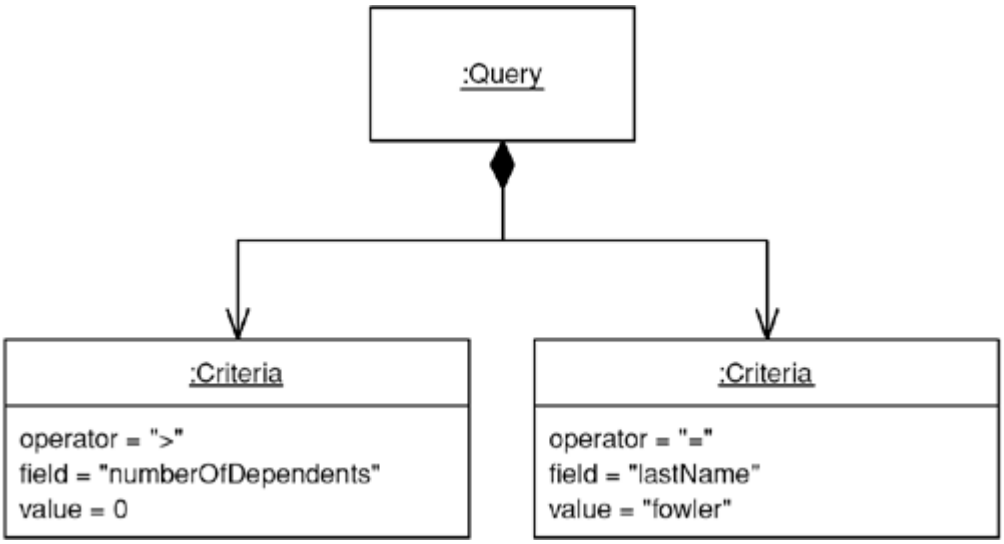


Figura 3-42: Query object[141]

Descripción y Funcionamiento

Un objeto query object es un ejemplo de implementación del patrón de programación Interpreter, está diseñado para crear sentencias sql dinámicas. La sentencia incluye qué campos se requiere retornar, los criterios y operadores, niveles de agrupamiento, etc.

[140] Martin Fowler, *op. cit.*
[141] Íd., Ibíd.

Un query object genera las sentencias basadas en los campos del objeto que genera la consulta; si los campos del modelo relacional y el modelo de objetos coinciden no se necesita un mapeo adicional, caso contrario se puede utilizar cualquier técnica de mapeo como mapeo de metadatos por ejemplo.

Adicionalmente se puede estructurar el query object para que sea parametrizable y genere consultas diferentes dependiendo de la base de datos que se requiera consultar.

Existe software de terceros que implementa éste patrón y puede llegar a ser bastante completo. Un caso práctico para la generación de SQL puede ser para reportes dinámicos y consultas complejas en las que dependiendo de las necesidades se vaya construyendo la consulta

3.3.4. Patrones de Presentación Web

Una de las partes más fundamentales y robustas de las aplicaciones empresariales son las interfaces de usuario basadas en Web. Son tan versátiles que no es necesario instalar ningún tipo de software en la parte del cliente y sobretodo se puede llegar a tener un acceso universal hacia la aplicación basada en Web.

Generalmente se colocan estas aplicaciones en un servidor Web para que éste gestione las peticiones de los clientes y envíe respuestas hacia los mismos mediante páginas de contenido dinámico que pueden ser scripts CGI, páginas servlets de java o páginas de servidor como ASP.net o JSP.

Los patrones de presentación Web nos hablan de cómo separar estas interfaces de la lógica de dominio, logrando que las aplicaciones sean más escalables y que al momento de hacer un cambio a cualquiera de los extremos no se tenga que cambiar todo el modelo.

Al separar éstas capas se consigue tal independencia para la parte de la vista que incluso la gente que diseña las interfaces Web no necesita saber nada de programación y puede continuar su trabajo haciendo interfases más amigables, mientras que al otro extremo se encuentran los programadores que se dedican únicamente a diseñar la lógica de dominio, plasmando las reglas de negocio y gestionando conexiones hacia las fuentes de datos.

3.3.4.1. Modelo Vista Controlador[142]

Intención

Separa la interfase de usuario en tres roles diferentes: el modelo, la vista y el controlador.

Descripción y Funcionamiento

El modelo vista controlador MVC tiene dos funciones importantes: separar la capa de presentación de la lógica de dominio y separar el controlador de la vista. Para entender mejor éstas premisas podemos dar los conceptos de cada componente:

[142] Martin Fowler, *op. cit.*

El modelo: El modelo representa la información que se desea extraer del dominio, en sí representa al objeto que se encuentra dentro del modelo, un ejemplo sería un objeto Cliente, el modelo en éste caso es en sí la información de un objeto Cliente.

La Vista: Representa cómo se muestra el modelo en la capa de presentación. Le provee al usuario de toda la información necesaria para ver e interactuar con el modelo.

El controlador: Cuando la información de la vista cambia el controlador es el encargado de recuperar los cambios generados por el usuario y procesarlos; es decir toda la interacción que el usuario genera hacia la aplicación es recibida por el controlador quien a su vez llama a los objetos del modelo para realizar todo el procesamiento necesario que viene de ésta interacción.

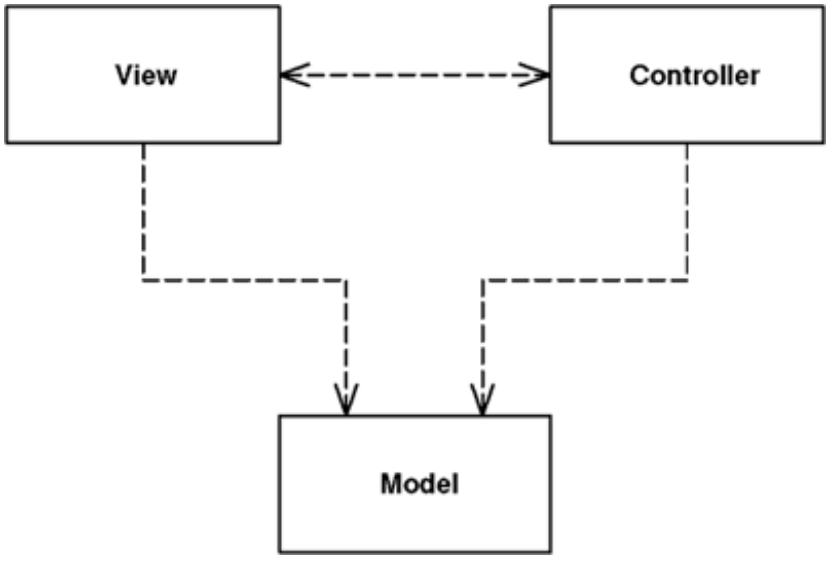


Figura 3-43: Modelo, Vista y Controlador[143]

[143] Martin Fowler, *op. cit.*

La separación se da principalmente entre el modelo y la presentación en donde el modelo no depende de la presentación pero la presentación si depende del modelo. Pueden existir varios tipos de presentaciones para un mismo modelo para diferentes casos.

La siguiente separación tiene relación con la vista y el controlador. La vista se encarga de realizar el proceso de formateo para que una vista genere la interfase Web, sin embargo un proceso muy diferente (el controlador) será el encargado de recibir los cambios que vienen de la interfase generada anteriormente. Sin embargo ésta separación no siempre se da ya que se puede combinar la funcionalidad de ambas partes en un solo esquema.

3.3.4.2. Page Controller[144]

Intención

Un objeto que controla las peticiones hacia una página o acción en un sitio Web.

Representación

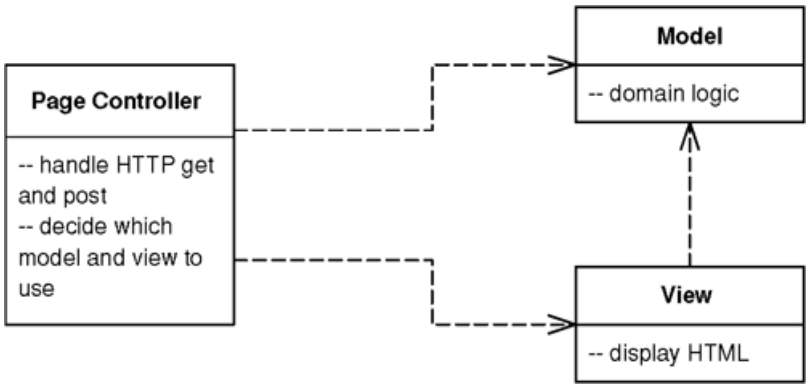


Figura 3-44: Page controller[145]

[144] Martin Fowler, *op. cit.*
[145] Íd., Ibíd..

Descripción y Funcionamiento

Un page controller es creado como un módulo de un servidor Web para atender las peticiones que se generan hacia una página, existe un page controller por cada página.

Las acciones que realiza un page controller son:

- Leer el URL y los datos del formulario que vienen desde la petición del cliente, concretamente de un explorador de Internet.
- Organizar los datos recibidos para poder enviarlos a los objetos de la lógica de dominio y comenzar el procesamiento de toda la información recibida. En este esquema también se pueden usar módulos de lógica de dominio encapsulados en Transaction Scripts, para ayudar al procesamiento de un page controller.
- Después del procesamiento el page controller determina qué vista desplegará el resultado de la operación.

3.3.4.3. Front Controller[146]

Intención

Un objeto que controla todas las peticiones hacia un sitio Web.

[146] Martin Fowler, *op. cit.*

Representación

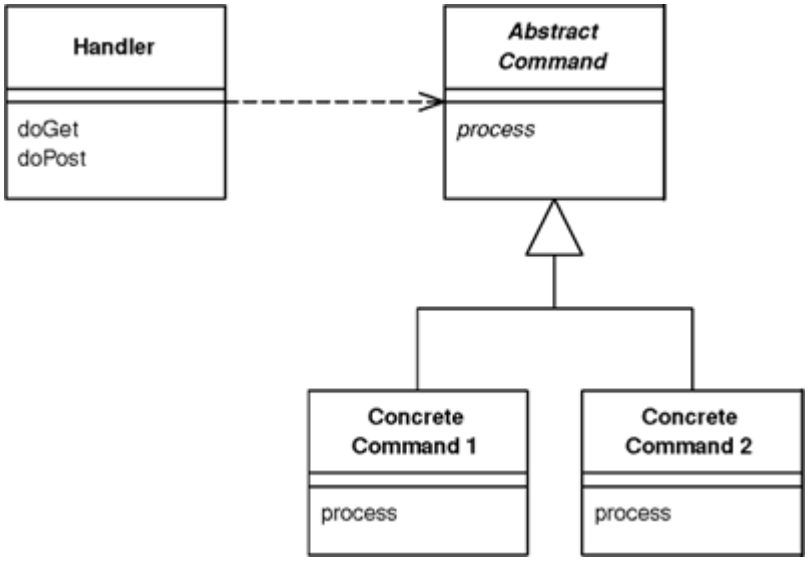


Figura 3-45: Front controller[147]

Descripción y Funcionamiento

El front controller es un objeto centralizado que se encarga de procesar y canalizar todas las peticiones generadas hacia una aplicación Web. El objetivo principal de tener centralizado en control de peticiones es que se puede incorporar un comportamiento común para todas las páginas de la aplicación, como seguridad, encriptación, compresión, caché de datos e internacionalización. De otra forma, sin un front controller, habría que encapsular toda ésta funcionalidad en cada página del sitio Web, lo cual si es una aplicación relativamente pequeña no estaría del todo mal, ya que la implementación de un front controller es una tarea más complicada.

[147] Martin Fowler, *op. cit.*

Un front controller está compuesto de dos partes fundamentales, un objeto handler que recibe como tal las peticiones generadas y objetos command que realizan la operación correspondiente que se invoca. El handler recibe la petición, determina el tipo de operación a realizarse y construye el comando específico. Para que el handler sepa qué comando invocar se puede programar en el mismo código una lógica condicional de forma estática; y una segunda forma de tipo dinámica que lee archivos de configuración y determina qué comando instanciar dependiendo, por ejemplo, del contenido de la dirección URL, ésta forma es más extensible ya que se puede seguir agregando comandos sin alterar el handler.

Una vez que el comando es ejecutado por el handler éste realiza todo el procesamiento necesario y es el encargado de seleccionar en qué vista mostrará el resultado. Con esto se separa que el controlador sea completamente independiente de la parte de la vista y en general la respuesta hacia el cliente.

3.3.4.4. Template View[148]

Intención

Construye información en HTML mediante marcadores colocados en una página o plantilla HTML.

[148] Martin Fowler, *op. cit.*

Representación

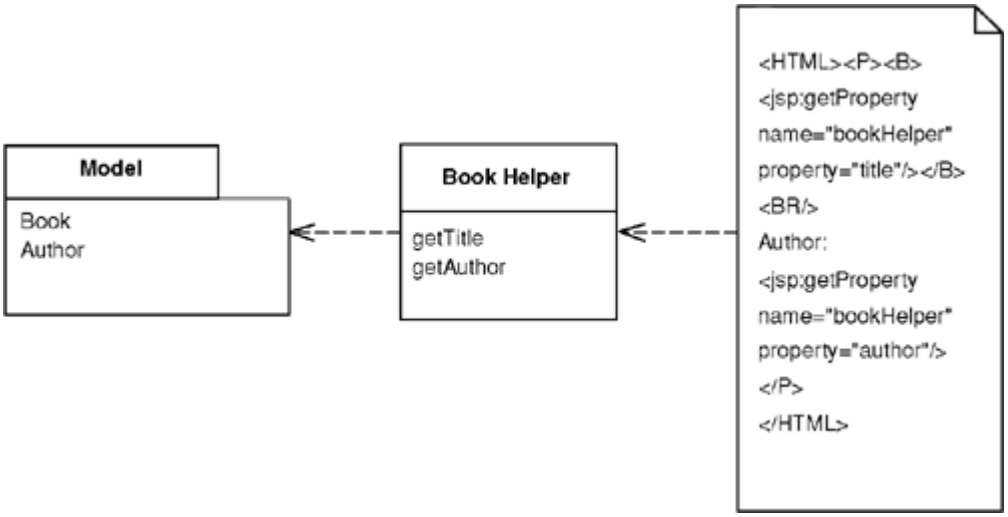


Figura 3-46: Template view[149]

Descripción y Funcionamiento

Existen dos tipos de páginas de contenido HTML: en su forma más simple los documentos estáticos, los cuales son generados por editores HTML y su contenido no cambia de petición a petición; el otro tipo son las páginas dinámicas que son generadas enteramente por el servidor y cambian entre petición y petición. Para construir páginas dinámicas lo más sencillos es combinar código HTML estático con secciones de código programables dentro de marcadores; así el contenido que se encuentra dentro de los marcadores es generado por el servidor y así se puede construir código HTML dinámico que puede ser algún cálculo, consulta hacia una base de datos, etc. Es así que el código HTML estático viene a convertirse en una plantilla HTML, de ahí el nombre de este patrón.

[149] Martin Fowler, *op. cit.*

Los marcadores pueden verse como tags HTML tradicionales para mejor comprensión y éstos luego serán reemplazados por el contenido dinámico. Dentro de los marcadores se puede directamente reemplazar por contenido generado desde el servidor o incluso se puede colocar algún tipo de lógica de programación denominado Scriptlets. Esta forma es utilizada en las páginas de servidor más conocidas como: ASP, PHP y JSP.

Código 3-3: Ejemplo de código ASP[150]

```
<table border="0" cellpadding="2" cellspacing="3" >

      <tr>
        <td ><font face="Verdana" size="2">Name :</font></td>
        <td ><%=name%></td>
      </tr>
      <tr>
        <td ><font face="Verdana" size="2">E-mail :</font></td>
        <td ><%=email%></td>
      </tr>
      <tr>
        <td ><font face="Verdana" size="2">Website :</font></td>
        <td ><%=website%></td>
      </tr>
      <tr>
        <td ><font face="Verdana" size="2">Website name :</font></td>
        <td ><%=websitename%></td>
      </tr>

      <tr>
        <td ></td>
        <td >

        <form action=addnewmember.asp method=post>
        <input type=hidden name=name value="<%=name%>">
        <input type=hidden name=email value="<%=email%>">
        <input type=hidden name=website value="<%=website%>">
        <input type=hidden name=websitename value="<%=websitename%>">
        <input type=submit value="Confirm">
        </form>

        </td>
      </tr>

    </table>
```

[150] http://www.planet-source-code.com/

El problema de utilizar lógica de programación dentro de la página HTML es que el código queda tan mezclado que es difícil de mantener, no puede ser editado fácilmente por la persona que realiza el diseño gráfico de la página ya que no se distingue qué partes son de HTML tradicional y qué partes pertenecen al lenguaje de programación que se utiliza para los scriptlets. Sin embargo, esto puede ser evitado colocando en los scriptlets solo la llamada hacia los objetos "ayudantes" y serán éstos lo que realmente contengan la lógica de programación dejando a la plantilla Tm. solo con lo correspondiente al HTML.

Hablando sobre el modelo vista controlador Template view cumple el rol de vista principalmente, sin embargo puede cumplir el rol de controlador e inclusive de modelo pero siempre será recomendado que se separe cada rol en diferentes componentes.

La ejecución de la parte de la vista necesariamente debe ser interpretada por un servidor Web, éste reemplaza las partes dentro de la plantilla donde debe colocar el código HTML generado. Para esto se requiere que la página se compile y esto puede ocurrir en cualquier instancia: cuando la página es creada por primera vez, cuando la página es ejecutada en la primera petición de un cliente, o incluso en cada petición de cada cliente lo cual no es recomendable a menos que sea un procesamiento bastante sencillo y no demande un compilación muy larga.

3.3.4.5. Transform View[151]

Intención

Una vista que procesa los datos de los elementos de un dominio y los transforma en HTML.

Representación

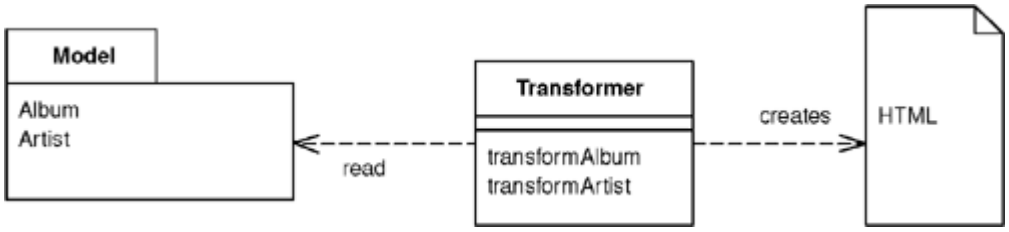


Figura 3-47: Transform view[152]

Descripción y Funcionamiento

Un objeto transform view transforma la estructura de un objeto de dominio en código HTML, y por cada elemento que se encuentre dentro de este objeto lo transforma en etiquetas Tm. y le coloca el formato deseado.

El método más común y usado en la mayoría de aplicaciones Web que utilizan éste patrón es el uso de XSLT, un lenguaje de programación que toma como entrada datos XML y mediante una plantilla XSLT combina el XML de entrada en HTML. Entonces si quisiéramos utilizar este método bastaría que los

[151] Martin Fowler, *op. cit.*
[152] Íd., Ibíd..

objetos del dominio tuviesen un método que transforme sus datos almacenados en XML, luego mediante la plantilla XSLT el objeto transform view convertiría el XML devuelto por el objeto en código HTML y devolvería la respuesta al cliente.

Existen factores importantes al momento de utilizar este modelo de procesamiento, el primero es al momento de tener varios tipos de salida hacia diferentes tipos de clientes, basta con cambiar la plantilla XSLT se genera una interfase diferente formateada hacia las necesidades específicas del tipo de cliente. Otro factor importante es que no se necesita de un servidor Web para probar el resultado de la transformación lo que resulta más simple al momento de desarrollo.

Código 3-4: Ejemplo de un código XSLT[153]

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0"
                xmlns="http://www.w3.org/1999/xhtml">

   <xsl:template match="card[@type='simple']">
     <html xmlns="http://www.w3.org/1999/xhtml">
       <title>business card</title><body>
         <xsl:apply-templates select="name"/>
         <xsl:apply-templates select="title"/>
         <xsl:apply-templates select="email"/>
         <xsl:apply-templates select="phone"/>
     </body></html>
   </xsl:template>

   <xsl:template match="card/name">
     <h1><xsl:value-of select="text()"/></h1>
   </xsl:template>

   <xsl:template match="email">
     <p>email: <a href="mailto:{text()}"><tt>
       <xsl:value-of select="text()"/>
     </tt></a></p>
   </xsl:template>

   ...
 </xsl:stylesheet>
```

[153] http://www.brics.dk/~amoeller/XML/xslt-4.1.html

3.3.4.6. Application controller[154]

Intención

Representa un punto centralizado para el control del flujo de las interfaces de usuario de una aplicación.

Descripción y Funcionamiento

Algunas aplicaciones requieren un flujo complejo de interfaces de usuario en las que un proceso debe seguir un orden y jerarquía en especial, para esto se establece un punto centralizado de control en el cual todo ese flujo es manejado por uno o varios controladores el cual conforman un controlador de aplicación.

El controlador provee la lógica de las interfaces que son desplegadas al usuario creando un flujo controlado que puede ir variando dependiendo de las variables de entrada que vienen del usuario.

Los controladores responden a las peticiones del usuario desplegando la interfase correspondiente para cada comando requerido; los comandos por lo general se conectan con el modelo o lógica de dominio o a otros controladores de página o de transformación; recopilan todos los datos necesarios y son enviados a través de la interfase de usuario correspondiente.

[154] Microsoft Patterns and Practices, User Interface Process Application Block Help

Un controlador de aplicación puede o no ser dependiente de una interfase de presentación (Web, Windows, dispositivos móviles) o puede ser genérica para cualquier caso, lo cual es bastante recomendable, así un controlador de aplicación estaría como una capa intermedia entre la presentación y el modelo o lógica de negocio.

El controlador de aplicación puede también ser un punto centralizado para guardar y mantener la sesión de un usuario, especialmente en las aplicaciones Web en la que un usuario al visitar la aplicación debe ser reconocido e identificado para que pueda mantener un flujo dentro de la interacción con la aplicación. Es así que el usar un controlador nos puede dar los siguientes beneficios relacionados con la sesión:

- Resumir una sesión Web: permite al usuario guardar cierta información en el explorador para que en la siguiente visita pueda continuar trabajando exactamente donde se quedó.
- Transferencia de una sesión: Ayuda al usuario a guardar una sesión y poderla continuar desde otra ubicación o computador o incluso en otro tipo de dispositivo.

3.3.5. Patrones de Distribución

3.3.5.1. Remote Facade[155]

Intención

Provee una fachada con menos nivel de granularidad hacia objetos con alto nivel de granularidad con el objetivo de optimizar las llamadas entre objetos dentro de una red.

Descripción y Funcionamiento

En la orientación a objetos la mejor práctica es tener objetos pequeños en donde cada uno de ellos se encarga de una tarea en específico y así se consigue aplicaciones más escalables y fáciles de utilizar, sin embargo hacerlos interactuar a éstos objetos unos con otros demanda una gran cantidad de invocaciones y llamadas. Esto tiene una gran repercusión cuando se utilizan procesos y objetos remotos, ya que por cada llamada entre ellos se requiere que se compruebe seguridad, encriptación, que los paquetes de datos sean construidos y enviados a través de la red, y otros factores importantes de rendimiento que hacen que una aplicación distribuida sufra el impacto de tantas transacciones e invocaciones entre objetos.

[155] Martin Fowler, *op. cit.*

El objetivo principal de este patrón es proveer una interfase para que las llamadas hacia estos componentes con alto nivel de granularidad pasen primero por una interfase o fachada que minimice el número de llamadas remotas a los objetos. Por ejemplo si quisiéramos retornar un objeto Factura y su detalle se cargarían ambos en una sola llamada independientemente de si están o no en diferentes objetos (Factura y Detalle Factura), otro ejemplo es si queremos llenar las propiedades de un objeto mediante una fachada remota se llenan todas las propiedades de una sola vez y luego ésta fachada se encarga de llenar una a una las propiedades del objeto que está al otro extremo.

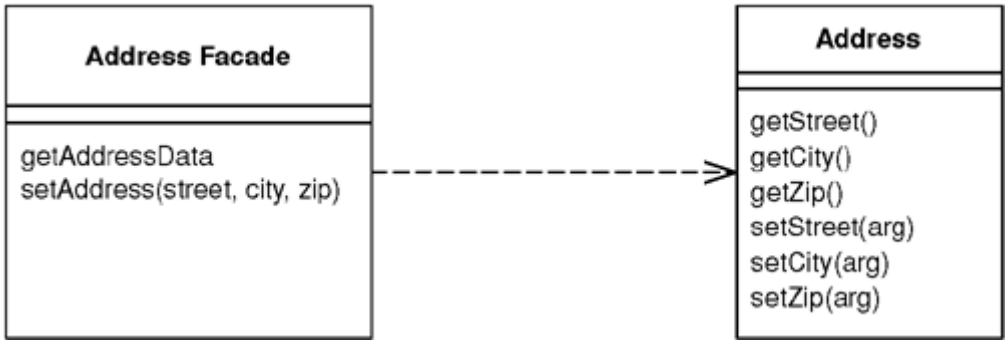


Figura 3-48: Remote facade[156]

Se pueden construir fachadas remotas por cada punto funcional de una aplicación; es decir puede existir una clase FachadeVentas, que incluya los métodos para controlar, órdenes, facturas y clientes. Cada uno de estos métodos servirá para recibir y enviar datos de forma "masiva" hacia los objetos internos.

La fachada remota puede contener varios servicios adicionales como manejar la seguridad y transaccionalidad de las peticiones que se originan de los

[156] Martin Fowler, *op. cit.*

clientes externos. Puede contener políticas de acceso y encriptación. En cuanto a transaccionalidad puede iniciar una transacción, operar con todos los objetos internos y finalizar la transacción.

Debe quedar muy claro que una fechada remota no debe contener el código de la lógica de dominio, la fachada lo único que hace es ser la interfase para operar los objetos internos y la lógica debe estar siempre almacenada ya sea en un Domain Model, un transaction script o en cualquier patrón de organización de lógica de dominio. La intención es que independientemente de si se utiliza o no una fachada remota la lógica de dominio de la aplicación debe funcionar en ambos casos.

Una fachada remota es generalmente usada para invocar métodos remotos en procesos que están en diferentes servidores, sin embargo se la puede utilizar en procesos de un mismo servidor si la comunicación entre estos procesos es compleja o larga.

Este patrón aporta rendimiento ya que permite recoger múltiples datos y enviarlos en una única llamada, reduciendo el número de conexiones que el cliente tiene que hacer al servidor.

3.3.5.2. Data Transfer Object[157]

Intención

Transfiere datos entre procesos con la finalidad de reducir el número de llamadas entre ellos.

Representación

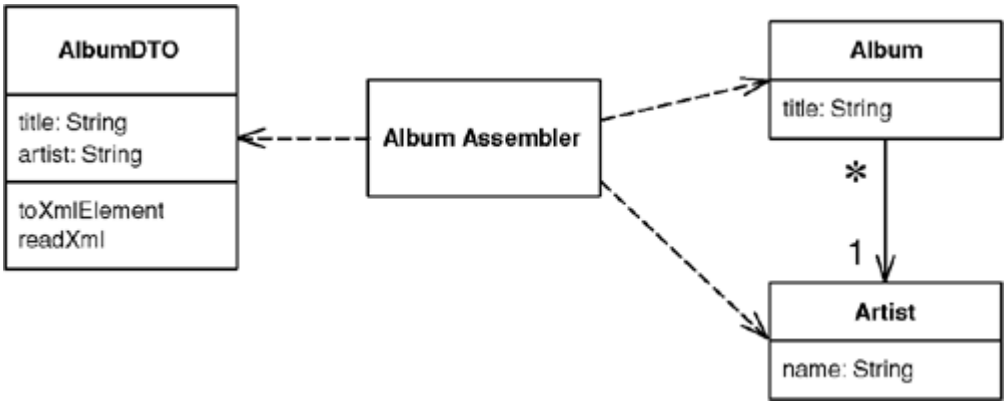


Figura 3-49: Data transfer object[158]

Descripción y Funcionamiento

Es también denominado Value Object, la intención principal de un Data Transfer Object es recoger múltiples datos y enviarlos en una única llamada, reduciendo el número de conexiones que el cliente tiene que hacer al servidor aportando significativamente al rendimiento de invocaciones remotas. Object contiene toda la información necesaria para que el objeto que hace la llamada no

[157] Martin Fowler, *op. cit.*
[158] Íd., Ibíd.

necesite hacer más llamadas a otros métodos para obtener más información, y es por esto que un data transfer object suele ser bastante pesado.

El objeto es construido por el servidor con los datos requeridos por el cliente, el servidor serializa éste objeto para que pueda ser transferido a través de la red. Se coloca un objeto ensamblador que transforma los datos de los objetos de negocio en datos serializados, los valores son asignados en forma local, son enviados por la red y luego son recuperados en forma local en la otra maquina.

Los datos que se transfieren dentro del Data Transfer Object generalmente no son directamente los objetos de la lógica de dominio sino más bien solo la información relevante de cada uno de ellos, sin ninguna clase de código o lógica que éstos pudieran tener. La información vital de cada objeto es copiada a los campos que pertenecen al Data Transfer Object y sus valores son asignados generalmente por métodos get y set; es así que la el objeto Data Transfer Object lo único que tiene son valores de strings, fechas, y tipos numéricos pero nada más.

3.3.6. Patrones de estado de Sesión

3.3.6.1. Client Session State[159]

Intención

Guarda el estado de la sesión en el cliente.

[159] Martin Fowler, *op. cit.*

Descripción y Funcionamiento

Es completamente necesario que el cliente guarde al menos una pequeña parte de la sesión que debe mantener con el servidor. El cliente debe guardar al menos el identificador único de sesión que lo distingue en el servidor, o también podría guardar completamente todos los objetos de sesión y así se liberaría al servidor de almacenar la sesión.

Cuando se trata de aplicaciones de escritorio se puede almacenar en clases de estado en la memoria del cliente; pero cuando se trata de aplicaciones Web almacenar la sesión se torna bastante más complicado, la sesión debe ser almacenada por mecanismos del explorador de Internet que pueden ser las siguientes:

Parámetros en la dirección URL

Es un mecanismo bastante simple y útil para cuando se desea almacenar datos pequeños. Se trata de colocar en la dirección URL parámetros que luego serán recuperados y procesados por el servidor. Es muy utilizado en páginas de servidor como ASP, JSP y PHP. Podemos ver el siguiente ejemplo:

http://www.compras.com/productos/vender.aspx?sessionid=78798azsad7zxx78

El caso anterior es un caso bastante simple en el que se le indica al servidor el identificador de la sesión del cliente y así este puede identificar únicamente al cliente que esta realizando la compra. No solo se puede almacenar el identificador

de la sesión sino cualquier cantidad de parámetros que sean necesarios, se los puede separar por el caracter &:

http://www.compras.com/productos/vender.aspx?sessionid=78798azsad7zxx78&ProductID=23&cantidad=24

Este método es bastante útil sin embargo tiene limitaciones ya que no se puede colocar más de 256 caracteres en la barra de dirección, otra limitación es no poder guardar estructuras de datos ni objetos en el URL ya que solo se puede almacenar tipos numéricos o texto. Un problema adicional es la seguridad ya que la información que está en el URL puede ser fácilmente observada y alterada.

Campos HTML escondidos

En este método se usa un campo de formulario HTML escondido el cual es representado por la etiqueta <INPUT type = "hidden">. En este campo se puede colocar cualquier tipo de información y no es visible al usuario, a menos que se trate de un usuario un poco más experimentado en cuyo caso podría acceder a la opción "ver código fuente de la página" del explorador de Internet; con esta opción queda revelado el código HTML de la página y se muestran las etiquetas <INPUT type = "hidden"> las cuales podrían interpretarse por el usuario y ser alteradas guardando la página, cambiando los valores y enviando dicha página de regreso hasta el servidor.

Cookies

Los cookies son archivos tipo texto que son colocados en el cliente en los que se almacena cualquier tipo de contenido de sesión. Es muy usado en ASP para almacenar el identificador de sesión con el servidor, pero también se lo puede usar para guardar toda la información de sesión con el servidor. El problema básico de éste mecanismo es que el cliente puede tener deshabilitada la función de cookies y no se podría almacenar ningún dato en el cliente.

El almacenamiento de la sesión en el cliente trae muchas ventajas ya que ayuda a no sobrecargar al servidor con los datos de sesión de cada cliente. Sin embargo hay un costo adicional si se almacena demasiada información en el cliente ya que se tiene que estar transfiriendo gran cantidad de información entre petición y petición entre llamada y llamada al servidor. La seguridad es un factor crítico al almacenar la sesión en el cliente, especialmente en las aplicaciones Web, y es vital la encriptación de los datos que se almacenan en el cliente; sin embargo esto trae problemas de rendimiento ya que por cada petición al servidor se debe encriptar y desencriptar los datos del cliente.

3.3.6.2. Server Session State[160]

Intención

Guarda el estado de la sesión en el servidor de forma serializada.

---

[160] Martin Fowler, *op. cit.*

Descripción y Funcionamiento

Un servidor de aplicaciones puede guardar el estado de la sesión de un cliente. El cliente debe identificarse únicamente en el servidor guardando el identificador de la sesión. El servidor guarda en memoria toda la información que un cliente necesita almacenar ya sean éstos tipos de datos simples u objetos completos, estos datos se serializan de forma binaria o texto. Lo más utilizado para esto es la forma binaria en la que el servidor puede almacenar el contenido de la sesión en archivos de texto o en la memoria RAM disponible.

En el caso de las aplicaciones Web, el servidor Web almacena en su proceso de ejecución toda la información de la sesión de un cliente y lo más importante es que en las páginas de servidor como ASP, JSP y PHP la administración de la sesión es automática y no se necesita ningún tipo de programación ya que usando los objetos Session se puede hacer un Session.Add(object a) consiguiendo que un objeto se guarde en la memoria del servidor, ésta variable del objeto será independiente por cada cliente de la aplicación y será identificada por el identificador único en el cliente.

3.3.6.3. Database Session State[161]

Intención

Guarda el estado de la sesión en un servidor de base de datos.

---

[161] Martin Fowler, *op. cit.*

Descripción y Funcionamiento

La memoria de un servidor de aplicaciones siempre es un factor importante, tiene una capacidad limitada la cual se debe tomar en cuenta si la aplicación va a tener gran cantidad de clientes conectados y más aun si la información de sesión que almacena de dichos clientes es grande.

La información de la sesión es almacenada en tablas de una base de datos en forma serializada en donde cada fila de la tabla representa la sesión de un cliente conectado; ésta sesión debe expirar en algún momento y la fila donde se almacenó dicha sesión debe ser borrada; esto se realiza a través de un proceso de daemon de la base de datos que va eliminando y comprobando las sesiones que son finalizadas, caso contrario la tabla de almacenamiento de la sesión llegaría a ser demasiado grande y con registros sin uso.

La desventaja más notoria en el uso de este patrón es que el rendimiento de la aplicación se ve un poco afectado debido al número de veces que la servidor de aplicaciones debe conectarse a la base de datos; esto puede ser mejorado un poco usando caché para los objetos que ya hayan sido recuperados para la sesión.

3.4. Análisis de los Patrones de Software

En el estudio de los patrones de software se pudo dar a conocer a profundidad la definición, características y aspectos funcionales y de los patrones de software. Ciertos patrones dentro del estudio cuentan con ejemplos prácticos del mundo real y en la mayoría de ellos se puede brindar al lector los beneficios y consecuencias de la utilización de patrones.

Para brindar una descripción y análisis más general de cada patrón se seleccionaron criterios comunes de evaluación que permitirán construir una tabla de comparación y mostrar gráficamente las características de cada uno de los patrones del estudio.

Para la construcción del framework de desarrollo planteado se seleccionará un grupo de patrones de diseño y arquitectura del estudio que serán acoplados para las necesidades del framework. Para seleccionar un patrón se tomará en cuenta la experiencia obtenida del estudio y también en base a la tabla de análisis de cada patrón.

3.4.1. Criterios de Evaluación de los Patrones

Los criterios para el análisis de los patrones serán los siguientes:

3.4.1.1. Aplicable a la tecnología .net

Este criterio busca medir el nivel en el que el patrón evaluado será posible de implementar utilizando la tecnología .net de Microsoft. Tanto a nivel del lenguaje de programación utilizado como a nivel del framework de .net.

3.4.1.2. Aplicable en Aplicaciones Distribuidas

Este criterio de evaluación permite evaluar si la implementación del patrón es posible en aplicaciones distribuidas. Esto es necesario ya que el ámbito del framework será en una aplicación distribuida empresarial.

3.4.1.3. Implementabilidad en el Framework

Este criterio permite evaluar que tan fácil es utilizar este patrón e implementarlo en el framework.

3.4.1.4. Aplicable en OOP

Este criterio busca evaluar si el patrón puede ser implementado utilizando lenguajes orientados a objetos.

3.4.1.5. Criterios de Calidad de Software

Los siguientes criterios de evaluación corresponden a la norma ISO:9126 que trata acerca de la calidad del software en el proceso y el producto final. Los patrones de software buscan mejorar la calidad en el software, tanto en su desarrollo como en el producto final, por esta razón se evalúan cada patrón con los criterios dictados por esta norma.

3.4.1.5.1. Portabilidad

Este criterio buscar realizar una evaluación sobre la facilidad de adaptación, instalación y reemplazo de un componente desarrollado en base al patrón evaluado con respecto a otro que no fue desarrollado en base a patrones de software.

3.4.1.5.2. Mantenibilidad

Este criterio busca evaluar la capacidad de modificación o realización de correcciones, mejoras, o cambios en su funcionalidad de un componente basado en el patrón de software evaluado.

3.4.1.5.3. Usabilidad

Este criterio realiza una evaluación sobre la facilidad de comprensión y utilización de un patrón dado.

3.4.1.5.4. Funcionalidad

Este criterio busca medir el aporte de un patrón de software a la interoperatibilidad del framework.

3.4.1.5.5. Confiabilidad

Este criterio busca medir el aporte de los patrones de software para la solución de problemas referentes a fallas de software y la recuperación luego de estos problemas.

3.4.1.5.6. Eficiencia

Con este criterio se busca realizar una evaluación del rendimiento de un sistema relacionado con su tiempo de respuesta y su consumo de memoria.

3.4.2. Evaluación de los Patrones

La tabla a continuación permite la evaluación de los patrones de software; esta tabla permite ver en modo macro los beneficios o dificultades de cada patrón y servirá también como guía para la selección los patrones que serán implementados en el framework.

Cuadro 3-3: Evaluación de los patrones de software (Elaboración Propia)

			Aplicable a .net / Java	Aplicable en Aplicaciones Distribuidas	Implementabilidad en el Framework	Aplicable en OOP	Facilidad de Integración	Portabilidad	Mantenibilidad	Usabilidad	Funcionalidad	Confiabilidad
Patrones de Arquitectura	Domain Logic Patterns	Transaction script	🟢	🟢	🟢	🔴	🟢	N/A	🔴	🔴	N/A	🔴
		Domain model	🟢	🟢	🟢	🟢	🟢	N/A	🟢	🟢	N/A	🟢
		Table Module	🟢	🟢	🟡	🟢	🟡	N/A	🟡	🟢	N/A	N/A
		Service Layer	🟢	🟢	🟢	N/A	🟡	🟢	🟢	🟢	🟢	N/A
	Mapping to Relational Databases Patterns	Table Data Gateway	🟢	N/A	🟡	🟢	🟢	🟢	🟢	🟢	🟡	🟢
		Row Data Gateway	🟢	N/A	🔴	🟢	🟡	🟢	🟡	🟢	🔴	N/A
		Active Record	🟢	N/A	🔴	🟢	🟡	🟢	🟡	🟡	🔴	N/A
		Data Mapper	🟢	N/A	🟢	🟢	🟢	🟢	🟢	🟢	N/A	N/A
		Unit of Work	🟢	🟢	🟢	🟢	🟢	🟢	N/A	🟢	N/A	🟢
		Identity Map	🟢	N/A	🟢	🟢	🟡	🟢	N/A	N/A	N/A	🟡
		Lazy Load	🟢	🟢	🔴	🟢	🔴	🟢	N/A	🟡	N/A	🟢
		Identity Field	🟢	N/A	🟢	🟢	🟢	🟢	N/A	N/A	N/A	🟢
		Foreign Key Mapping	🟢	N/A	🟢	🟢	🔴	🟢	🟢	N/A	N/A	N/A
		Dependent Mapping	🟢	N/A	🔴	🟢	🔴	🟢	🟢	N/A	N/A	N/A
		Embedded Value	🟢	N/A	🟡	🟢	🟢	🟢	🔴	N/A	N/A	N/A
		Serialized Lob	🟢	N/A	🟢	🟢	🟢	🟢	N/A	N/A	🟢	N/A
		Single Table Inheritance	🟢	N/A	🟢	🟢	🟢	🟢	N/A	🟡	N/A	N/A
		Class Table Inheritance	🟢	N/A	🔴	🟢	🔴	🟢	N/A	🔴	N/A	N/A
		Query Object	🟢	N/A	🔴	N/A	🟢	🟢	N/A	🔴	🟢	N/A
	Web Presentation Patterns	Modelo Vista Controlador	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	N/A	🟢
		Page Controller	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟡	N/A	🟢
		Front Controller	🟢	🟡	🔴	🔴	🔴	🟢	🔴	🔴	N/A	🟡
		Template View	🟢	🟡	🟢	🔴	🟢	🔴	🟡	🔴	N/A	🟡
		Transform View	🟢	🟢	🔴	🟡	🔴	🟢	🟢	🟡	🟢	🔴
		Application controller	🟢	🟢	🟢	🟢	🟡	🟢	🟢	🔴	N/A	🟢
	Distribution Patterns	Remote Facade	🟢	🟢	🟢	🟢	🟡	🟢	🟢	🟡	🟢	N/A
		Data Transfer Object	🟢	🟢	🟢	🟢	🟡	🟢	🟢	🟢	🟢	N/A
	Session State Patterns	Client Session State	🟢	N/A	🟢	N/A	🟢	🟢	N/A	N/A	N/A	🟡
		Server Session State	🟢	N/A	🟢	N/A	🟢	🟢	N/A	N/A	N/A	🟡
		Database Session State	🟢	N/A	🟢	N/A	🟡	🟢	N/A	N/A	N/A	🟢
Patrones de Diseño	Creacionales	Factory Method	🟢	N/A	🟢	🟢	🟢	🟢	🟢	🟡	N/A	🟢
		Abstract Factory	🟢	N/A	🟡	🟢	🔴	🟡	🟢	🔴	N/A	🟢
		Builder	🟢	N/A	🔴	🟢	🟡	🟡	🟢	🟡	N/A	🟢
		Prototype	🟢	N/A	🟢	🟢	🟡	🟢	🟢	🟡	N/A	🟢
		Singleton	🟢	🟢	🟢	🟢	🟡	🟢	🟢	🔴	N/A	🟢
	Estructurales	Adapter	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟡	🟢	🟡
		Bridge	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟡	🟢	🟡
		Composite	🟢	🔴	🔴	🟢	🔴	🟡	🟢	🔴	N/A	🟡
		Decorator	🟢	🟡	🟢	🟢	🟢	🟡	🟢	🔴	🟢	🟡
		Facade	🟢	🟢	🟢	🟢	🟡	🟢	🟢	🟡	🟢	N/A
		Flyweight	🟢	🔴	🔴	🟢	🔴	🔴	🔴	🔴	N/A	N/A
		Proxy	🟢	🟢	🟡	🟢	🟢	🟢	🟢	🟢	🟡	🟡
	Comportamiento	Chain of Responsibility	🟢	🟢	🔴	🟢	🔴	🟡	🟢	🔴	N/A	🟢
		Command	🟢	🟢	🔴	🟢	🟡	🟡	🟢	🔴	N/A	🟢
		Iterator	🟢	N/A	🟢	🟢	🟢	🟡	🟢	🟢	N/A	N/A
		Mediator	🟢	🟢	🔴	🟢	🔴	🟢	🟢	🟡	N/A	N/A
		Memento	🟢	🟢	🔴	🟢	🔴	🟡	🟡	🔴	N/A	N/A
		Observer	🟢	🟢	🟡	🟢	🟡	🟢	🟢	🟢	N/A	N/A
		State	🟢	🟢	🔴	🟢	🟡	🟡	🟡	🟡	N/A	N/A
		Strategy	🟢	🔴	🔴	🟢	🔴	🔴	🔴	🔴	N/A	🟢
		Visitor	🟢	🟡	🔴	🟢	🔴	🔴	🟡	🔴	N/A	N/A
		Interpreter	🟢	🔴	🔴	🟢	🔴	🔴	🔴	🔴	N/A	🟢
		Template Method	🟢	🔴	🔴	🟢	🔴	🔴	🔴	🔴	N/A	🟢

🔴 BAJO 🟡 MEDIO 🟢 ALTO N/A NO APLICABLE

3.4.3. Selección de los Patrones

Para el desarrollo del framework planteado se seleccionó un grupo de patrones. Para la selección se tomó en cuenta el conocimiento obtenido del estudio de patrones y también la tabla de análisis de cada patrón en el punto anterior. Cada patrón seleccionado se ubica en una o más capas o bloques del framework. Los patrones seleccionados y su relación con cada bloque del framework se pueden ver en el Cuadro 3-4: Patrones seleccionados para el framework (Elaboración Propia).

3.4.4. Explicación de cada patrón seleccionado

Los patrones seleccionados servirán para la construcción del framework de desarrollo planteado. Cada patrón fue estudiado cuidadosamente, analizando sus beneficios directos que brindará al framework. La explicación de la selección de cada patrón es la siguiente:

3.4.4.1. Patrones de Diseño

3.4.4.1.1. Factory Method

Este patrón fue seleccionado para la implementación por la facilidad de comprensión, la facilidad de integración dentro del Framework definido. La utilización de este patrón permite al Framework una flexibilidad adicional en la creación de objetos de determinada clase, que debe ser determinado en tiempo de ejecución y este patrón de diseño fue desarrollado para ser utilizado en este tipo de casos.

Cuadro 3-4: Patrones seleccionados para el framework (Elaboración Propia)

			Seleccionado para Framework
Patrones de Arquitectura	Domain Logic Patterns	Transaction script	✗
		Domain model	✓
		Table Module	✗
		Service Layer	✓
	Mapping to Relational Databases Patterns	Table Data Gateway	✓
		Row Data Gateway	✗
		Active Record	✗
		Data Mapper	✓
		Unit of Work	✓
		Identity Map	✓
		Lazy Load	✗
		Identity Field	✓
		Foreign Key Mapping	✗
		Dependent Mapping	✗
		Embedded Value	✗
		Serialized Lob	✗
		Single Table Inheritance	✗
		Class Table Inheritance	✗
		Query Object	✗
	Web Presentation Patterns	Modelo Vista Controlador	✓
		Page Controller	✓
		Front Controller	✗
		Template View	✗
		Transform View	✗
		Application controller	✓
	Distribution Patterns	Remote Facade	✗
		Data Transfer Object	✓
	Session State Patterns	Client Session State	✓
		Server Session State	✓
		Database Session State	✓
Patrones de Diseño	Creacionales	Factory Method	✓
		Abstract Factory	✗
		Builder	✗
		Prototype	✗
		Singleton	✓
	Estructurales	Adapter	✓
		Bridge	✓
		Composite	✗
		Decorator	✗
		Facade	✓
		Flyweight	✗
		Proxy	✓
	Comportamiento	Chain of Responsibility	✗
		Command	✗
		Iterator	✗
		Mediator	✗
		Memento	✗
		Observer	✗
		State	✗
		Strategy	✗
		Visitor	✗
		Interpreter	✗
		Template Method	✗

✓ SELECCIONADO
✗ NO SELECCIONADO

3.4.4.1.2. Singleton

Este patrón de diseño es utilizado debido a que nos brinda ventajas en la utilización de memoria de la aplicación, es fácil de implementar en tecnología .net y es completamente aplicable en aplicaciones distribuidas.

3.4.4.1.3. Adapter

Este patrón fue seleccionado ya que los componentes que integran el Framework no siempre tienen interfaces compatibles, y es necesario que estos componentes interactúen entre ellos. Este escenario cumple con todas las características para la utilización de este patrón de diseño.

3.4.4.1.4. Bridge

La selección de este patrón se debe a que en una aplicación distribuida es necesario que los componentes puedan ser modificados independientemente de las capas superiores, y la implementación de un componente debe brindar la flexibilidad de ser cambiada sin afectar a el resto de la aplicación. La intención del patrón de diseño es acorde con el escenario sobre el que se define el Framework, por lo tanto este patrón ha sido seleccionado para ser utilizado dentro del Framework.

3.4.4.1.5. Facade

La intención de un Facade es el hacer más simple la utilización de los componentes, los componentes del Framework mantienen muchas opciones que no siempre son necesarias para el usuario, por lo que un Facade es la mejor opción para realizar esta interacción de una manera más simple, sin restarle funcionalidad en caso de ser necesario al Framework.

3.4.4.1.6. Proxy

La utilización de proxys en una aplicación distribuida es necesaria, ya que de esta manera se pueden optimizar los canales de comunicación utilizando proxys remotos, o proxys virtuales. Es por esta razón que este patrón de diseño fue seleccionado para su utilización.

3.4.4.2. Patrones de Arquitectura

3.4.4.2.1. Domain model

Domain model brinda un esquema de organización de lógica de negocio orientado a objetos y particularmente a entidades empresariales. La lógica de negocio del framework está dividida según cada ente del mundo real y este patrón implenta este criterio.

3.4.4.2.2. Service Layer

Este patrón fue seleccionado debido a la característica de separar la lógica de negocio en dos partes, la primera parte es una barrera o capa que separa a la lógica de negocio como tal coordinando el flujo de trabajo, notificación de eventos y servicios de asincronía y seguridad; la segunda capa es la implementación de la lógica de negocio.

El service layer también provee una interfase para que las capas superiores puedan comunicarse con la capa de lógica de negocio utilizando más de un transporte.

3.4.4.2.3. Table Data Gateway

Este patrón brindará el soporte para la implementación de las entidades empresariales, particularmente este patrón brinda la funcionalidad de tener varias instancias de los datos en una sola entidad.

3.4.4.2.4. Data Mapper

La implementación se este patrón es muy importante ya que permite realizar el mapeo objeto-relacional necesarios para convertir los datos de una base de datos hacia objetos fácilmente manipulables en memoria.

3.4.4.2.5. Unit of Work

Este patrón permite conseguir transaccionalidad y manejo del estado de las operaciones hacia la base de datos. Se asegura que las entidades empresariales puedan ser persistidas en la fuente de datos de forma controlada y con un registro de posibles errores.

3.4.4.2.6. Identity Map

Este patrón administra de mejor manera la memoria, logrando que una objeto pueda ser cargado una única vez y poder ser reutilizado. La aplicación práctica de este patrón se realiza para la reutilización de los transportes de comunicación entre las capas superiores y la capa de lógica de negocio.

3.4.4.2.7. Identity Field

Una vez realizado el proceso de mapeo objeto-relacional, con este patrón podemos guardar la correspondencia de los registros de la base de datos con las entidades empresariales del modelo de objetos.

3.4.4.2.8. Modelo Vista Controlador

La arquitectura del framework requiere necesariamente la separación de la capa de presentación de la lógica de dominio y separar el controlador de la vista, lo cual nos provee de manera adecuada este patrón.

3.4.4.2.9. Page Controller

Para la implementación del modelo vista controlador, es necesario uno o varios controladores de página. Este patrón permite que exista un controlador por cada página lo cual es mucho mas organizado que tener un solo controlador global.

3.4.4.2.10. Application controller

Para apoyar al modelo vista controlador, se implementa éste patrón, el cual ayuda a simplificar las tareas de administración de la sesión del usuario y el flujo de navegación entre interfaces de usuario.

3.4.4.2.11. Data Transfer Object

El modelo orientado a servicios del framework necesita de la implementación de mensajes. Los mensajes encapsulan todos los datos necesarios entre las capas y es precisamente este patrón que implementa este criterio.

3.4.4.2.12. Client Session State

En las aplicaciones Web, especialmente, es necesario poder reconocer a un usuario que utiliza la aplicación y poder interactuar con este. Este patrón explica las formas de cómo poder guardar información en el cliente y enviarla al servidor para identificarse en el servidor.

3.4.4.2.13. Server - Database Session State

Una vez que el cliente se identifica con el servidor, éste guarda en memoria toda la información que un cliente requiere almacenar. Se pueden conjugar éstos dos patrones para guardar de forma persistente (base de datos) o en memoria la sesión del usuario.

CAPÍTULO IV

4. DEFINICIÓN DEL FRAMEWORK

4.1. Descripción framework

4.1.1. Propósito

El framework contiene toda la funcionalidad de transporte de una aplicación empresarial, así como los puntos de ingreso para las conexiones a bases de datos, y extensión de la presentación de usuario.

El framework definido es denominado “Millwood” y será referenciado con ese nombre durante la explicación de su estructura, componentes y servicios.

Millwood está desarrollado sobre la versión 1.1 del .net Framework de Microsoft que soporta tres formas básicas de componentes remotos, que son:

- WebServices
- Remoting
- MSMQ

Existen versiones nuevas del .net framework que proveen los mismos servicios y transportes que la versión 1.1 con algunas mejoras en rendimiento y seguridad. Millwood podría ser implementado o migrado a las nuevas o futuras

versiones del .net framework siempre y cuando no afecte a todos sus componentes internos y su interacción.

El framework propuesto provee al usuario el esquema básico para la implementación de una aplicación empresarial distribuida, definiendo la estructura de clases, interfaces, servicios y la solución en general de la aplicación; provee también los puntos de extensión necesarios para la implementación de las partes específicas de la aplicación.

El esquema por capas y modular del framework permite también que el desarrollo de las capas principales de una aplicación (presentación, negocio y datos) pueda ser realizado por diferentes grupos de trabajo, permitiendo control en el equipo de desarrollo, cada equipo es el responsable de una capa de la aplicación pudiendo integrar cada una de ellas sin problemas.

4.1.2. Millwood en las aplicaciones empresariales

Millwood en su concepción está orientado a ser la base para la construcción de aplicaciones empresariales. Los patrones implementados, sobretodo los de arquitectura, corresponden a arquitecturas diseñadas para la construcción de este tipo de aplicaciones.

Millwood pretende cubrir aplicaciones empresariales cuyo objetivo principal sea el de automatizar un proceso de negocio, acceder, gestionar e interpretar información para tomar decisiones empresariales rápidas y eficaces y mantener los mayores niveles de satisfacción de sus clientes.

En general, este grupo de aplicaciones puede ser el siguiente:[162]

- Soluciones de inteligencia de negocio
- Aplicaciones de Gestión de Clientes (CRM)
- Aplicaciones de Portales Empresariales
- Servicios para entornos SAP
- Aplicaciones de Comunicación y Productividad Personal
- Servicios de Mensajería y Colaboración

4.1.3. Requerimientos

"Millwood" está implementado en su totalidad en tecnología Microsoft, los requisitos para una aplicación desarrollada sobre éste framework son:

Tabla 4-1: Requisitos mínimos para el servidor (host)

Hardware	Software
• Computador con procesador x86 mínimo 1.5 Ghz • 512 MB RAM • 3 gigas libres de disco duro	• Windows Server 2003 • Internet Information Services 6.0 • Message Queue Server • Microsoft SQL Server 2000

[162] Aplicaciones Empresariales, http://h20219.www2.hp.com/services/cache/9269-0-0-197-470.html

Tabla 4-2: Requisitos mínimos para clientes Windows

Hardware	Software
• Computador con procesador x86 mínimo 800Mhz • 256 MB RAM • 1 giga libre de disco duro	• Windows XP o Windows Server 2003 • Microsoft .Net framework 1.1

Tabla 4-3: Requisitos mínimos para clientes Web

Hardware	Software
• Computador con procesador x86 o x64 • 64 MB RAM	• Windows 9x, Me, XP, Windows Server 2003, Linux, Mac OS X • Internet Explorer, FireFox, Opera, Safari o Netscape.

4.1.4. Convenciones y nomenclatura

Todo el código de la aplicación estará escrito de acuerdo al Estándar de Codificación (Ver Anexo A: **¡Error! No se encuentra el origen de la referencia.**). Todos los nombres de variables, nombres de campos, tablas, assemblies, namespaces y los comentarios serán escritos en inglés.

4.2. Estructura del framework “Millwood”

4.2.1. Patrones de software utilizados dentro del framework

La siguiente tabla ilustra los patrones implementados para el framework y su relación con las capas o bloques de implementación dentro del framework.

Tabla 4-4: Patrones seleccionados y su relación dentro del framework[163]

			Bloque de Implementacion	Componente
Patrones de Arquitectura	Domain Logic Patterns	Domain model	Business Layer / UI Process	Business Actions / Operations
		Service Layer	Business Layer	Interface Transports
	Mapping to Relational Databases Patterns	Table Data Gateway	Business Layer	DataSet
		Data Mapper	Data Access Layer	DataAdapter, CLogicalConnection, SQLHelper, DALCs
		Unit of Work	Business Layer	DataSets
		Identity Map	UI Process	Transport Proxy
		Identity Field	Business Entities	DataSets
	Web Presentation Patterns	Modelo Vista Controlador	Presentation Layer / UI Process	ASP.NET, UIPAB
		Page Controller	Presentation Layer	ASP.NET
		Application controller	UI Process	UIPAB
	Distribution Patterns	Data Transfer Object	Messages	Messages
	Session State Patterns	Client Session State	Presentation Layer	ASP.NET
		Server Session State	Presentation Layer	ASP.NET
		Database Session State	UI Process	UIPAB
Patrones de Diseño	Creacionales	Factory Method	UI Process	Transport Proxy
		Singleton	UI Process	Transport Proxy
	Estructurales	Adapter	UI Process	Operations
		Bridge	UI Process / Data Access Layer	Transport Proxy, Tasks / CLogicalConnection
		Facade	UI Process / Data Access Layer	Controllers, Operations / SQLHelper
		Proxy	UI Process	Remote Proxy: Web Services Virtual Proxy:Transport Proxy

[163] Elaboración Propia

4.2.2. Diagrama de bloques

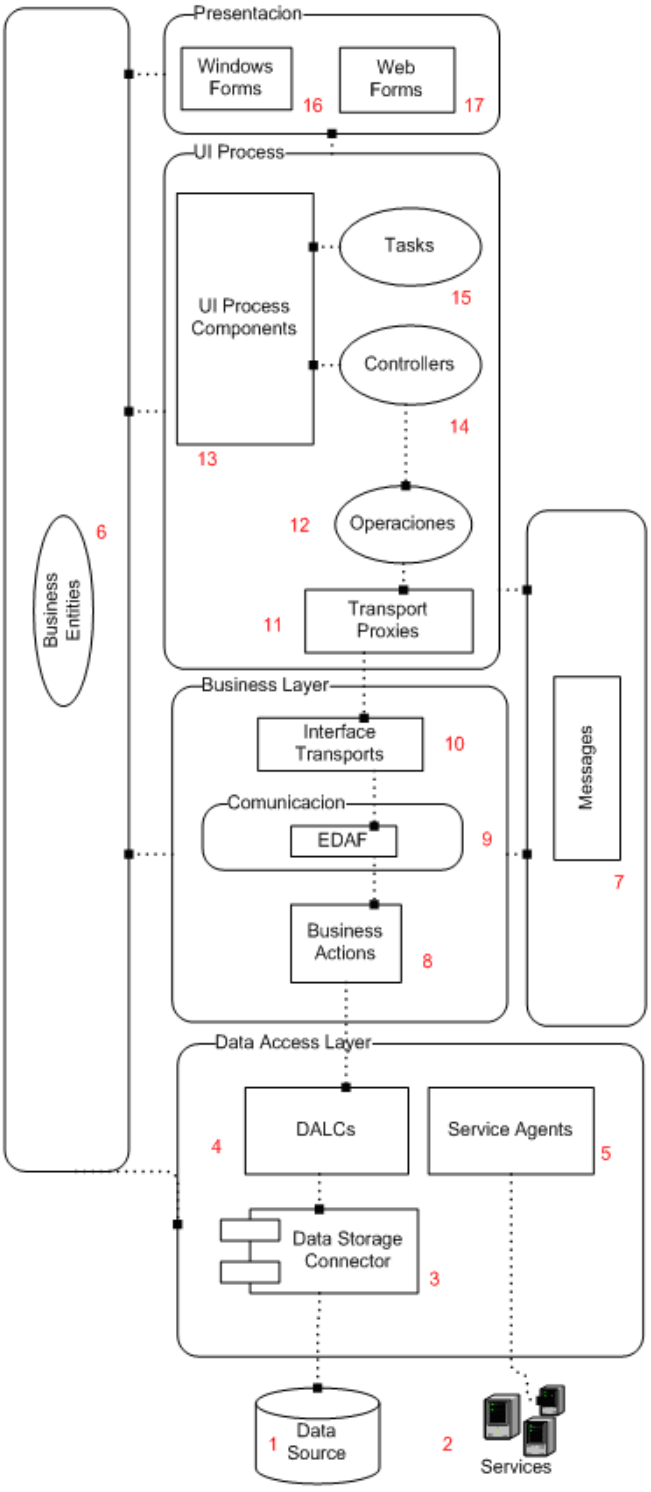


Figura 4-1: Diagrama de bloques "Millwood"[164]

[164] Elaboración Propia

Millwood está estructurado por bloques principales o capas (véase 3.3.1 Layers (Capas)) que contienen a su vez bloques más pequeños que implementarán toda la funcionalidad del framework mediante componentes, clases, etc. Cada bloque tiene una conexión directa hacia el siguiente bloque y así sucesivamente, sin embargo existen bloques que pueden ser compartidos o conectados entre dos o más de ellos (Business entities, mensajes). Cada bloque se comunica con el siguiente a través de referencias directas o utilizando protocolos de transporte (WebServices, remoting o MSMQ)

4.2.3. Fuentes de datos

4.2.3.1. Bases de Datos -1-

Las bases de datos son el almacenamiento del modelo de la aplicación. La base de datos está estructurada por tablas, vistas, funciones, procedimientos almacenados y tipos de datos personalizados.

4.2.3.2. Servicios Web XML -2-

El framework puede consumir servicios Web XML como fuente de datos, por éste medio se podría realizar el intercambio de información con otra aplicación o servicio empresarial a través de mensajes, estos mensajes envían y reciben datos que pueden ser manipulados en capas superiores.

4.2.4. Entidades Empresariales -6-

Las entidades empresariales representan entidades del mundo real necesarias para la aplicación. Las entidades están presentes en la mayoría de las capas de la aplicación y son muy útiles para transferir información ya sea en grandes o en pocas cantidades.

Las entidades empresariales pueden reflejar un modelo de objetos basado en la estructura de una o más tablas de la fuente de datos. Estas clases por lo general mantienen el estado de uno o más registros que vienen desde la fuente de datos a través de un proceso de mapeo objeto-relacional la cual es realizada por los Data access logic components en la capa de acceso a datos.

Relación con patrones de software

Las entidades empresariales son la aplicación de los siguientes patrones:

- Table Data Gateway (Ver 3.3.3.1 Table Data Gateway)
- Unit of Work (Ver 3.3.3.5 Unit of Work)
- Identity Field (Ver 3.4.4.2.7 Identity Field)

4.2.5. Capa de acceso de datos (Data Access Layer)

Esta capa de la aplicación es la encargada del acceso a datos de toda la aplicación desde un lugar centralizado. De esta manera es posible variar la capa de acceso a datos sin que se afecte ninguna otra parte de la aplicación. Mientras los componentes de acceso a datos cumplan con los requerimientos de las capas

superiores, el proceso e implementación de acceso a datos es completamente transparente para el resto de la aplicación.

Es posible reescribir los DALCs (ver 4.2.5.1 Data access logic components (DALCS)) y la clase de conexión lógica para utilizar otra fuente de datos diferente a SQL Server, por ejemplo ORACLE o MySQL, incluso orígenes de datos no convencionales como servicios Web, o archivos XML pueden ser utilizados sin la necesidad de afectar a otras capas de la aplicación lo que brinda la posibilidad de soporte para varias bases de datos, posibilidad de configuración de la misma y las características de escalabilidad, seguridad, y portabilidad a una aplicación con un mínimo esfuerzo.

Los componentes internos de la capa de acceso a datos son:

- Data access logic components (DALCS)
- Agentes de Servicio

4.2.5.1. Data access logic components (DALCS) -4-

Son las clases encargadas de realizar el mapeo Objeto-Relacional de la base de datos, en estas clases se define como se realiza el mapeo de las tablas y relaciones de la base de datos hcia relaciones de objetos. Estas clases son las encargadas de la definición de las diferentes operaciones que se puede realizar sobre la base de datos con una tabla específica. Debido a que cada tabla de la base de datos corresponde a un objeto en la aplicación, cada uno de estos componentes es específico para el mapeo entre una tabla y uno de estos objetos.

Relación con patrones de software

Los DALCs son parte del componente o patrón Data Mapper. (Ver 3.3.3.4 Data Mapper)

4.2.5.2. Service Agents -5-

Los Service Agents (Agentes de Servicios) son las clases encargadas del manejo de datos que son originados en servicios externos como pueden ser WebServices u otras aplicaciones de las que se recolecta los datos necesarios para el funcionamiento de la aplicación en cuestión. Los service Agents cumplen las mismas funciones que los DALCs, definiendo operaciones que se pueden realizar sobre los orígenes de datos, y realizando la transformación de los datos entregados por los servicios externos a objetos de la aplicación que puedan ser manejados por la aplicación que se esta implementando. La utilización de DALCs o de Service Agents para una capa superior debe ser completamente transparente e incluso debe poder ser modificable o intercambiable sin afectar de ninguna manera a capas superiores.

4.2.5.3. Data storage connector -3-

Cada una de las fuentes de datos en el mercado ofrece diferentes formas de acceso a sus recursos ya sea con drivers específicos para cada uno de los motores o con métodos de acceso comunes como ODBC. Dependiendo del método de acceso elegido se debe realizar una implementación en la que se utilicen los objetos de acceso a datos correspondientes para ese método o para un motor en particular, por ejemplo, si se utiliza el driver de .net para Sql Server se deberán utilizar los objetos contenidos en el namespace

System.Data.SqlClient, en el caso de que se utilice ODBC se deberán utilizar los objetos de el namespace System.Data.Odbc.

El cambio o migración entre distintos orígenes de datos al igual que entre motores de Datos y Servicios debe ser lo más transparente posible de manera que al momento de cambiar la base de datos se deba afectar el código existente lo menos posible. Si se tiene solamente un bloque dentro de la capa de acceso a datos en el que se utilicen objetos dependientes del origen de datos usado se podría cambiar este bloque sin afectar al resto de la aplicación incluso con un impacto mínimo dentro de la misma capa de acceso a datos. Este es el propósito de Data Storage Connector. Esta clase o conjunto de clases será la única en la que se haga una referencia directa a objetos dependientes del driver en uso actualmente, de manera que el esfuerzo para cambiar de origen de datos sea mínimo.

Relación con patrones de software

El Data Storage Connector es parte del componente o patrón Data Mapper. (Ver 3.3.3.4 Data Mapper)

4.2.6. Mensajes -7-

A lo largo de toda la aplicación los distintos componentes en distintas capas deben interactuar entre ellos intercambiando datos. En una operación entre dos componentes siempre se tiene una petición (request) que define los datos de entrada del proceso, los parámetros de cómo esos datos deben ser procesados o simplemente lo que se debe realizar; también se necesita una respuesta

(response) en la que se entrega a quien envió la petición los datos resultantes de la operación realizada.

Los datos necesarios para la realización de una operación son encapsulados dentro de un objeto en la aplicación para que este objeto sea el responsable de transportar los datos entre las capas de la aplicación. Estos objetos son llamados mensajes. Independientemente de la forma en la que se alcance la implementación de una acción de negocio, este objeto tendrá los datos necesarios para realizar el procesamiento respectivo. El procesamiento interno del mensaje es mucho más simple ya que entre capas se maneja un solo objeto.

Relación con patrones de software

Los mensajes son la aplicación del patrón Data Transfer Object (Ver. 3.3.5.2 Data Transfer Object)

4.2.7. Business layer

Business Layer, es la capa que se podría llamar el “corazón” de una aplicación. En esta capa se define las reglas del negocio y la forma en la que se exponen estas reglas a la aplicación cliente. Esta capa es la encargada de realizar el procesamiento de los datos de la aplicación decidiendo las acciones que se deben tomar con los datos.

4.2.7.1. Business actions -8-

Los Business Actions son las clases en las que se realiza la implementación de las reglas de negocio, aquí se encuentra codificada la lógica necesaria para

asegurar el cumplimiento de los requerimientos de la aplicación. Los datos recibidos son procesados según las reglas de negocio, realizando las operaciones necesarias con los orígenes de datos para que estos datos sean almacenados de manera correcta luego de su procesamiento.

Estas clases son completamente independientes de la implementación real del acceso a datos y de la implementación de la presentación a los usuarios, son clases en las que el código presente es código de procesamiento de información y no debe de ninguna manera estar involucrado ningún tipo de código de presentación o de acceso a datos. Estas clases son las responsables del procesamiento de los mensajes enviados por la aplicación y de la generación del mensaje de respuesta. Toda la comunicación hacia las capas superiores se realiza a través de mensajes.

Relación con patrones de software

Los Business Actions corresponden a la forma de organización de la lógica de negocio que se relaciona con el patrón Domain Model (Ver 3.3.2.2 Domain model)

4.2.7.2. EDAF -9-

EDAF o Enterprise Development Application Framework es un framework de desarrollo de aplicaciones distribuidas empresariales que forma parte de EDRA (Enterprise Development Reference Architecture). Es una de las posibles

implementaciones de la Arquitectura conceptual planteada en la documentación de EDRA[165].

EDAF como parte de EDRA, propone un Framework sólido de desarrollo de aplicaciones distribuidas empresariales que toma en cuenta consideraciones comunes para el desarrollo de aplicaciones distribuidas como son concurrencia, transaccionalidad, transporte, implementación física, entre otros. EDAF provee al desarrollador un punto de inicio para el desarrollo de un Framework sólido y confiable.

La interacción entre una aplicación cliente y un servicio o un Business Action pueden ser simplificados como una petición y una respuesta. Hay otras consideraciones que deben ser tomadas en cuenta al momento de realizar el envío de una petición a un servicio como por ejemplo:

- ¿Cómo se entregará el mensaje?
- ¿Cómo debe ser enviada la petición entre procesos o entre servidores?
- ¿Cómo será el manejo de errores?
- ¿Se soportarán transacciones?

Consideraciones o preguntas como estas son solamente algunas de varias que complican esta interacción entre la aplicación cliente y el Business Action. EDAF conceptualmente se encuentra localizada entre la aplicación cliente y el

[165] Microsoft Corporation, Patterns & Practices: Enterprise Development Reference Architecture

Business Action, de manera que EDAF es quien se encarga de manejar la forma en la que se conectan las aplicaciones cliente con los servicios.

Dentro de EDAF se encuentran dos capas: La capa de Service Interface y Service Implementation. Mientras service Interface se encarga del enlace con la aplicación cliente, filtrando las peticiones, la capa de Service Implementation se encarga de realizar la llamada al servicio solicitado.

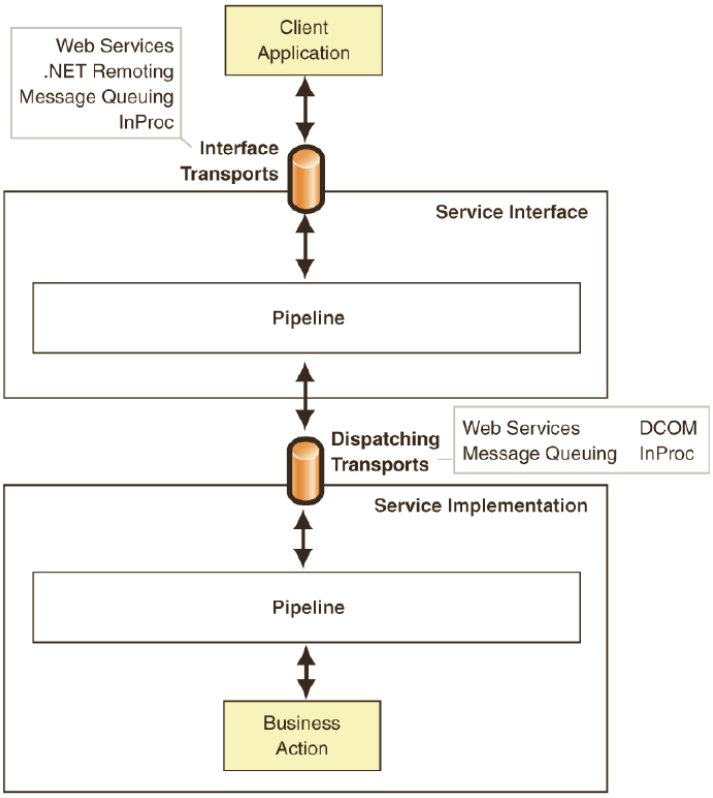


Figura 4-2: Componentes de EDAF[166]

[166] Microsoft Patterns and Practices, EDRA Powerpoints

El proceso de llamada a un servicio es el siguiente:

- Una aplicación cliente envía una petición para un servicio utilizando uno de los transportes disponibles, como pueden ser Web Services, Message Queue, InProc, o un Componente de Remoting.
- El transporte recibe la petición y la pasa a una instancia de la línea de procesamiento de Service Interface Pipeline.
- La línea de proceso aplica los handlers necesarios a la petición entrante. Y la petición es pasada a los Dispatching Transports.
- Dispatching Transports toma la petición, y la pasa a la línea de Procesamiento de Service Implementation Pipeline.
- Service implementation Pipeline, aplica los handlers necesarios, e invoca a el Business Action.
- Business Action genera la respuesta a la petición y lo pasa hacia arriba en el mismo orden hasta llegar a la aplicación cliente que originó la petición.

EDAF es la capa encargada del manejo de las comunicaciones en la arquitectura propuesta. La arquitectura e implementación de EDAF brinda un marco de trabajo de comunicaciones sobre el que es posible agregar los demás bloques de la aplicación, manejando los mensajes de una manera consistente e independientemente del transporte utilizado para que el mensaje llegue a su destino permitiendo exponer los servicios en distintos transportes.

Las capas superiores de la aplicación se conectan a EDAF por medio de las interfaces de servicio expuestas por diferentes transportes en la capa de Service Interface, y por medio de los Business Actions en la capa de Service Implementation. Estos son los puntos sobre los cuales la aplicación es ampliada e implementada dependiendo de los requerimientos específicos de la misma.

4.2.7.3. Interface transports -10-

Los Interface transports representan el borde de la conexión con EDAF, los cuales reciben las peticiones enviadas por capas superiores para pasarlos a EDAF y comenzar su procesamiento hasta llegar al Business Action y éstos retornan una respuesta a esa petición. Los interface transports se podrían tomar como los servicios que expone la capa de negocio a las capas superiores.

Para cada Business Actions se tendrá uno o más Interface Transports, que corresponden a los transportes por los cuales es posible realizar la llamada a ese Busines Action. Por ejemplo si un Business Action puede ser llamado por medio de Remoting, MSMQ y Web Service, se tendrá tres Interface Transports, cada uno de ellos correspondiente a uno de los transportes mencionados.

Relación con patrones de software

Los Interfaces Transports son la aplicación del patrón Service Layer. (Ver 3.3.2.4 Service Layer)

4.2.8. UI Process Layer

La capa de Process UI es responsable del manejo de la comunicación entre la interface de usuario y la capa de negocios. Muchas veces no es posible el paso directo de datos entre la vista de los datos de un usuario y un componente de proceso de negocio y es necesario realizar un proceso de transformación de los datos, estableciendo canales de comunicación con las capas inferiores para lograr esta tarea. Si todo este código estuviera directamente en la Interface de Usuario, resultaría un código difícil de mantener, poco portable y difícil de reutilizar.

Usualmente mientras el usuario va realizando operaciones dentro de la aplicación se generan datos de estado que deben mantenerse, ya sea para futuras sesiones del mismo usuario, o para utilizarlos en la misma sesión en una actividad futura, estos datos generados por la interacción del usuario con la aplicación son denominados estados, normalmente este estado debe ser transferido entre los distintos componentes de la interface como por ejemplo entre las vistas de la aplicación.

El manejo de los estados de la aplicación si no es manejado de manera correcta puede resultar en código poco extensible, difícil de mantener y soluciones poco elegantes a este problema, dando como resultado además componentes de interfaz de usuario que son difíciles de modificar y mantener sin que cause problemas en otros componentes relacionados incluso en caso de que se desee modificar el orden de las vistas, o agregar pasos intermedios por la dependencia de las vistas entre ellas la tarea se tornará muy complicada.

La capa de Process UI (Proceso de Interfaz de Usuario) es la encargada de proporcionar una solución extensible y elegante al problema del manejo de estados y dependencia de los componentes de la interfaz de usuario. El propósito de esta capa es proveer una forma extensible y genérica de manejo de estados de una aplicación, interacción y el flujo de los componentes de usuario y a la vez permitir que el código contenido en los componentes de la interfaz sea lo más simple y reducido posible, descargando de estos componentes todo código referente a las tareas antes mencionadas.

4.2.8.1. Transport proxies -11-

Los transport proxies son clases encargadas de la creación, mantenimiento y manejo de canales de comunicación, que son establecidos con los Transport Interfaces. Estos proxies de transporte son encargados de verificar la existencia de un transporte específico para un Business Action, verificar la prioridad en la que los diferentes transportes pueden ser utilizados y realizar las llamadas correspondientes para la que se ejecuten los procesos de negocio.

Muchas veces un servidor no se encuentra disponible por varias razones, pero no se puede parar la aplicación, entonces es necesario el levantar un canal auxiliar de comunicaciones con otro servidor para continuar el proceso, o en otros casos dependiendo del ambiente en el que se desea implementar la aplicación un método de comunicación puede ser preferido sobre otro. Por ejemplo en una LAN posiblemente remoting sea más adecuado, pero en ambientes en Internet quizá

no sea posible la utilización de ese transporte y se deba utilizar Web Services para la comunicación.

Si toda la codificación para el manejo de los transportes, servidores y servicios de una aplicación sería realizada de una forma manual, debería escribirse este código por cada uno de los Business Actions de la aplicación, peor aún si esta codificación es realizada directamente en los componentes de la interface de usuario resultaría en código poco organizado y difícil de mantener, además de interfaces de usuario poco reutilizables.

Es por estas razones que se vuelve necesario un componente en el cual se encapsule toda la lógica de manejo de los transportes que una aplicación contiene y es necesario que sea realizado de una forma lo más transparente posible para los componentes que deben utilizar estos transportes sean lo más simples posible. Además es necesario que este componente sea completamente configurable y reutilizable. Los Transport Proxies separan toda la lógica necesaria para administración de los transportes y proporcionan la extensibilidad necesaria al ser configurados mediante un archivo de configuración en texto plano que puede ser cambiado de una manera simple.

Relación con patrones de software

Los Transport proxies implementan los siguientes patrones de software:

- Identity Map(Ver 3.3.3.6 Identity Map) Factory method
- Factory Method (Ver 3.2.2.1.1 Factory method)
- Singleton (Ver 3.2.2.1.5 Singleton)

- Bridge (Ver 3.2.2.2.2 Bridge)
- Virtual Proxy (Ver 3.2.2.2.7 Proxy)

4.2.8.2. Operaciones -12-

Las operaciones son las responsables de la generación de los mensajes que se envían a los Business Actions. Estas clases generan el encapsulamiento para generar o procesar los objetos mensajes que son enviados o son respuestas de las llamadas a los Business Actions. Las operaciones dependen directamente de los proxies de transporte ya que por medio de ellos realizan la comunicación.

Las operaciones son los puntos en los que se expone la funcionalidad de los Business Actions para la aplicación cliente, ya que es aquí donde se definen las operaciones que luego serán enviadas a los procesos de negocio para ser ejecutadas. Estos componentes son los únicos en los que se mantiene una referencia directa a los proxies de transporte y los únicos que manejan los objetos de mensajes.

Relación con patrones de software

Los UI Process Components implementan los siguientes patrones de software:

- Facade (Ver 3.4.4.1.5 Facade) Factory method
- Adapter (Ver 3.2.2.2.1 Adapter)
- Domain Model (Ver 3.3.2.2 Domain model)

4.2.8.3. UI process components -13-

Las aplicaciones usualmente contienen código para manejar la interacción de los componentes en la capa de aplicación, por ejemplo un formulario puede determinar el siguiente formulario al que el usuario debe ir. Este código comúnmente se encuentra escrito dentro de la interfaz como tal. Sin embargo cuando esto sucede, la lógica de flujo de la aplicación, y el estado que comparten los formularios no puede ser reutilizado.

Mientras un usuario interactúa con la aplicación, se pueden iniciar tareas, ponerlas en espera, y luego retornar a ellas. Si el usuario cierra la aplicación entre sesiones, se pueden perder datos y puede ser necesario el empezar la tarea desde el principio, esto es tedioso para el usuario y en otros casos no es aceptable para la lógica de negocio. Es por esto que mientras se diseña la aplicación se debe considerar el flujo de la misma, la navegación, y la interacción con los Business Actions y componentes como temas separados de cómo los datos son adquiridos y presentados a los usuarios.

Microsoft nos provee de un bloque o capa denominado UIP (User Interface Process) que implementa las principales características del modelo vista controlador y del controlador de aplicación. Estas características incluyen:

- Controladores de aplicación
- Control del flujo de navegación de interfaces de usuario
- Manejo de estado persistente de sesión del usuario

- Resumir una sesión Web desde cualquier dispositivo o desde cualquier ubicación
- Reutilización de código entre tipos de aplicaciones (Aplicaciones Windows o Web)
- Creación de tareas ordenadas que implementan un flujo o proceso de negocio complejo.

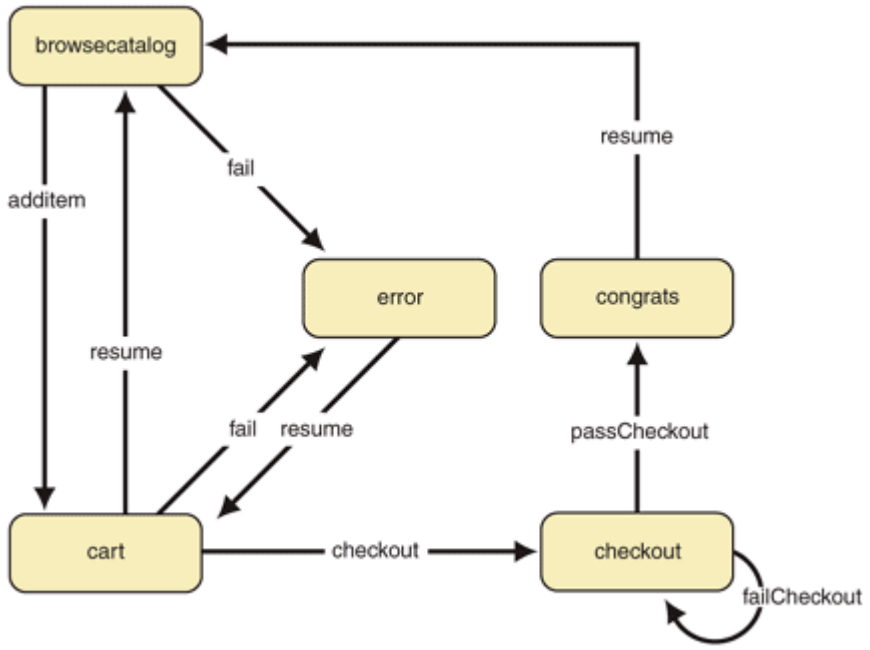


Figura 4-3: Ejemplo de un flujo de navegación de interfases de usuario[167]

4.2.8.3.1. Controllers -14-

Una parte central del bloque de UIP (User Interface Process) son los controladores. Estos son el equivalente a la porción del controlador del modelo de MVC (Ver 3.3.4.1 Modelo Vista Controlador), y su propósito principal es el de controlar la navegación entre las vistas y el de actual como una fachada entre la

[167] Microsoft Patterns and Practices, User Interface Process Application Block Help

capa de presentación y la capa de negocios de la aplicación. Desde el controlador se tiene acceso al estado de una tarea particular, y se tiene la habilidad de controlar la navegación por las distintas vistas de la interface de usuario.

4.2.8.3.2. Tasks -15-

Un Task es una instancia que se encuentra en ejecución de un proceso de un usuario; por ejemplo un usuario puede iniciar una tarea para la creación de un nuevo producto. Esta tarea entonces estará presente y su estado encapsulará todos los datos generados por este proceso mientras el usuario esté ejecutando este proceso. Cada tarea tendrá entonces un identificador único que enlazará el proceso del usuario con los datos del estado del mismo.

Relación con patrones de software

Los UI Process Components implementan los siguientes patrones de software:

- Facade (Ver 3.4.4.1.5 Facade) Factory method
- Modelo Vista Controlador (Ver 3.3.4.1 Modelo Vista Controlador)
- Application controller (Ver 3.3.4.6 Application controller)
- Database Session State (Ver 3.3.6.3 Database Session State)
- Remote Proxy (Ver 3.2.2.2.7 Proxy)
- Bridge (Ver 3.2.2.2.2 Bridge)

4.2.9. Capa de presentación

La capa de presentación es donde se encuentran todos los componentes visuales que son presentados al usuario. Tradicionalmente existen dos formas de presentación al usuario: Windows y Web.

Esta capa se encarga de procesar las entradas de los usuarios y convertirlas en datos que serán procesados dentro de las capas inferiores de la aplicación. Una capa de presentación también se encarga de cómo mostrar al usuario los datos que este requiera. Por ejemplo, si los datos de una tabla se deben presentar en forma de un grid o en forma de una lista.

4.2.9.1. Windows Forms -16-

Esta forma de presentación de una aplicación obedece a las aplicaciones de escritorio o ventanas que son instaladas y ejecutadas localmente en un PC. Una aplicación de éste tipo necesita estar instalada en un sistema operativo por lo general Windows.

4.2.9.2. Web Forms -17-

Esta forma de presentación de aplicaciones basada en Web es muy versátil y muy común en la actualidad. No es necesario instalar ningún tipo de software en la parte del cliente ya que solo basta abrir un explorador de Internet y se tiene acceso a la aplicación.

Relación con patrones de software

Los Web Forms son la aplicación de los patrones de software:

- Modelo Vista Controlador (Ver 3.3.4.1Modelo Vista Controlador)
- Client Session State (Ver 3.3.6.1 Client Session State)
- Server Session State (Ver 3.3.6.2 Server Session State)
- Page Controller (Ver 3.3.4.2 Page Controller)

CAPITULO V

5. DEFINICIÓN E IMPLEMENTACIÓN DE LA APLICACIÓN PROTOTIPO USANDO EL FRAMEWORK DEFINIDO

Este capítulo pretende mostrar los pasos para la implementación de una aplicación empresarial utilizando el framework definido en el capitulo anterior, mediante una guía que ejemplifica cómo construir una aplicación denominada “prototipo” al mismo tiempo que muestra los pasos de cómo se debería implementar una aplicación nueva.

5.1. Definición de la aplicación prototipo

5.1.1. Descripción general

La aplicación prototipo representa la implementación práctica del framework definido denominado “Millwood”, esta aplicación pretende mostrar de forma real el uso y aplicación del framework mediante la implementación de un sitio virtual de compras.

El sitio virtual de compras estará compuesto de una aplicación Web en la cual, se desplegará un catálogo de productos, el visitante podrá añadir uno o más productos al “carrito de compras” y finalizará el pedido. Adicionalmente existirá

una aplicación de escritorio la cual realizará la administración del inventario del sitio de compras.

5.1.2. Alcance

La funcionalidad definida en la especificación de requerimientos será implementada mediante la construcción de una aplicación que estará basada completamente en el framework definido. Cada parte dentro de la arquitectura distribuida de la aplicación utilizará los patrones de software que brinda el framework, mostrando la forma de implementación de cada una de las capas necesarias desde las inferiores (acceso a datos) hasta las superiores (presentación).

5.1.3. Análisis y Diseño

5.1.3.1. Especificación de requerimientos funcionales

La aplicación prototipo estará compuesta por dos aplicaciones, una aplicación Web donde se implementará el sitio virtual de compras y una aplicación de escritorio en donde se podrá realizar la administración del catálogo de productos.

5.1.3.1.1. Especificación de Requerimientos: Aplicación Web

La aplicación Web representa el sitio virtual de compras en línea, en el cual un usuario visitante se identifica en el sitio y consulta el catálogo de productos para realizar un pedido. Los requerimientos específicos son los siguientes:

1. Identificar al usuario visitante mediante una pantalla de inicio de sesión.
2. Presentar al usuario registrado un catálogo de productos filtrado por categoría. Un producto pertenece solo a una categoría.
3. El usuario agrega al “carrito de compras” uno o varios productos pudiendo seleccionar la cantidad de cada uno.
4. El usuario puede consultar en cualquier momento los elementos añadidos al carrito de compras y volver al catálogo de compras y seguir comprando.
5. Los productos añadidos al carrito de compras pueden ser modificados, ya sea cambiando la cantidad o eliminándolos.
6. Una vez finalizado el proceso de selección de productos el usuario se dirige a la página de pago. En esta pantalla se ingresa la tarjeta de crédito y una observación opcional.
7. El usuario recibe una página de confirmación con los productos que ha comprado y se confirma la orden.
8. El usuario puede en cualquier momento interrumpir el proceso de compra y continuarlo posteriormente, sin perder ninguna información y regresando al mismo paso en donde se interrumpió el proceso.

5.1.3.1.2. Especificación de Requerimientos: Aplicación Windows

La aplicación Windows permitirá la administración de los productos que se venden en el sitio virtual de compras mediante la aplicación Web. Los requerimientos específicos son los siguientes:

9. Un usuario debe identificarse como administrador del sistema a través de un proceso de inicio de sesión.
10. Una vez identificado el usuario como administrador, se despliega una pantalla de administración de productos. La pantalla puede realizar las siguientes tareas:
10.1 Consultar el catálogo de productos filtrados por categoría.
10.2 Se puede agregar, editar o eliminar productos
10.3 Se puede agregar, editar o eliminar categorías de productos.
10.4 Un producto es asignado a una categoría de producto.

5.1.3.2. Modelo Funcional

5.1.3.2.1. Casos de uso

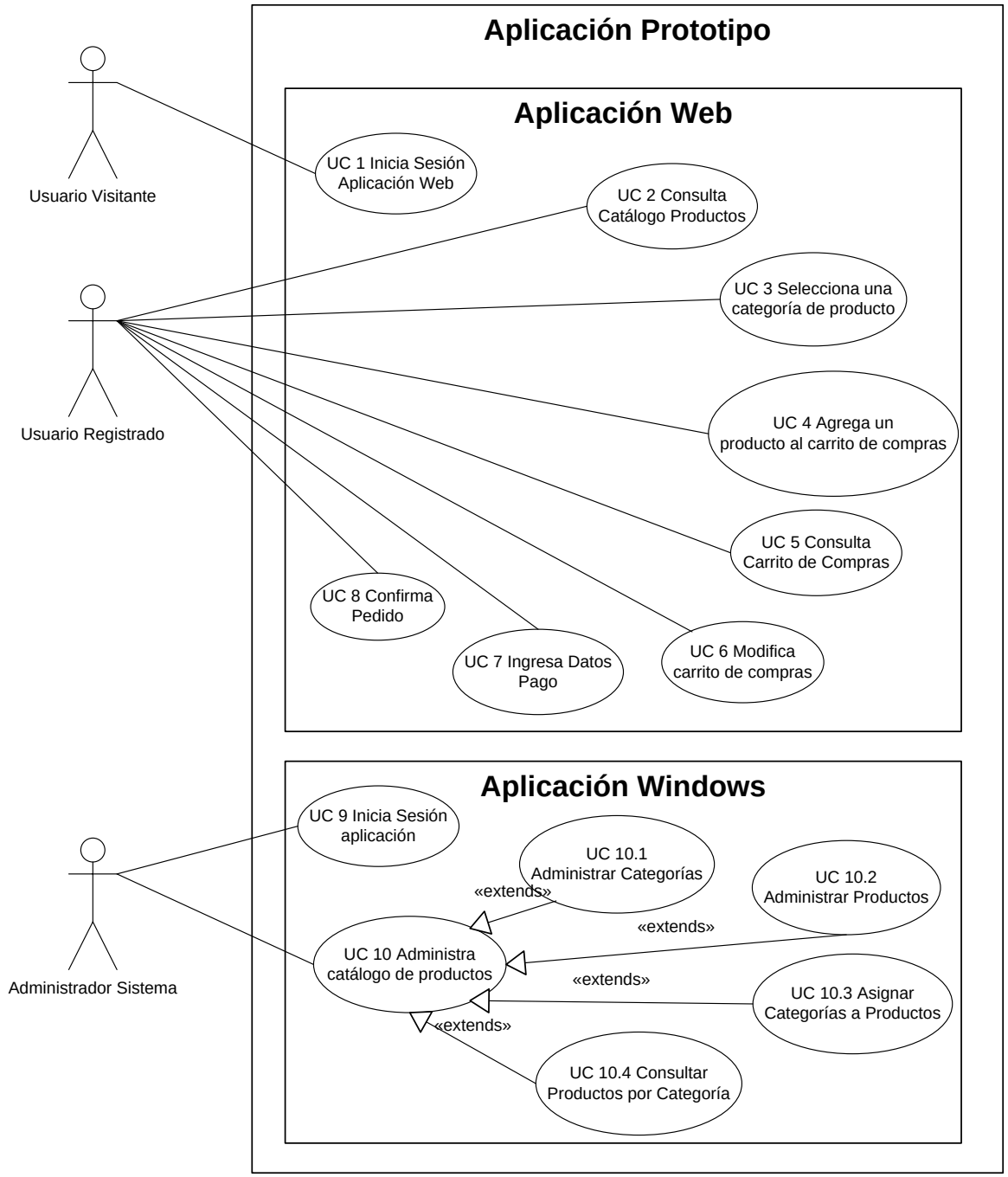


Figura 5-1: Casos de uso de la aplicación prototipo[168]

[168] Elaboración Propia

5.1.3.2.2. Descripción de Casos de Uso

La descripción de casos de uso se encuentra en el Anexo B: Descripción de Casos de Uso.

5.1.3.3. Modelo de Objetos

5.1.3.3.1. Diagrama de Clases

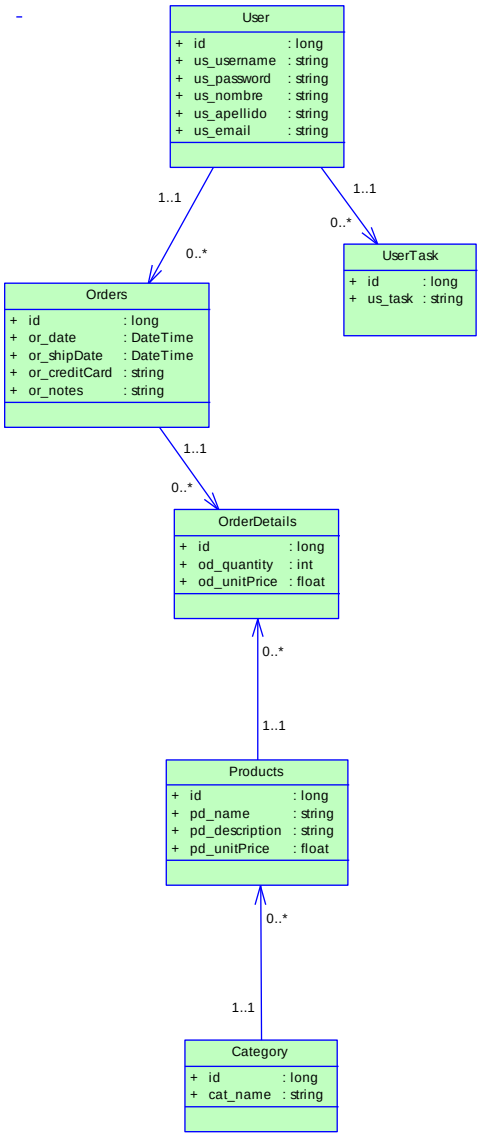


Figura 5-2: Diagrama de clases de la aplicación prototipo[169]

[169] Elaboración Propia

5.1.3.4. Modelo Dinámico

5.1.3.4.1. Diagrama de Secuencia

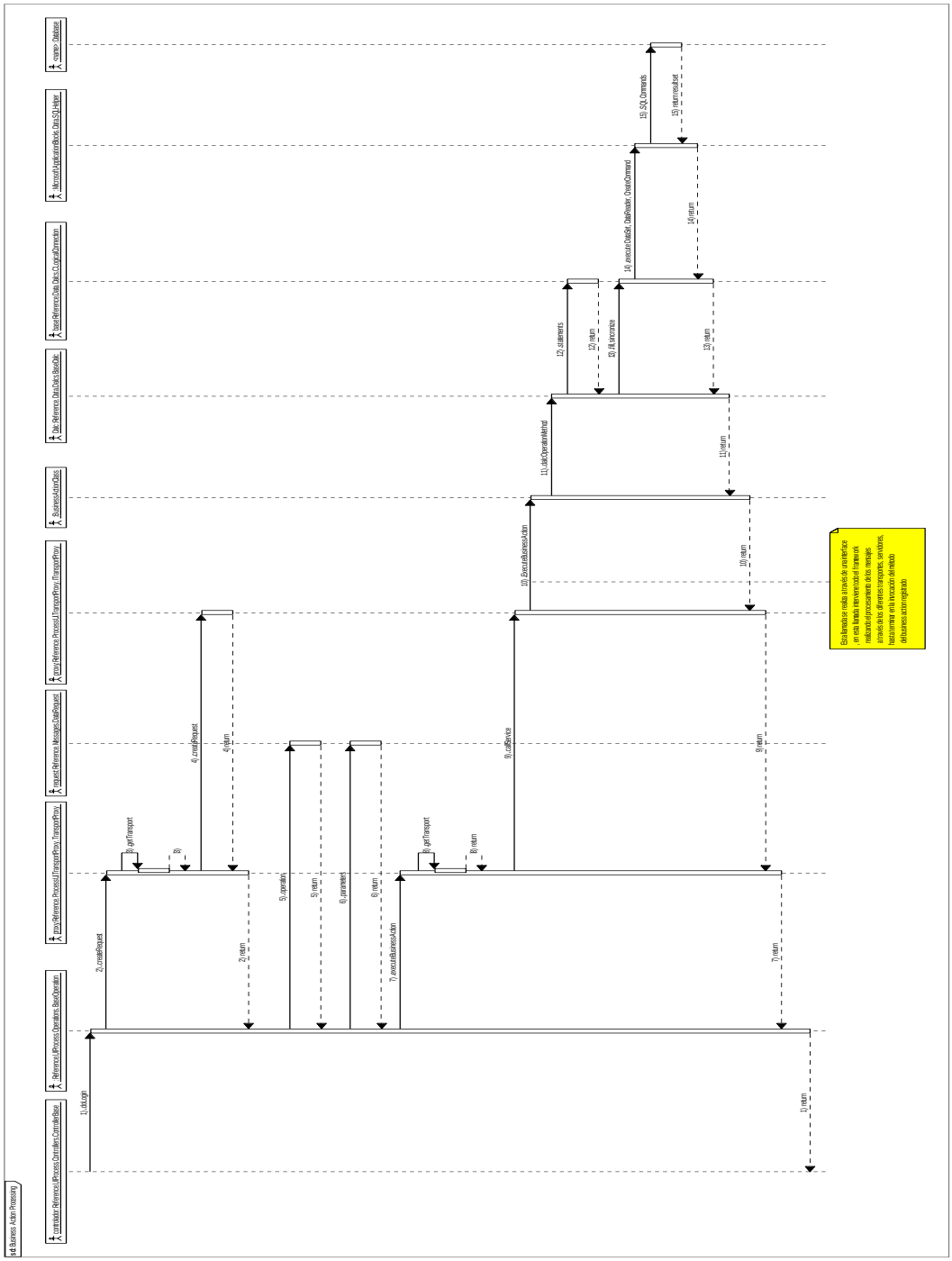


Figura 5-3: Diagrama de secuencia de la aplicación prototipo[170]

[170] Elaboración Propia

5.1.3.5. Modelo de Implementación

5.1.3.5.1. Diagrama de Componentes

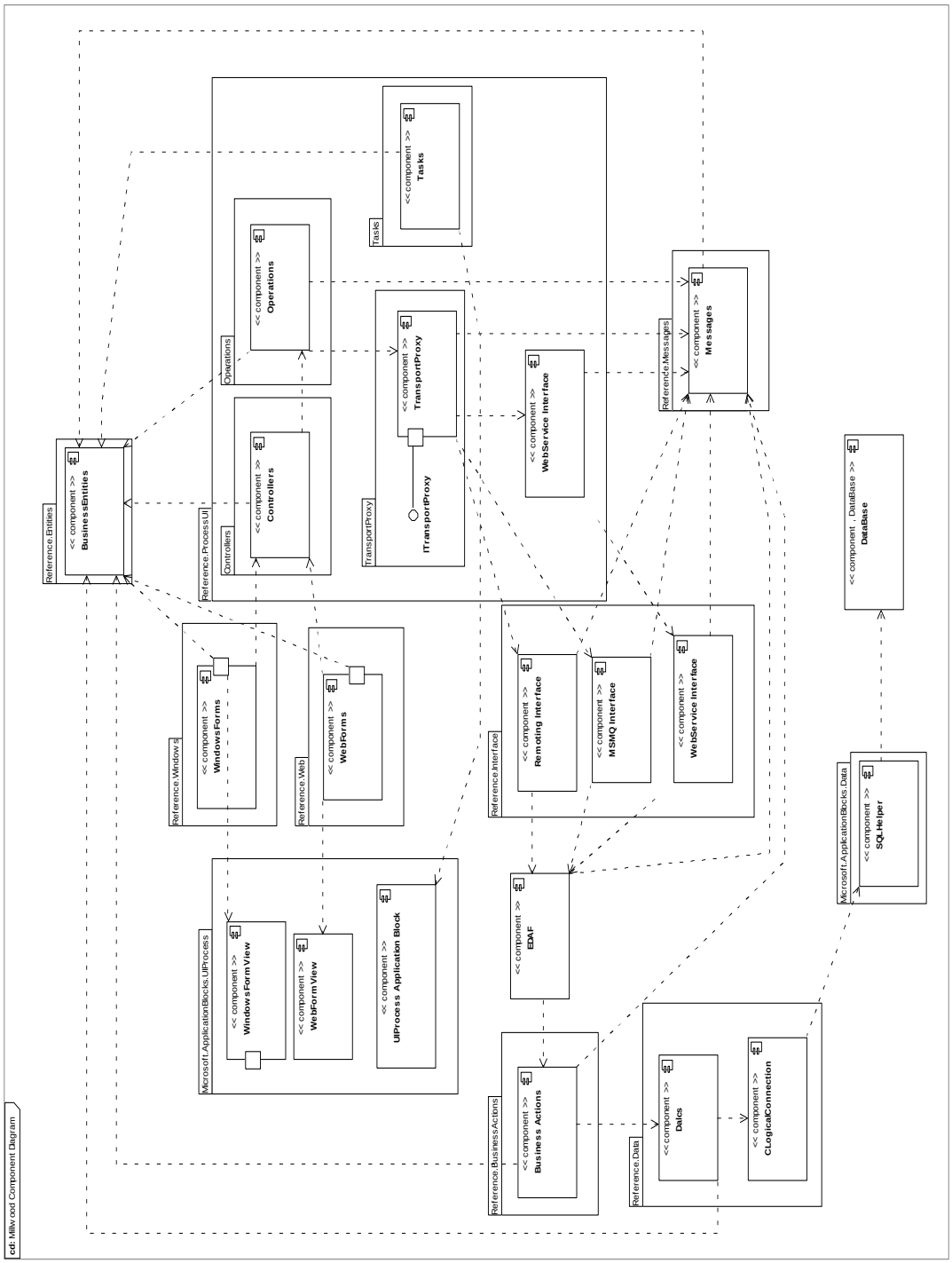


Figura 5-4: Diagrama de Componentes de la aplicación prototipo[171]

[171] Elaboración Propia

5.1.3.5.2. Modelo Conceptual

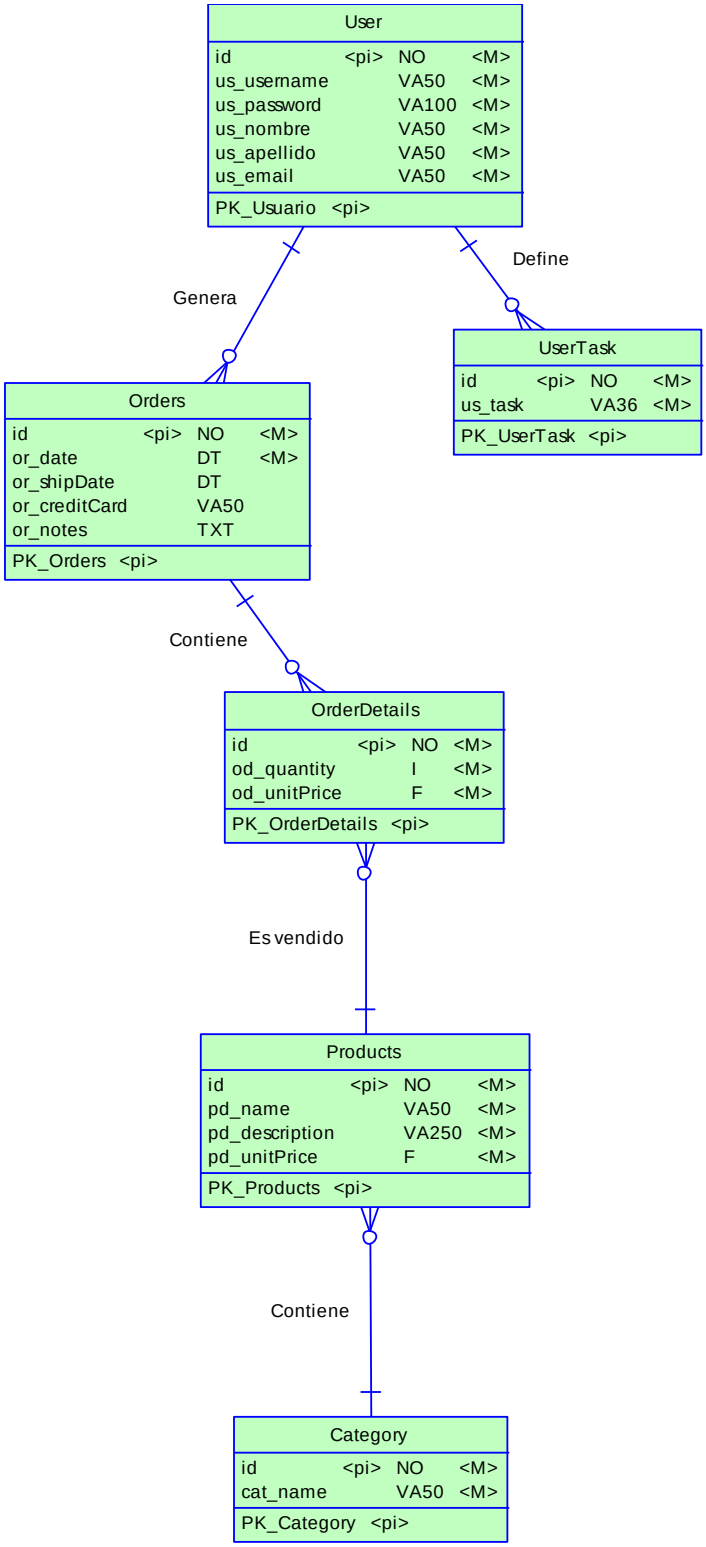


Figura 5-5: Modelo conceptual de datos de la aplicación prototipo[172]

[172] Elaboración Propia

5.1.3.5.3. Modelo Físico

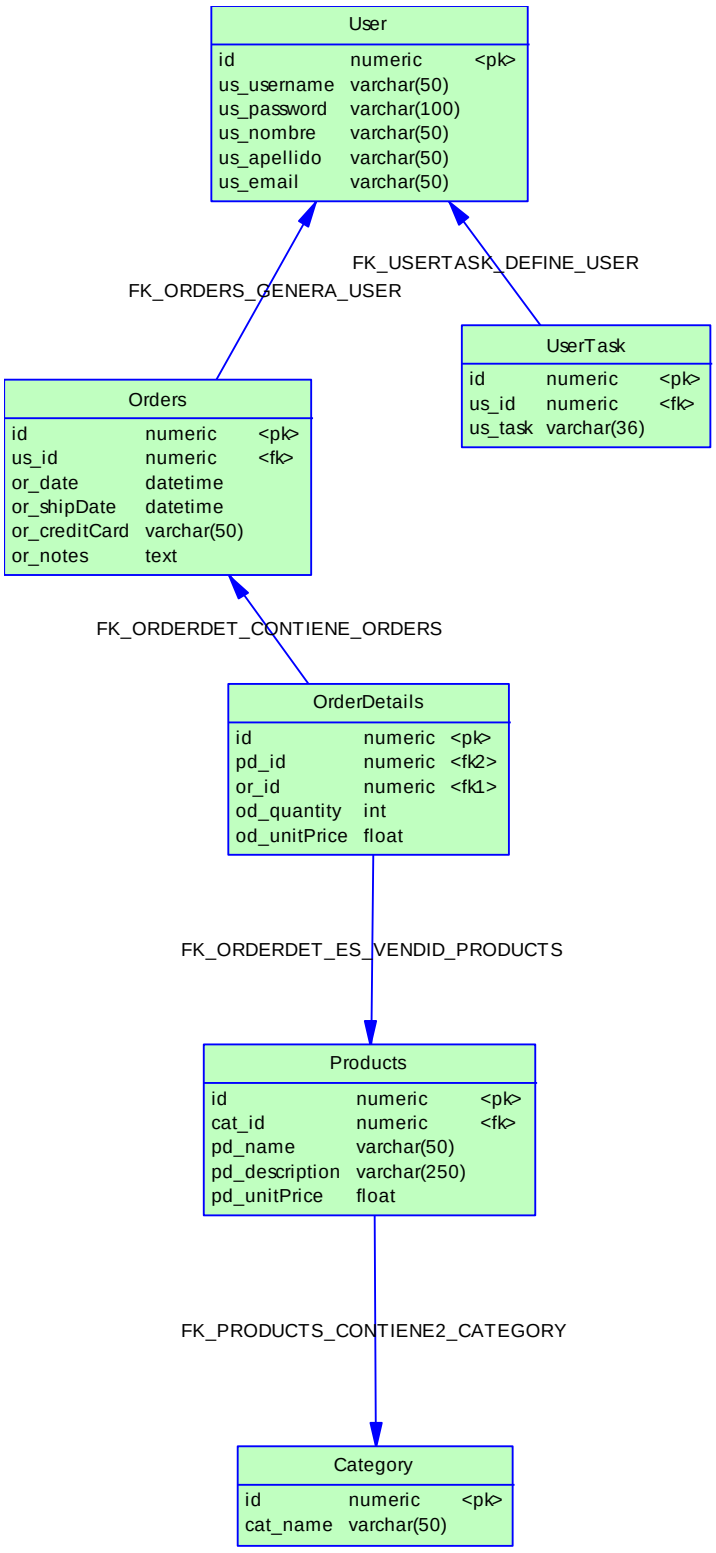


Figura 5-6: Modelo Físico de datos de la aplicación prototipo[173]

[173] Elaboración Propia

5.2. Guía de implementación de la aplicación prototipo

5.2.1. Guía de instalación de requerimientos

Para la instalación de requerimientos se asume que están instalados los siguientes componentes:

- Microsoft Windows XP o Windows Server 2003
- Microsoft SQL Server 2000 con seguridad en modo mixto. El modo mixto permite configurar ambos esquemas de seguridad (Autenticación en modo Windows y autenticación SQL Server) para comodidad del desarrollador. Sin embargo es recomendable la utilización de la seguridad integrada para mayor nivel de seguridad.
- Visual Studio .net 2003 Enterprise Architect

5.2.1.1. Instalación de componentes de Windows

Se debe instalar los servicios de Internet information Server (IIS) y Message Queue Server

- Nos dirigimos al Panel de control -> Agregar o quitar programas.
- Seleccionamos agregar o quitar componentes de Windows
- Seleccionamos “Message Queue Server” y Servicios de Internet information Server”

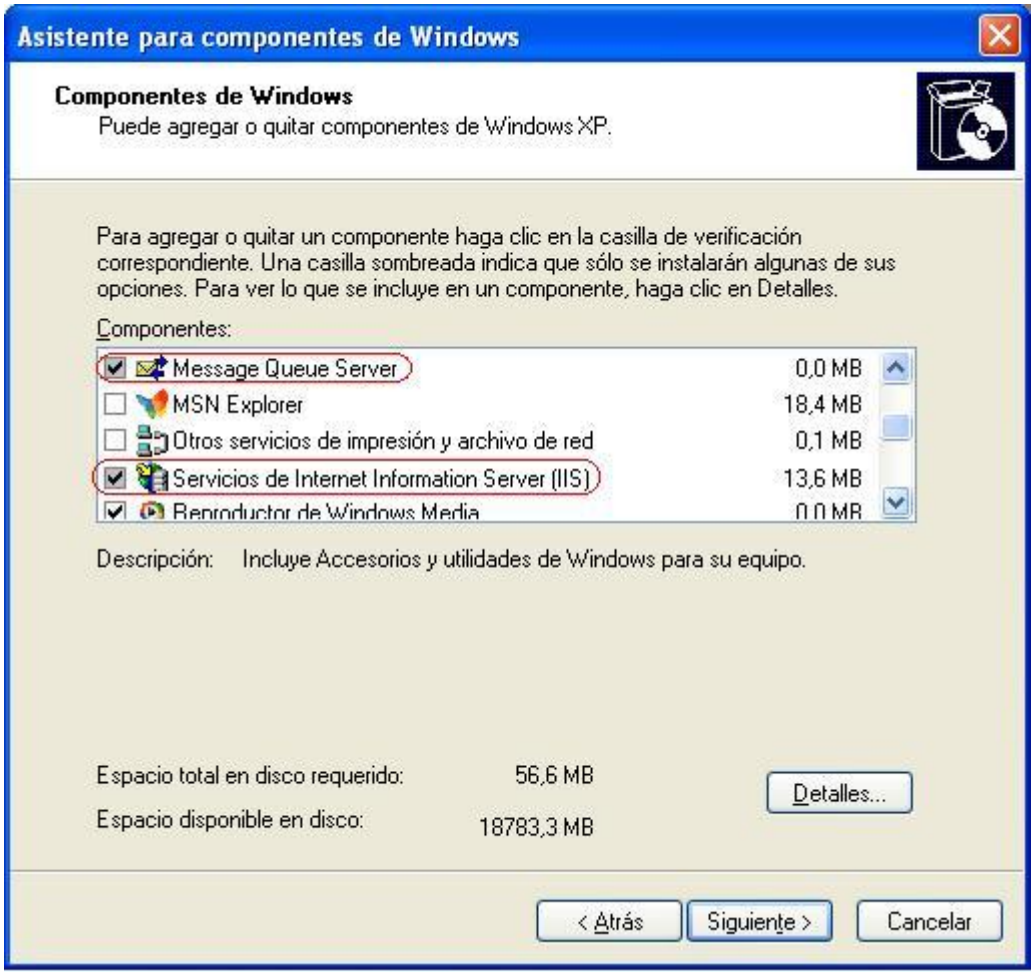


Figura 5-7: Instalación de IIS y MSMQ[174]

- Presionamos siguiente para concluir la instalación

5.2.1.2. Instalación de requisitos

Se deben instalar los componentes adicionales que requiere el framework Millwood, estos se encuentran en el CD que acompaña a éste texto guía. Los componentes son:

[174] Elaboración Propia

- Enterprise Development Reference Architecture (EDRA). Este paquete instala los componentes de EDAF necesarios para la implementación de las interfaces de servicio. (Ver 4.2.7.2 EDAF -9-)
- Microsoft enterprise instrumentation framework. Este componente de Microsoft permite implementar un sistema de registro automático de eventos que ocurren durante las fases de comunicación del framework, especialmente en la parte de EDAF.
- Microsoft EDRA Wizards. Estos wizards de EDRA ayudan a la implementación de los Business Actions en las interfaces de servicio.
- Microsoft application block: User Interface Process 2.0. Este componente de Microsoft nos ayuda en la implementación de la capa de Process UI, para el manejo de controladores, tareas, sesión y navegación. (Ver 4.2.8.3 UI process components -13-)

5.2.1.2.1. Instalación de enterprise instrumentation framework

Para instalar enterprise instrumentation framework nos dirigimos a la carpeta Requisitos y hacemos doble clic en “EnterpriseInstrumentation.exe”.

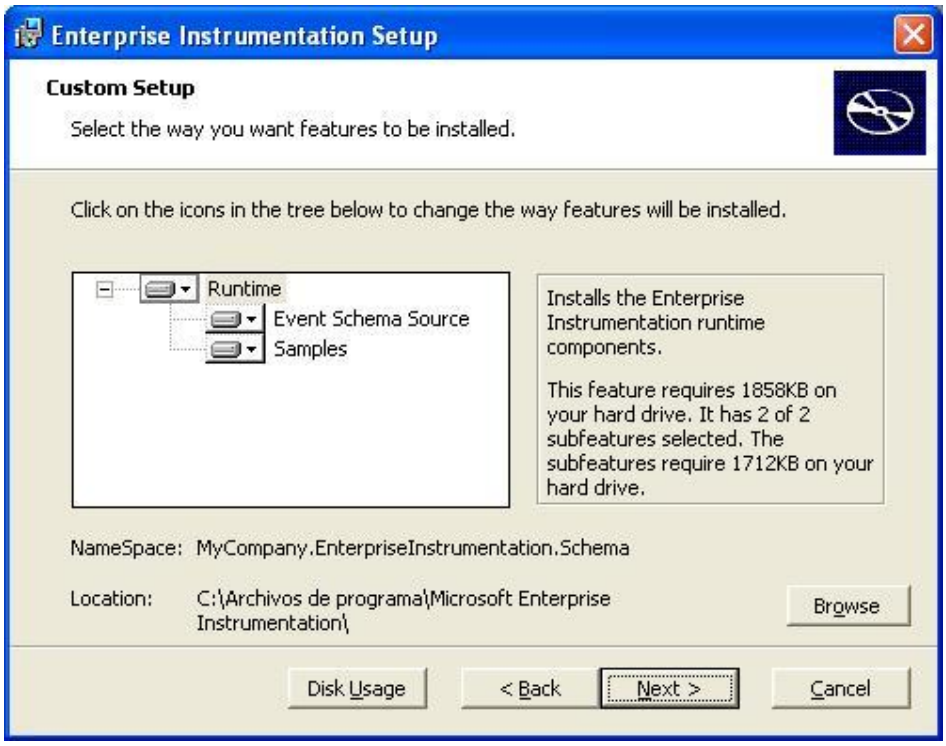


Figura 5-8: Instalación de enterprise instrumentation framework[175]

Clic en siguiente, y aceptar los términos de la licencia.

Figura 5-9: Instalación de enterprise instrumentation framework (2)[176]

[175] Elaboración Propia
[176] Elaboración Propia

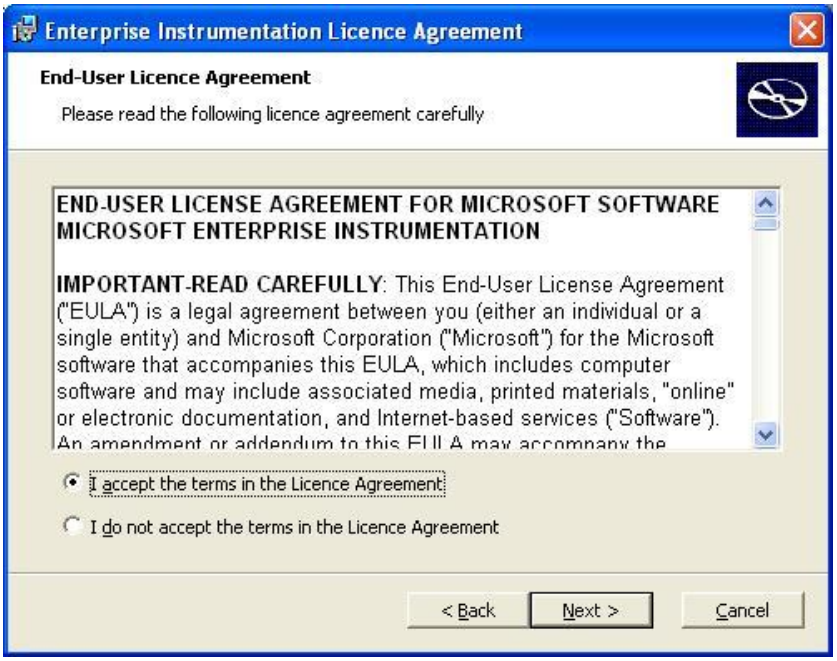


Figura 5-10: Instalación de enterprise instrumentation framework (3)[177]

Seleccionamos la carpeta de destino donde se va a instalar el componente.

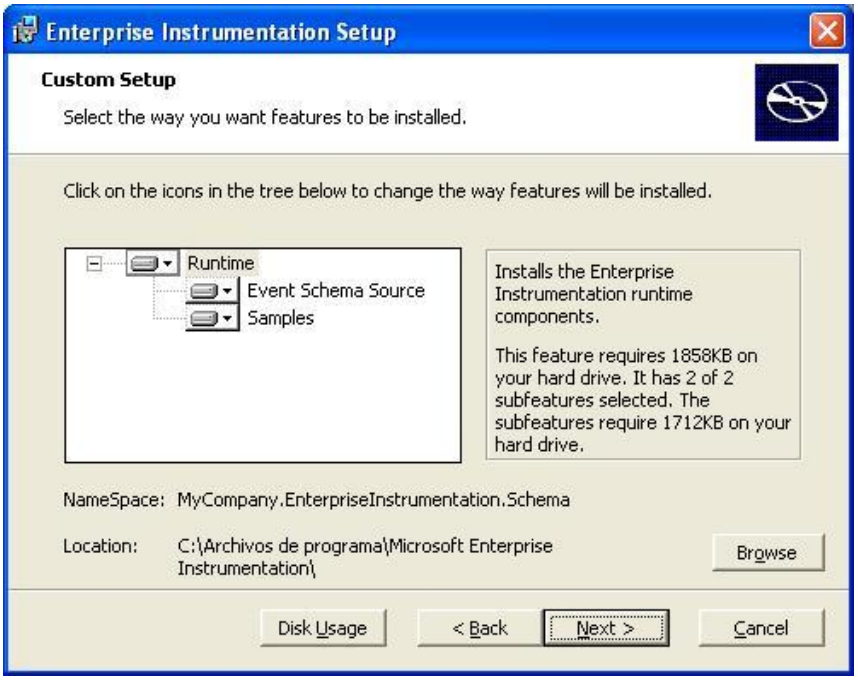


Figura 5-11: Instalación de enterprise instrumentation framework (4)[178]

Para finalizar la instalación hacemos clic en finalizar.

---

[177] Elaboración Propia
[178] Elaboración Propia

5.2.1.2.2. Instalación de Enterprise Development Reference Architecture

Para instalar Enterprise Development Reference Architecture nos dirigimos a la carpeta Requisitos y hacemos doble clic en “Enterprise Development Reference Architecture.exe”.

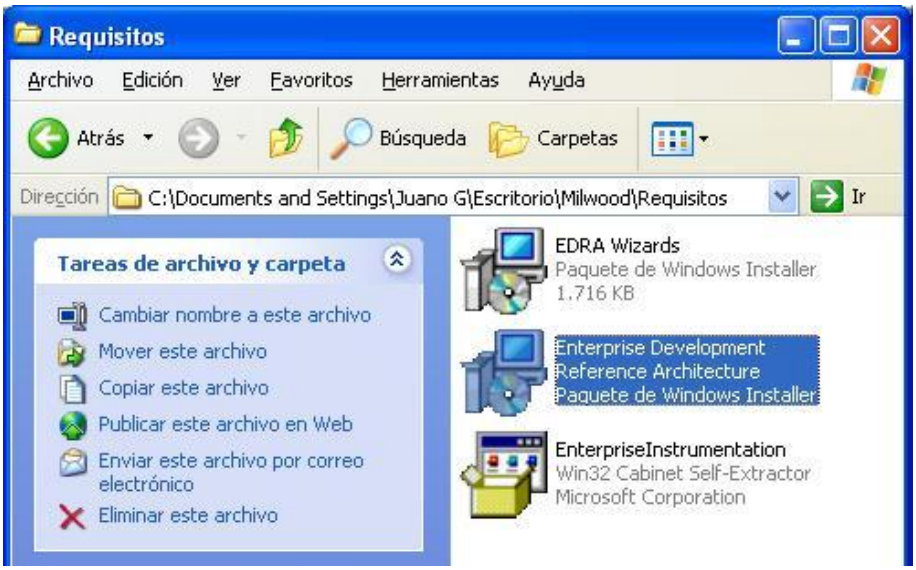


Figura 5-12: Instalación de EDRA[179]

Clic en siguiente, y aceptar los términos de la licencia.

[179] Elaboración Propia

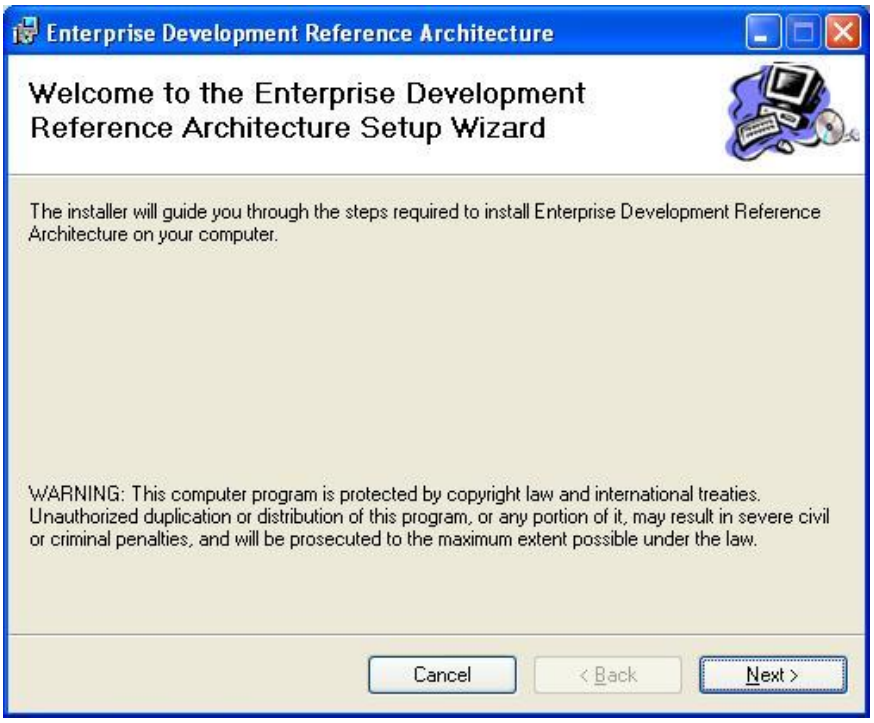


Figura 5-13: Figura 4 7: Instalación de EDRA (2)[180]

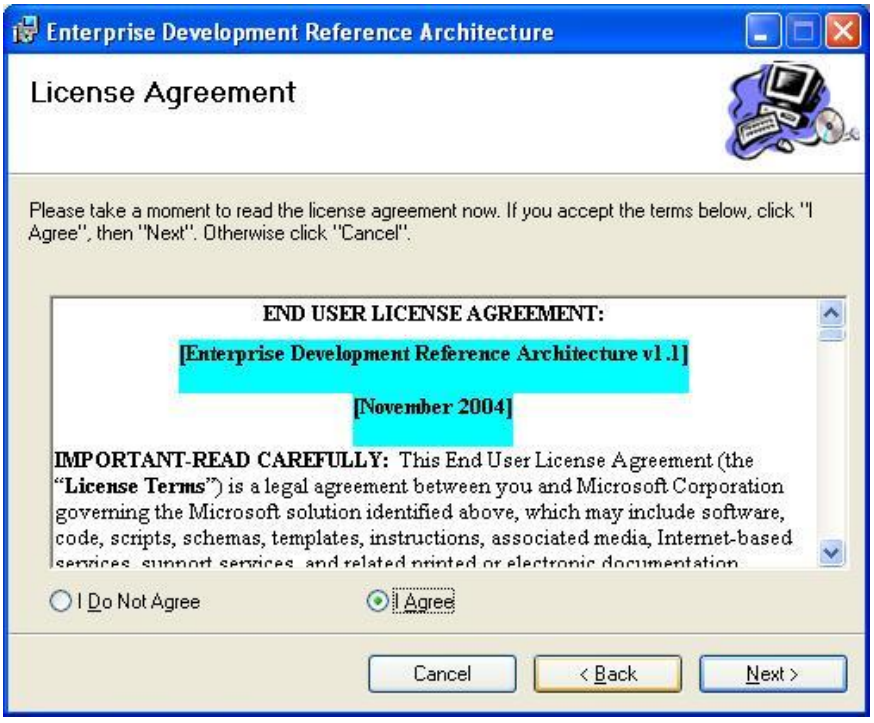


Figura 5-14: Instalación de EDRA (3)[181]

[180] Elaboración Propia
[181] Elaboración Propia

En el diálogo de selección siguiente, seleccionamos la segunda opción que permite especificar el archivo que servirá para que todos los Assemblies estén firmados con un nombre fuerte, esto es requisito para la parte de EDAF dentro del framework.

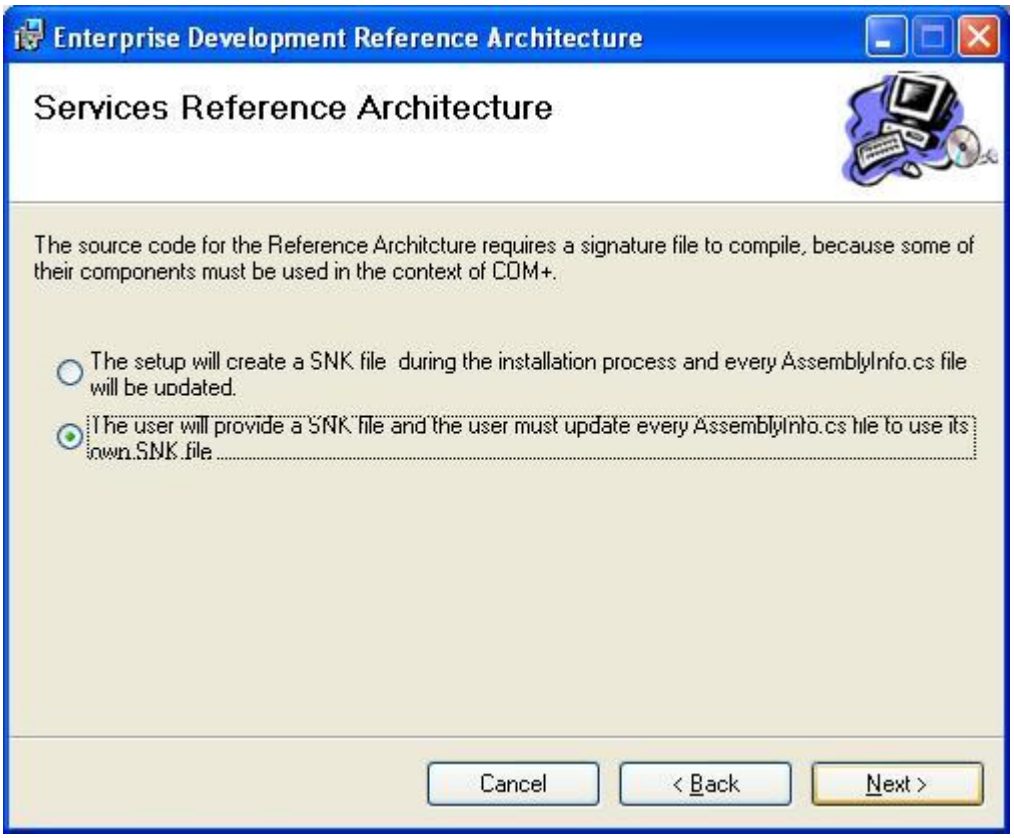


Figura 5-15: Instalación de EDRA (4)[182]

Seleccionamos la carpeta de destino donde se va a instalar el componente.

[182] Elaboración Propia

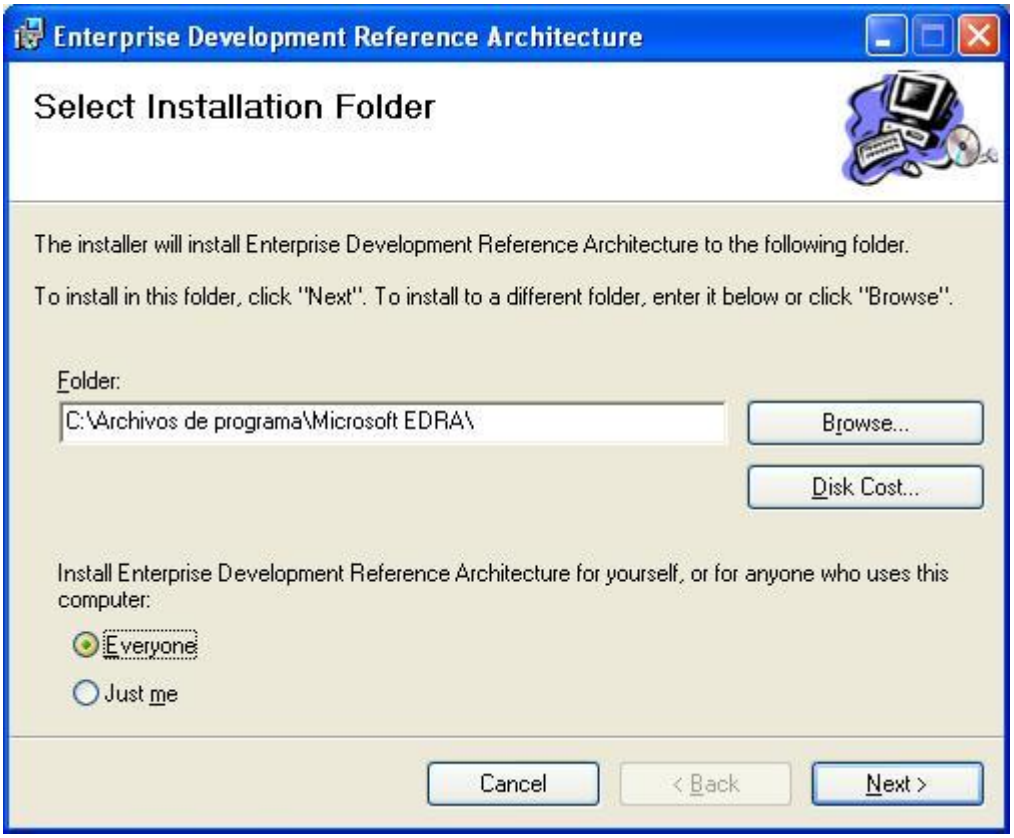


Figura 5-16: Instalación de EDRA (5)[183]

A continuación se desplegará una pantalla para seleccionar el archivo SNK (strong name key) que permite firmar los assemblies del proyecto con un nombre fuerte. El archivo se encuentra en la carpeta Requisitos del CD que acompaña al texto.

[183] Elaboración Propia

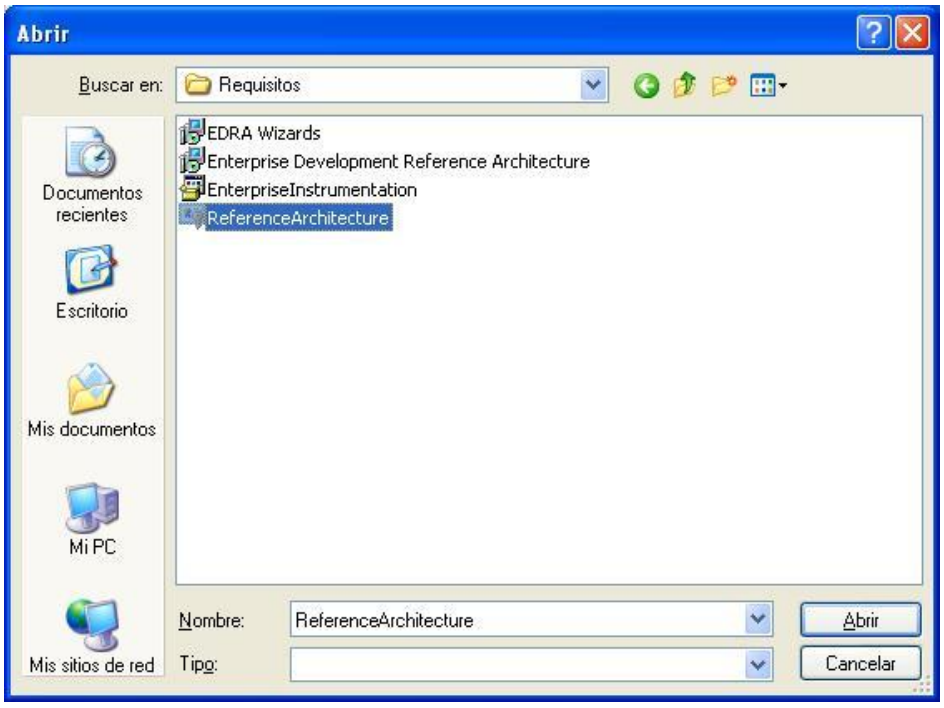


Figura 5-17: Instalación de EDRA (6)[184]

Para finalizar la instalación hacemos clic en finalizar y se nos despliega una pantalla de información del componente.

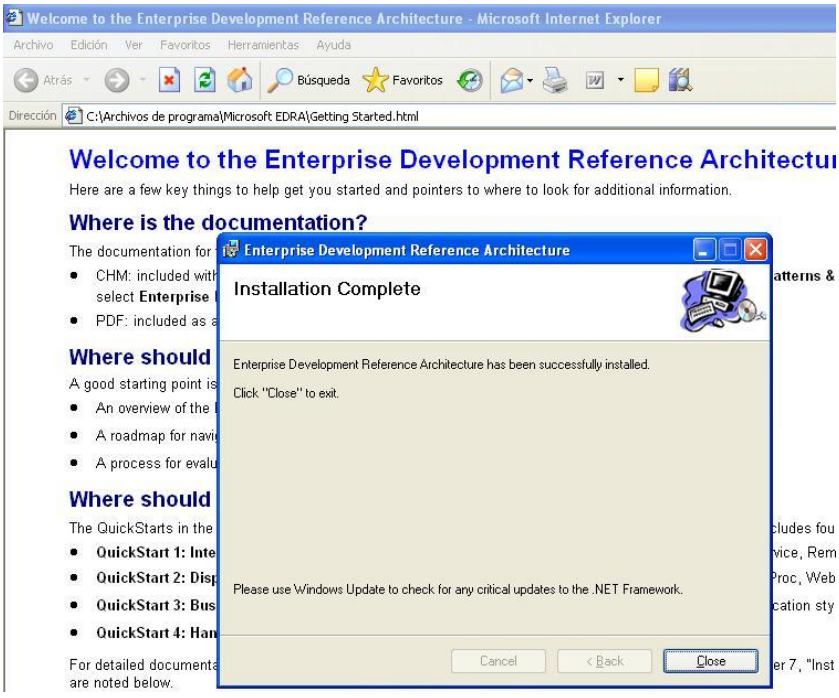


Figura 5-18: Instalación de EDRA (7)[185]

[184] Elaboración Propia
[185] Elaboración Propia

5.2.1.2.3. Instalación de EDRA Wizards

Para instalar Edra Wizards nos dirigimos a la carpeta Requisitos y hacemos doble clic en “EDRA Wizards.exe”.

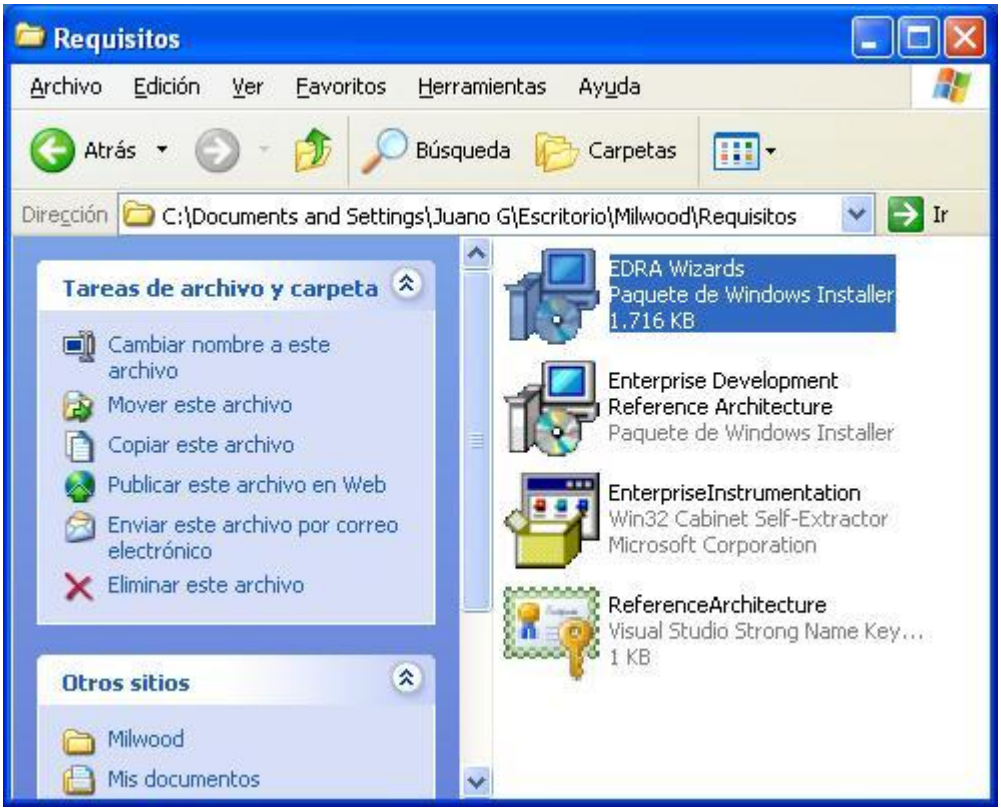


Figura 5-19: Instalación de EDRA Wizards (1)[186]

Clic en siguiente, y aceptar los términos de la licencia.

[186] Elaboración Propia

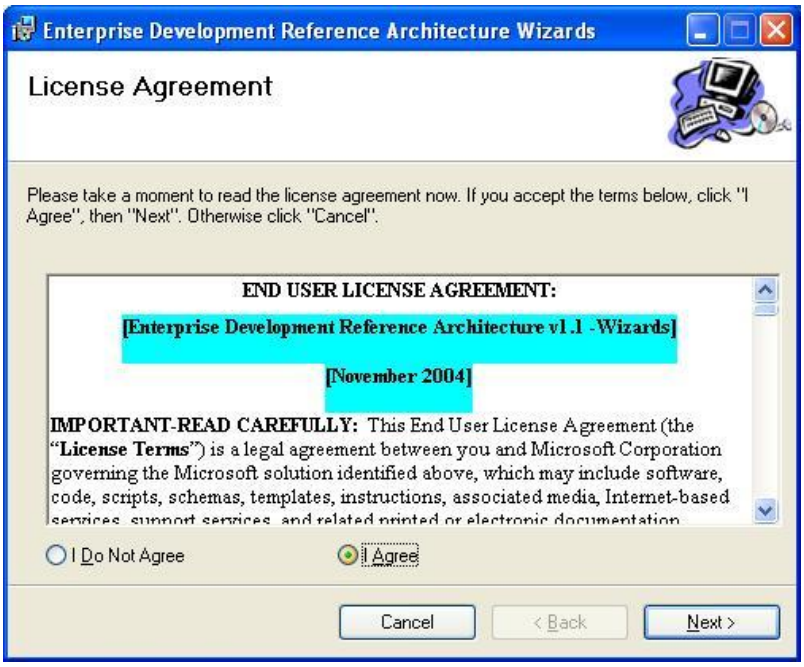


Figura 5-20: Instalación de EDRA Wizards (2)[187]

Seleccionamos la carpeta de destino donde se va a instalar el componente.

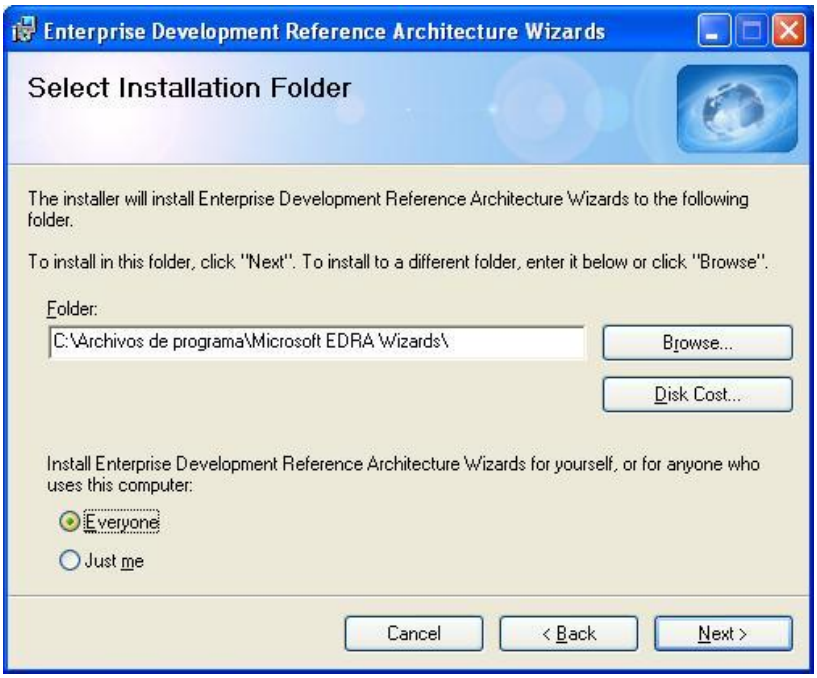


Figura 5-21: Instalación de EDRA Wizards (3)[188]

[187] Elaboración Propia

Para finalizar la instalación hacemos clic en finalizar.

5.2.1.2.4. Instalación de Microsoft application block: User Interface Process 2.0

Para instalar User Interface Process nos dirigimos a la carpeta Requisitos y hacemos doble clic en “User Interface Process 2.0.ese”.

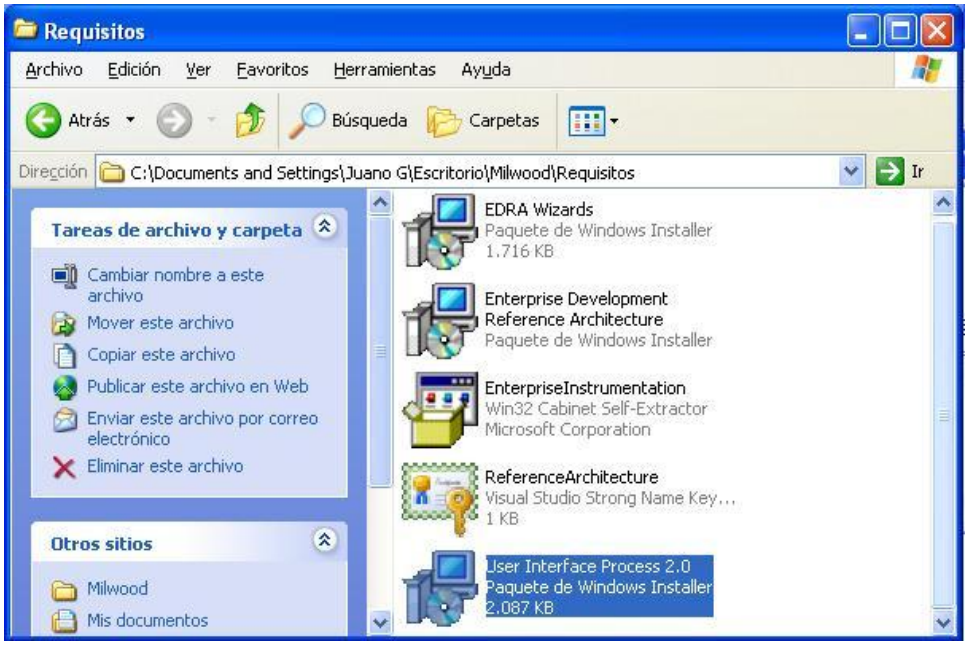


Figura 5-22: Instalación de UI Process[189]

Clic en siguiente, y aceptar los términos de la licencia.

---

[188] Elaboración Propia
[189] Elaboración Propia

Figura 5-23: Instalación de UI Process (2)[190]

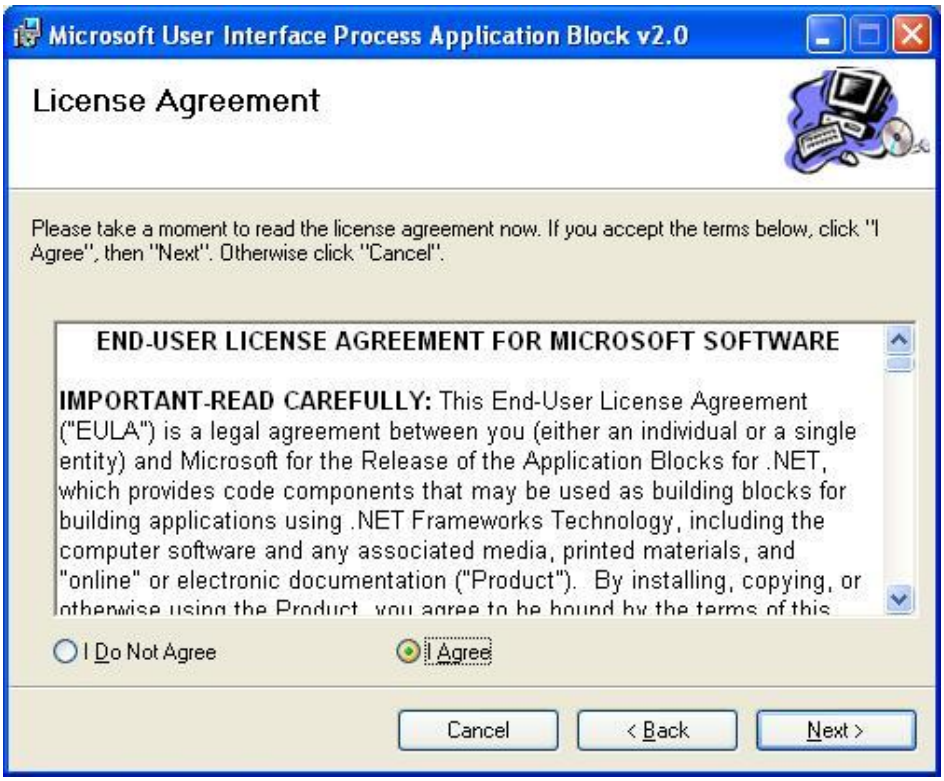


Figura 5-24: Instalación de UI Process (3)[191]

[190] Elaboración Propia

Seleccionamos la carpeta de destino donde se va a instalar el componente.

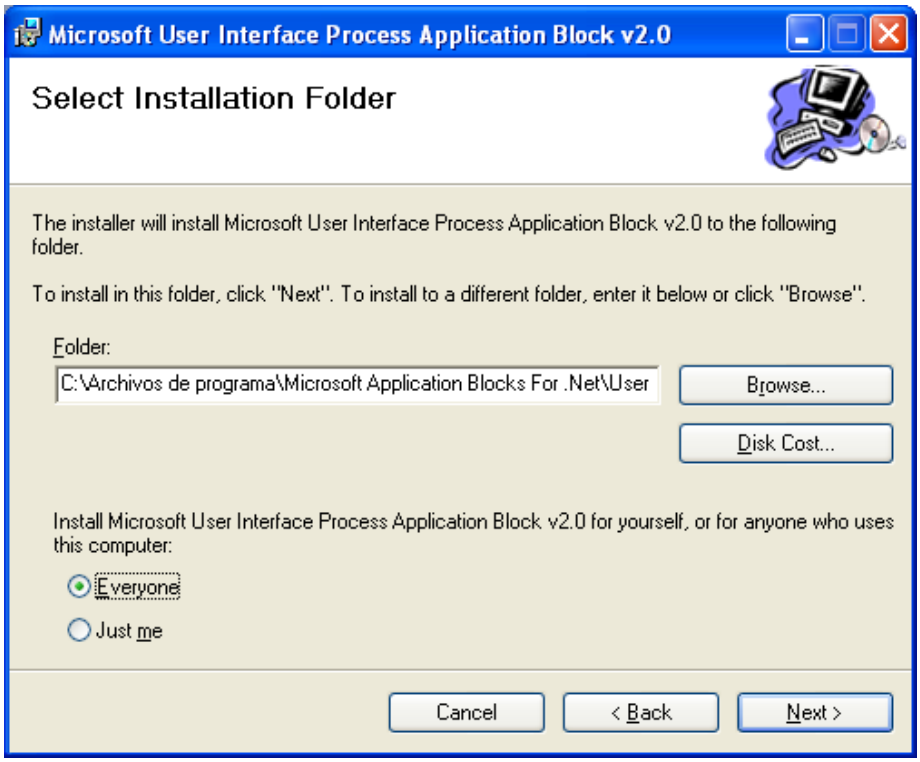


Figura 5-25: Instalación de UI Process (4)[192]

Para finalizar la instalación hacemos clic en cerrar.

5.2.1.3. Instalación del framework Millwood

Ubicamos la carpeta “Framework” del CD que acompaña al texto y la copiamos en la raíz de la unidad (Ej. “c: \”).

Abrimos el contenido de la carpeta y hacemos doble clic sobre el icono “Install Millwood” para poder ejecutar los scripts de instalación del framework. Los

---

[191] Elaboración Propia
[192] Elaboración Propia

scripts crearán las bases de datos y las aplicaciones Web necesarias para el framework.

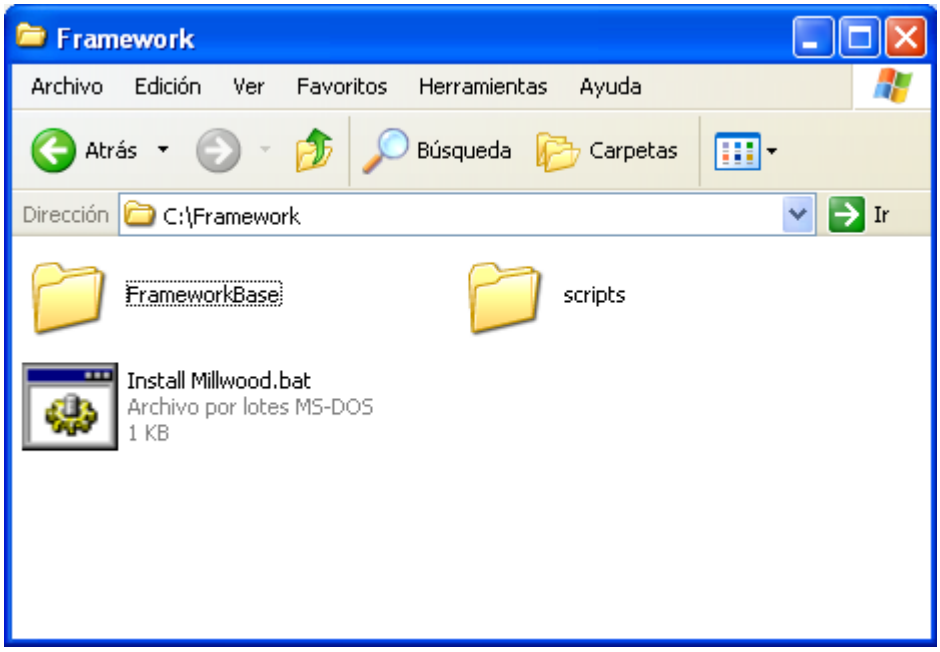


Figura 5-26: Instalación del framework Millwood (2)[193]

En el cuadro de diálogo escribimos la instancia de la base de datos SQL Server para generar la base de datos de soporte de EDRA y la base de datos de ejemplo.

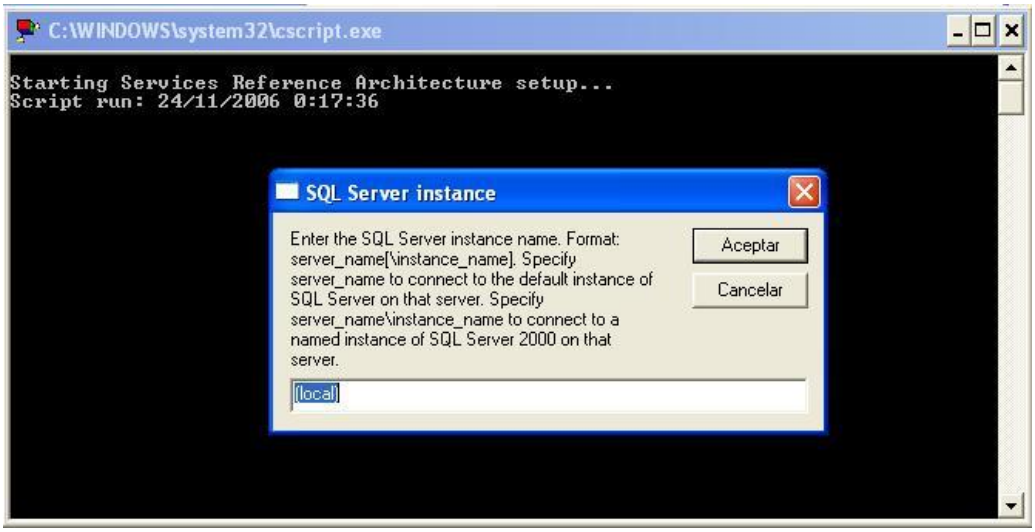


Figura 5-27: Instalación del framework Millwood (3)[194]

[193] Elaboración Propia
[194] Elaboración Propia

Una vez finalizada la instalación y configuración de los componentes del framework podemos ver una pantalla de resumen.

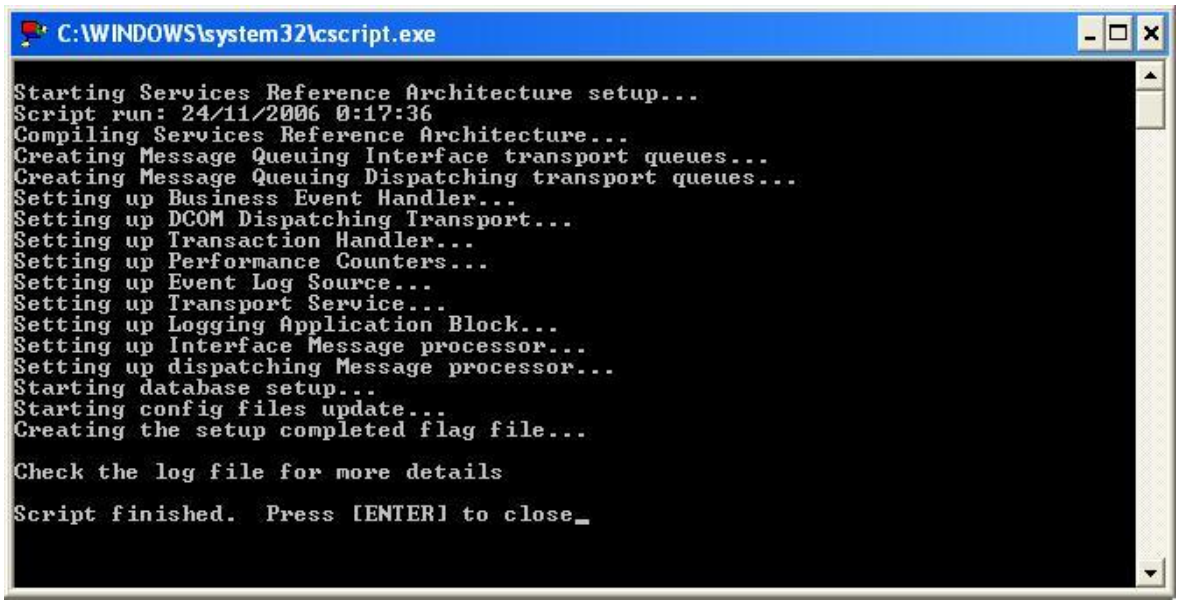


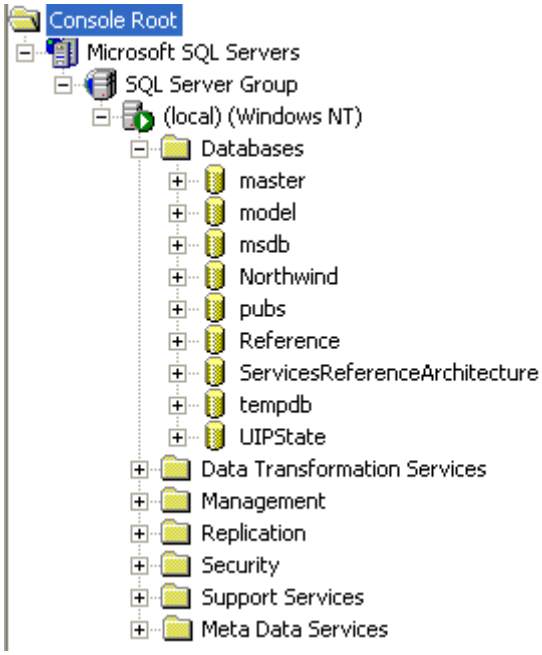
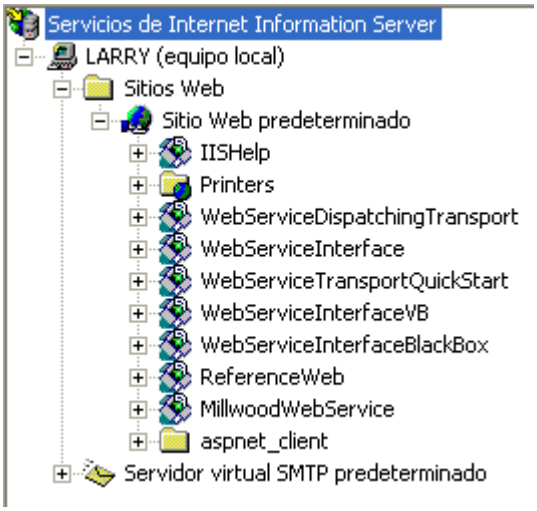
Figura 5-28: Instalación del framework Millwood (4)[195]

5.2.1.4. Comprobación de la instalación de los requerimientos

Para comprobar que todas las partes del framework están correctamente instaladas y configuradas abrimos SQL Server Enterprise Manager e Internet Information services respectivamente y podremos visualizar lo siguiente:

[195] Elaboración Propia

Tabla 5-1: Comprobación de la instalación de los requerimientos

Bases de datos: SQL Server Enterprise Manager	Aplicaciones Web: Internet Information Services
	

En SQL Server deben haberse instalado las siguientes bases de datos:

- **Reference**: aquí se instalan las tablas que servirán de base para la aplicación prototipo y para los ejemplos base de “Millwood”.
- **ServiceReferenceArchitecture**: Contiene las tablas que requiere EDRA.
- **UIPState**: Contiene las tablas necesarias para la el componente de ProcessUI y para almacenar los estados de sesiones persistentes de los usuarios.

En Internet Information Services deben haberse creado los siguientes directorios virtuales o aplicaciones:

- **WebServiceDispatchingTransport, WebServiceInterface, MillwoodWebService**: Necesarios para las interfaces de transporte y EDAF.
- **ReferenceWeb**: Contiene la aplicación web prototipo y la aplicación de ejemplo de Millwood.

5.2.2. Estructura del framework “Millwood” en Visual Studio .net 2003

Para abrir la solución en Visual Studio 2003 nos dirigimos a la carpeta creada en el punto 5.2.1.3 y abrimos la carpeta “Millwood\FrameworkBase\Source\Millwood” aquí encontramos el archivo de solución Millwood, hacemos doble clic para abrir la solución en Visual Studio.

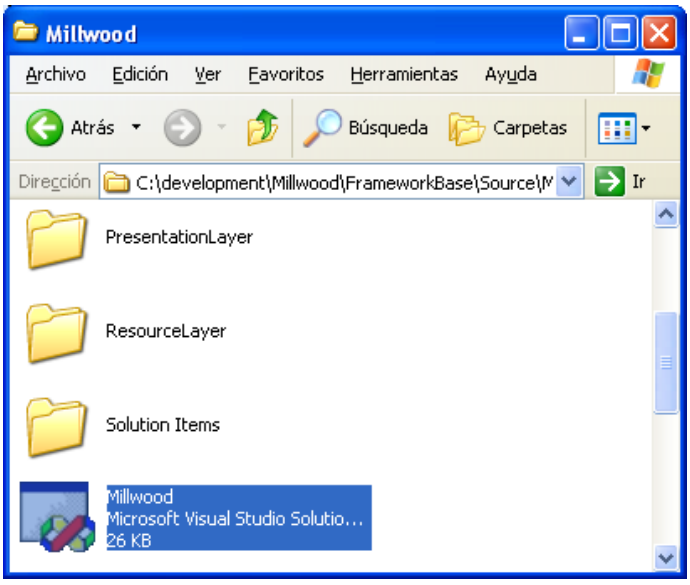


Figura 5-29: Archivo de la solución del Framework "Millwood" Visual Studio .net 2003[196]

[196] Elaboración Propia

5.2.2.1. Enterprise templates

Dos o más proyectos pueden estar agrupados en un conjunto de proyectos denominados Visual Studio Enterprise templates. “Millwood” está estructurado por un conjunto de enterprise templates que representan las capas primarias o bloques de la aplicación que se detallan en la sección 4.2.2 Diagrama de bloques. Los bloques de la solución son:

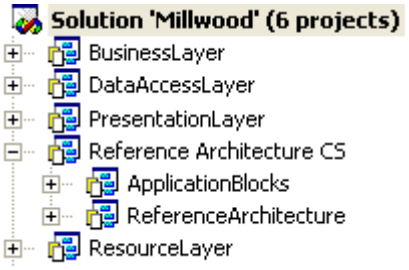


Figura 5-30: Bloques de proyectos del Framework "Millwood”[197]

- PresentationLayer: representa el bloque de “Presentación” y el bloque de “Process UI”.
- BusinessLayer: representa el bloque de “Business Layer”.
- DataAccessLayer: representa el bloque de “Data Access Layer”
- ReferenceArchitectureCS
    - ReferenceArchitecture: representa el bloque de comunicación EDAF dentro de la capa de lógica de negocio.
    - ApplicationBlocks: Son componentes de Microsoft parte del conjunto de componentes Enterprise Library.
- ResourceLayer: Contiene los proyectos de configuración para el framework.

[197] Elaboración Propia

5.2.2.2. Estructura y descripción de los proyectos de la solución

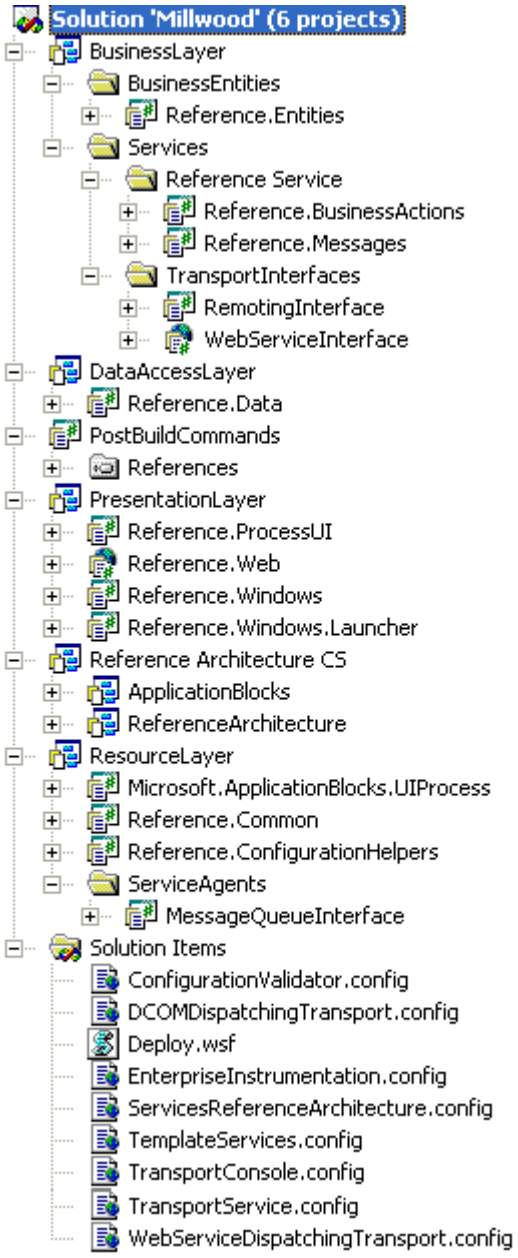


Figura 5-31: Proyectos del Framework "Millwood"[198]

[198] Elaboración Propia

5.2.2.2.1. Proyectos de la Capa de acceso a datos

La capa de acceso a datos se encuentra dentro del Enterprise Template "DataAccessLayer" y contiene los siguientes proyectos:

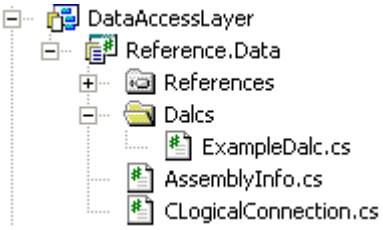


Figura 5-32: Proyectos de la Capa de acceso a datos[199]

Reference.Data

Este proyecto contiene las clases de los DALCs (ver 4.2.5.1 Data access logic components (DALCS)).

Comunicación con la fuente de datos

El proyecto usa la referencia "Microsoft.ApplicationBlocks.Data" que se encuentra dentro de "Reference Architecture CS\ApplicationBlocks"; este proyecto representa al "Data Storage Connector" (ver 4.2.5.3 Data storage connector).

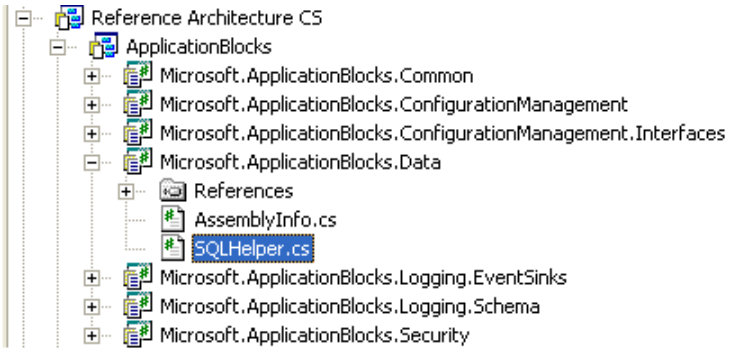


Figura 5-33: Data storage connector[200]

[199] Elaboración Propia

5.2.2.2.2. Proyectos de la Capa de Negocio

La capa de negocio se encuentra dentro del Enterprise Template "BusinessLayer" y contiene los siguientes proyectos:

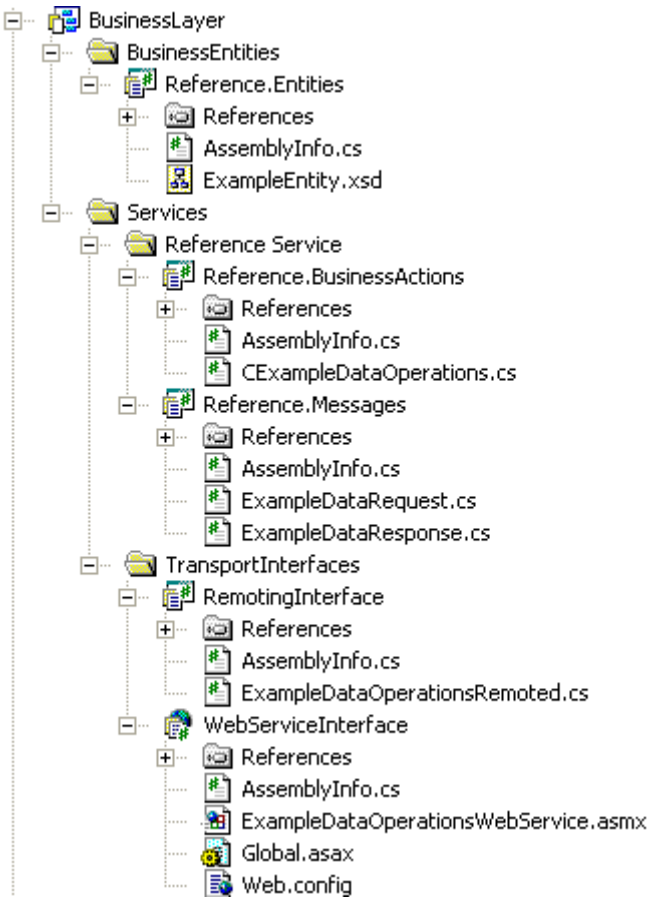


Figura 5-34: Proyectos de la Capa de Negocio[201]

Reference.Entities

Este proyecto contiene las entidades empresariales que son referenciadas en todas las capas de la aplicación (Ver 4.2.4 Entidades Empresariales)

---

[200] Elaboración Propia
[201] Elaboración Propia

Reference.BusinessActions

Este proyecto contiene los Business Actions. (Ver 4.2.7.1 Business actions).

Reference.Messages

Este proyecto contiene los mensajes que son referenciados por los proyectos de la capa de lógica de negocio y parte de la capa de presentación, específicamente en el bloque de Process UI. (Ver 4.2.7.1 Business actions)

RemotingInterface y WebServiceInterface

Estos proyectos pertenecen al bloque de "Interface Transports" (ver 4.2.7.3 Interface transports).

Comunicación EDAF

Parte del bloque de lógica de negocio lo compone el framework EDRA que se encuentra en el enterprise template "ReferenceArchitecture" (ver 4.2.7.2 EDAF).

Figura 5-35: Enterprise template del Framework EDRA[202]

5.2.2.2.3. Proyectos de la Capa de Presentación

La capa de presentación se encuentra dentro del Enterprise Template "PresentationLayer" y contiene los siguientes proyectos:

[202] Elaboración Propia

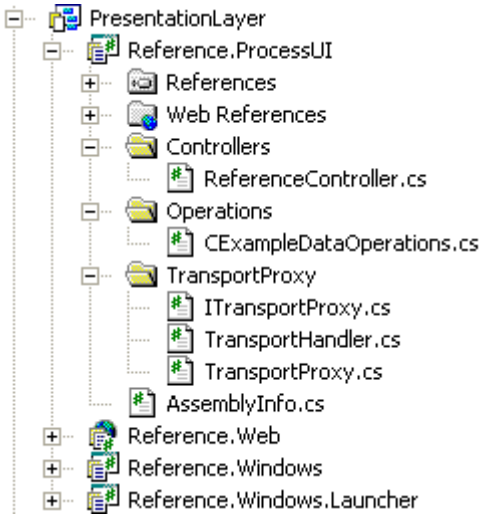


Figura 5-36: Proyectos de la Capa de Presentación[203]

Reference.ProcessUI

Este proyecto representa a todo el bloque de Process UI (ver 4.2.8 UI Process Layer). Los componentes de éste proyecto son:

- Controllers: representan a los controladores de aplicación. (ver 4.2.8.3.1 Controllers).
- Operations: representan a las operaciones de los controladores (ver 4.2.8.2 Operaciones).
- TransportProxy: representa a los transport proxies para la comunicación con la lógica de negocio. (ver 4.2.8.1 Transport proxies)

Componente de Process UI

El bloque utiliza la referencia “Microsoft.ApplicationBlocks.UIProcess” que se encuentra dentro de “Reference Architecture CS\ApplicationBlocks”; este proyecto representa a “UI Process Components” (ver 4.2.8.3 UI process components).

[203] Elaboración Propia

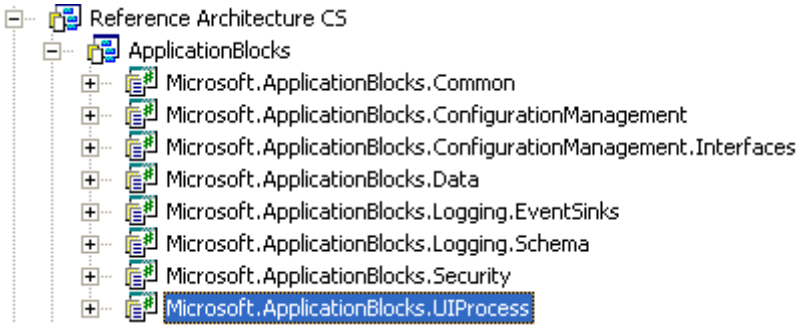


Figura 5-37: Componente de Process UI[204]

Reference.Windows y Reference.Windows.Launcher

Representa la aplicación de escritorio implementada en Windows Forms. La aplicación utiliza el controlador de aplicación de UI Process (Ver 4.2.9.1 Windows Forms).

Reference.Web

Representa la aplicación Web implementada en ASP.net utilizando el controlador de aplicación de UI Process (Ver 0

Web Forms).

5.2.3. Implementación de la base de datos

La base de datos estará alojada en un servidor SQL Server 2000. El framework se instala con una base de datos de referencia denominada "Reference" que será la base para crear la aplicación prototipo.

[204] Elaboración Propia

La base de datos debe cumplir con ciertos requerimientos para que sea posible su implementación correcta utilizando el modelo de acceso a datos definido en el Framework.

Los requerimientos de la base de datos son los siguientes:

- Todas las tablas deben tener una columna de clave primaria, esta columna debe ser definida como un tipo de dato numérico entero (int, long, byte, o cualquiera compatible), y debe ser auto numérico. La clave primaria de una tabla debe ser solamente esta columna, no deben existir claves primarias de varias columnas ni de otros tipos de datos.

- Para todas las tablas se deben crear procedimientos almacenados. Por cada tabla en caso de que se desee realizar todas las operaciones normales deberán existir por lo menos cinco procedimientos almacenados que corresponden a las operaciones de: Insert, Update, Delete, SelectAll, SelectByPrimaryKey, que deben cumplir las siguientes características:

    - Insert: Debe tener un número de parámetros correspondiente al número de columnas cuyos valores serán insertados, nunca se pasará como parámetro el valor de la clave primaria y al final del procedimiento deberá retornarse la fila insertada de la base de datos.
    - Delete, el único parámetro debe ser el valor de la clave primaria a eliminar.

- Update, el procedimiento deberá tener el mismo número de parámetros que el número de columnas de la tabla. Un update deberá afectar a solamente una fila de la base de datos, por lo que en la sentencia WHERE del update se deberá siempre comprobar la clave primaria enviada para realizar la actualización.
- SelectAll, no debe tener parámetros y retornará todas las filas de la tabla sin ningún tipo de filtrado.
- SelectByPrimaryKey, debe tener un único parámetro entero correspondiente al valor de la clave primaria de la fila buscada, retornará una única fila de la base de datos.

Creación de las tablas de la aplicación

El modelo de datos de la aplicación prototipo debe ser implementado en la base de datos, para esto se ejecuta el script SQL desde el archivo “PrototipoSQL” en la carpeta “Aplicación Prototipo\BaseDatos” del CD que acompaña al texto.

5.2.4. Implementación de las entidades empresariales

Las entidades empresariales están implementadas mediante DataSets con tipo (Ver 2.4.10 Ado.net). Cada DataSet representa una entidad empresarial, por lo general una tabla del modelo de datos es representada como una entidad empresarial. (Ver 4.2.4 Entidades Empresariales)

5.2.4.1. Diagrama de clases de una entidad empresarial

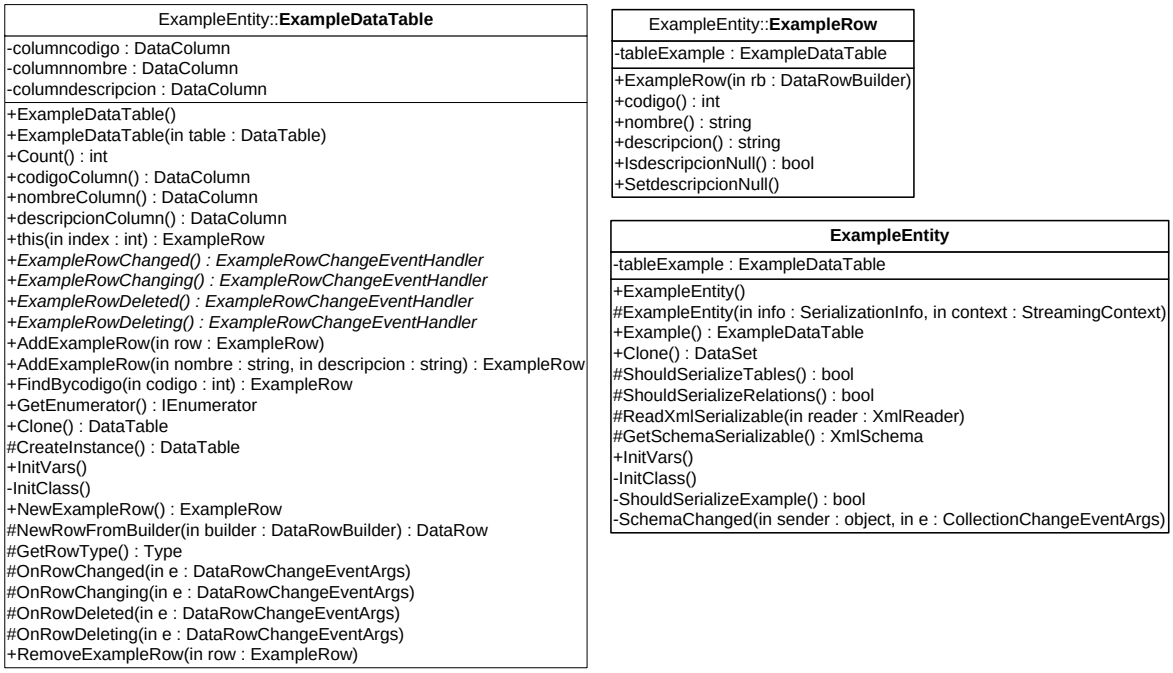


Figura 5-38: Diagrama de clases de una entidad empresarial[205]

5.2.4.2. Creación de una entidad empresarial

Para crear una entidad empresarial se deben seguir algunos pasos definidos a continuación. El ejemplo de creación se toma en base a la Entidad Usuario de la aplicación prototipo.

a. Crear un nuevo DataSet en el proyecto Reference.Entities, en la capa de BusinessLayer, dentro de la carpeta de BusinessEntities. Hacemos clic derecho sobre el proyecto, Add, Add New Item.

[205] Elaboración Propia

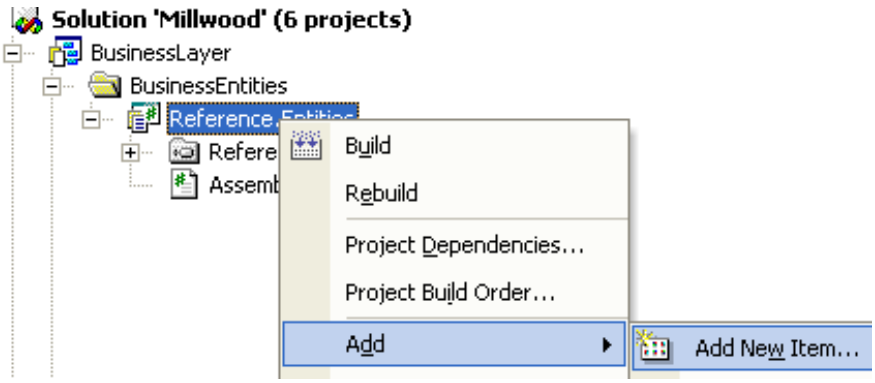


Figura 5-39 Agregar una entidad empresarial[206]

En el cuadro de nuevo item seleccionamos DataSet, dentro de la carpeta Local Project Items, y le damos el nombre UserEntity. Por convención las entidades empresariales se denominarán con la nomenclatura ***<Nombre de la tabla>Entity***.

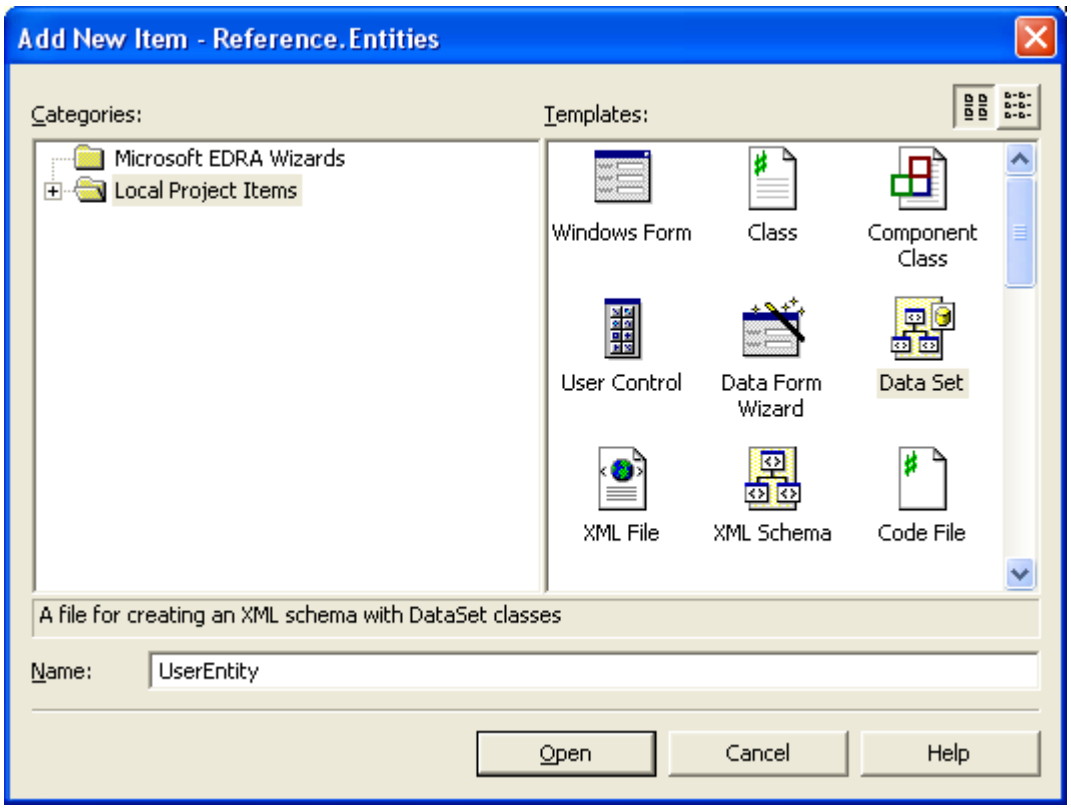


Figura 5-40 Agregar una entidad empresarial 2[207]

[206] Elaboración Propia

b. Abrimos el Server Explorer de Visual Studio. Haciendo clic en View, Server Explorer.

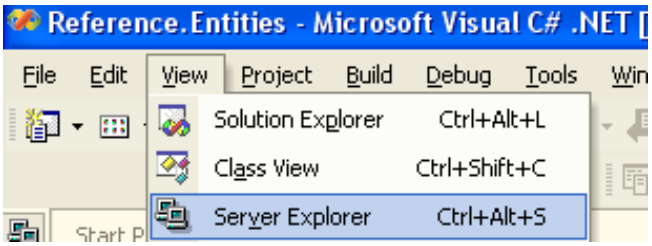


Figura 5-41 Abrir Server Explorer[208]

Luego seleccionamos la computadora donde se encuentra instalado SQL Server, si no se muestra la podemos agregar haciendo clic derecho sobre Servers -> Add Server, luego seleccionamos SQL Servers y expandimos la base de datos del proyecto, y luego tables hasta encontrar la tabla User. Se lo puede hacer también creando una conexión directa a la base de datos, haciendo clic derecho sobre la sección Data Connections del Server Explorer, y llenando los datos necesarios.

[207] Elaboración Propia
[208] Elaboración Propia

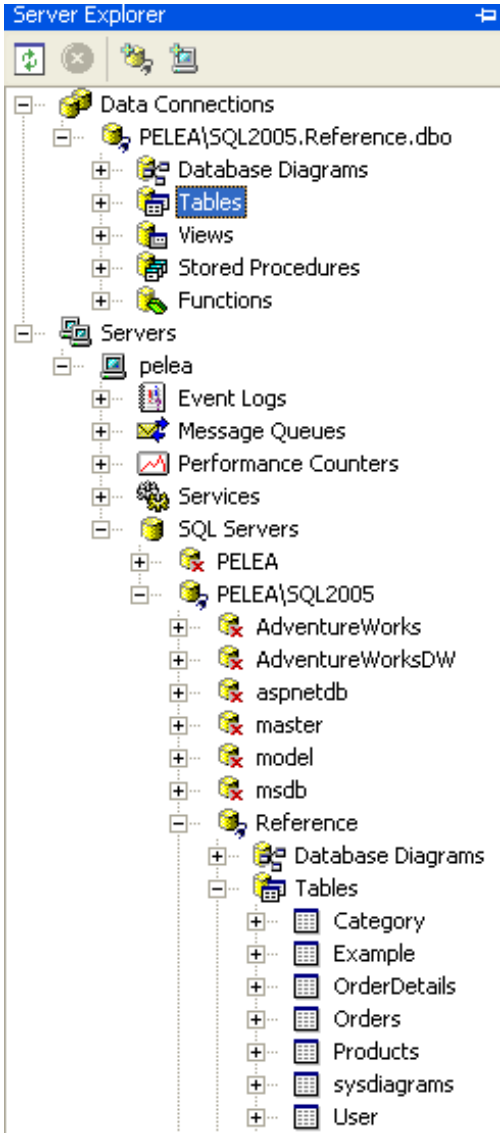


Figura 5-42 Seleccionar la tabla de la entidad empresarial[209]

c. Luego arrastramos la Tabla hacia el diseñador de visual Studio, con esto crearemos la tabla con la misma estructura que la de la base de datos en el diseñador.

[209] Elaboración Propia

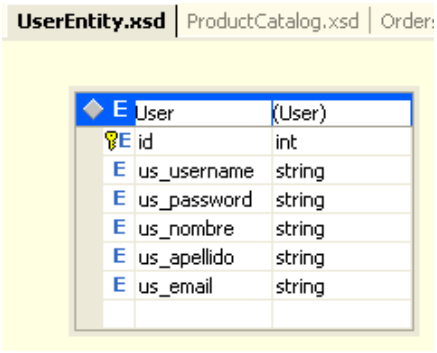


Figura 5-43 Ejemplo entidad empresarial[210]

d. Guardamos y la entidad esta creada.

Para la aplicación prototipo se deberán repetir los pasos anteriores para crear las siguientes entidades empresariales: (el código de cada entidad se encuentra en el CD que acompaña al texto)

- OrderDetailsEntity
- OrdersEntity
- ProductCatalog: Una entidad empresarial compuesta en la que deben estar las tablas de Category y Products relacionadas de la misma manera que en la base de datos.
- UserEntity
- UserTaskEntity

[210] Elaboración Propia

5.2.5. Implementación de la capa de acceso a datos

5.2.5.1. SQL Helper

SqlHelper es un componente de acceso a datos desarrollado por Microsoft, parte de un conjunto de componentes denominado Enterprise Library, que actúa como un Data Storage Connector dentro del framework. (Ver 4.2.5.3 Data storage connector -3-).

5.2.5.2. Clase CLogicalConnection

La clase CLogicalConnection es la clase encargada de operar con el Data Storage Connector (SqlHelper). Esta es una clase abstracta de la que se derivan los componentes lógicos de acceso a datos. En esta clase se encuentra encapsulada toda la funcionalidad básica, así como la forma de conexión a la base de datos que los componentes de acceso a datos deben seguir.

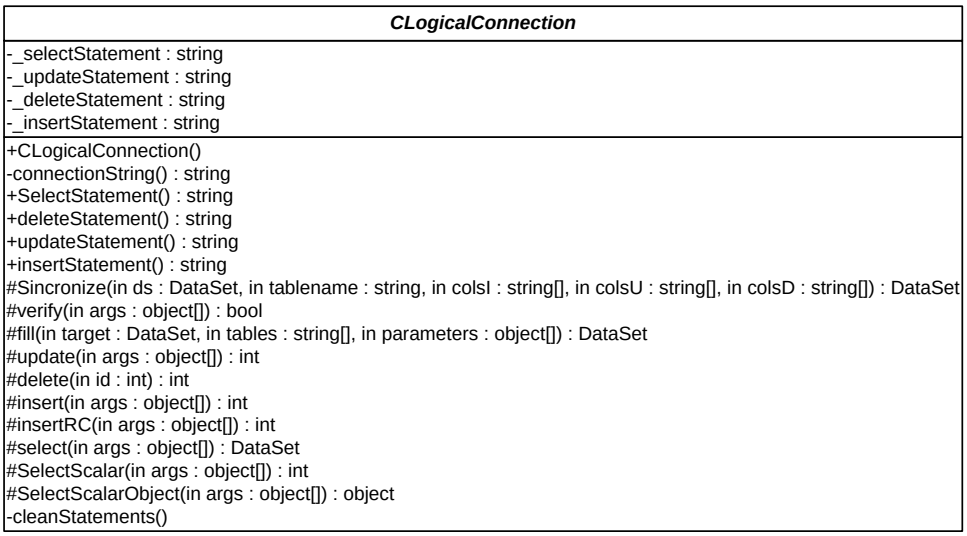


Figura 5-44: Diagrama de clases de CLogicalConnection[211]

[211] Elaboración Propia

5.2.5.3. DALCS

La implementación de los DALCS esta directamente relacionada con la base de datos, por lo que al realizar la implementación de los mismo debemos realizar tareas de programación y tareas sobre la base de datos. Al igual que las entidades empresariales, un DALC usualmente corresponde a una tabla de una base de datos. (Ver 4.2.5.1 Data access logic components (DALCS) -4-)

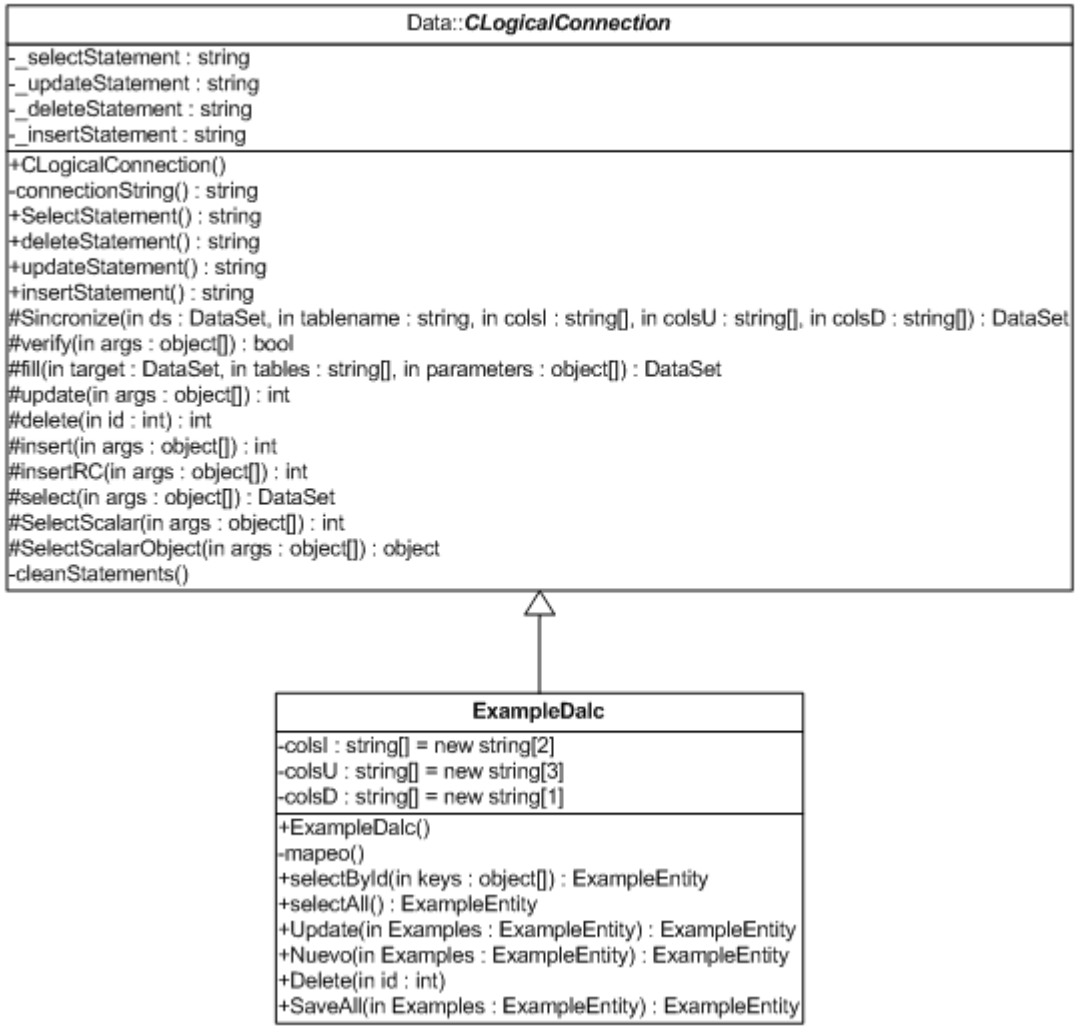


Figura 5-45: Diagrama de clases de un DALC[212]

[212] Elaboración Propia

Para la implementación de un DALC seguimos los siguientes pasos:

Crear una nueva clase en, DataAccessLayer, Reference.Data, dentro de la carpeta Dalcs. Usando la opción Add Class.

Figura 5-46 Crear una nueva clase[213]

Le damos el nombre correspondiente a la clase, por convención se denominará a todas las clases DALC como ***<Nombre de la tabla>Dalc***. Nosotros vamos a crear como ejemplo el Dalc para la tabla de nombre User de la aplicación prototipo, por lo que ingresamos UserDalc en el nombre de la clase.

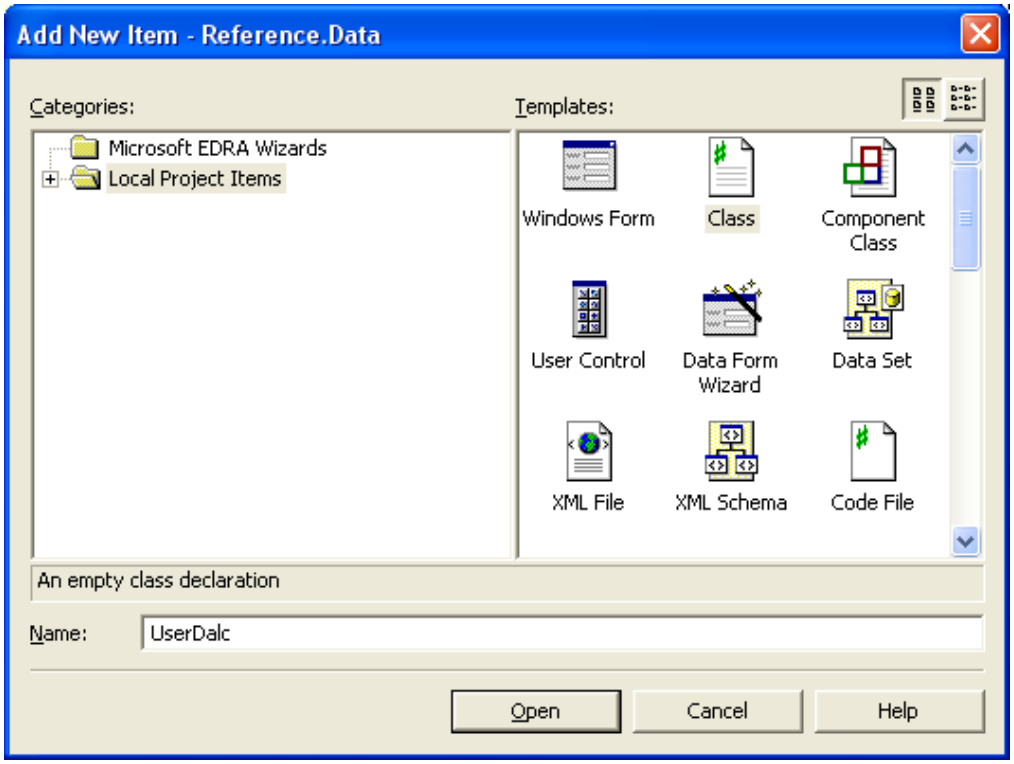


Figura 5-47 Nombrar la clase del Dalc[214]

[213] Elaboración Propia

Definimos los includes necesarios en la cabecera y heredamos la clase CLogicalConnection.

Código 5-1 Cabecera de un Dalc

```
using System;
using System.Collections;
using Reference.Entities;

namespace Reference.Data.Dalcs
{
	/// <summary>
	/// UsereDalc. A Dalc Used to process User data
	///
	/// Authors: Ivan Fernandez
	///          Juan Jalil
	///
	/// This class handles all the operations needed by the application
	/// to perform any task with data associated with users of the
	/// application
	/// </summary>
	Publio class UserDalc : CLogicalConnection
```

Definimos los procedimientos en la base de datos de las operaciones correspondientes. Los procedimientos para la tabla User de la aplicación de ejemplo se encuentran en el script adjunto en el CD de referencia. (Ver 5.2.3 Implementación de la base de datos)

Definimos el mapeo de la entidad empresarial: declaramos tres variables de la clase y creamos una función llamada mapeo, en la que se definirán los nombres de las columnas del DataSet, que serán usados como argumentos para los Stored Procedures de la base de datos y realizamos la llamada en el constructor por defecto.

[214] Elaboración Propia

Código 5-2 Creación de función de Mapeo

Procedimiento

```
CREATE PROCEDURE [dbo].[spReUsuarioInsert]
(
      @us_username varchar(50),
      @us_password varchar(100),
      @us_nombre varchar(50),
      @us_apellido varchar(50)
)
AS
SET NOCOUNT OFF;

INSERT INTO [Usuario] ([us_username], [us_password], [us_nombre],
[us_apellido]) VALUES (@us_username, @us_password, @us_nombre,
@us_apellido);

SELECT id, us_username, us_password, us_nombre, us_apellido FROM
Usuario WHERE (id = SCOPE_IDENTITY())'
END
```

función de Mapeo

```
/// <summary>
/// An array of strings that defines the columns of the DataSet used
/// source for the stored procedure of Inserting data
/// </summary>
private string[] colsI = new string[5];

/// <summary>
/// method to perform the mapping between the name of the columns in
/// the database and the names of the columns in the Entity
/// </summary>
private void mapeo()
{
      colsI[0] = "us_username";
      colsI[1] = "us_password";
      colsI[2] = "us_nombre";
      colsI[3] = "us_apellido";
      colsI[4] = "us_email";
}
```

En este ejemplo se muestra como se debe realizar el mapeo de los argumentos del Stored Procedure de la base con las columnas de la entidad empresarial. El primer argumento del procedimiento es el valor para la columna us_username en el arreglo que representa a las columnas de inserción de datos, el primer valor del arreglo es el nombre correspondiente a la columna nombre en la entidad empresarial que se llama us_username.

A continuación la función de mapeo completa para el Dalc de la tabla Example.

Código 5-3 Función de Mapeo Completa

```
/// <summary>
/// An array of strings that defines the columns of the DataSet used
/// source for the stored procedure of Inserting data
/// </summary>
private string[] colsI = new string[5];

/// <summary>
/// An array of strings that defines the columns of the DataSet used
/// as source for the stored procedure for updating data
/// </summary>
private string[] colsU = new string[6];

/// <summary>
/// An array of strings that defines the columns of the DataSet used
/// as source for the stored procedure for deleting data
/// </summary>
private string[] colsD = new string[1];

/// <summary>
/// Default contructor, performs the initialization of the class
/// </summary>
Publio UserDalc()
{
      mapeo();
}

/// <summary>
/// method to perform the mapping between the name of the columns in
/// the database and the names of the columns in the Entity
/// </summary>
private void mapeo()
{
      colsU[0] = colsD[0] = "id";
      colsU[1] = colsI[0] = "us_username";
      colsU[2] = colsI[1] = "us_password";
      colsU[3] = colsI[2] = "us_nombre";
      colsU[4] = colsI[3] = "us_apellido";
      colsU[5] = colsI[4] = "us_email";
}
```

Luego, creamos las funciones correspondientes a las operaciones. Se asigna a las operaciones las propiedades: SelectStatement, InsertStatement, DeleteStatement o UpdateStatement de la clase base, también el nombre del procedimiento que se va a ejecutar en la base pasando los parámetros

correspondientes. En caso de operaciones de Selección de datos se utilizará la función Fill de la base y en caso de operaciones de modificación de datos la función Sincronize. La función Fill realiza el proceso de carga de datos en el DataSet o entidad. La función sincronize realiza la actualización de los datos del DataSet hacia la base de datos.

Código 5-4 código Ejemplo Operaciones con la Base de Datos

```
/// <summary>
/// method used to find a row by its primary key value
/// </summary>
/// <param name="keys">The value of the primary key that we are looking
/// for within the table managed by this DALC</param>
/// <returns>An entity with the result of the operation</returns>
public UserEntity selectById (object[] keys)
{
      base.SelectStatement = "spReUserSelectById";
      UserEntity User = new UserEntity();
      return base.fill(User,new string[]{User.User.TableName},keys)
            as UserEntity;
}

/// <summary>
/// Updates the data in the database using as source the DataSet
/// provided, only executes Update operations.
/// </summary>
/// <param name="Examples">The entity used as source of the data for
/// the updates</param>
/// <returns>An entity with the result of the operation</returns>
public UserEntity Update (UserEntity Users)
{
      base.updateStatement="spReUserUpdate";
      return base.Sincronize(Users,Users.User.TableName,colsI,
            colsU,colsD) as UserEntity;
}
```

Las funciones de selección deberán recibir los parámetros correspondientes para la ejecución de los procedimientos. Las funciones de modificación de datos deberán recibir, además de cualquier parámetro necesario, el DataSet con los datos que serán usados como base para la actualización a la base.

Se debe crear una función por cada una de las operaciones que se necesite realizar con la base de datos, de esta manera en el Dalc se tendrán al menos las siguientes funciones:

- SelectAll: Selecciona todas las filas de la base sin ningún filtro
- SelectByPrimaryKey: Selecciona una fila de la base usando como filtro el valor de la clave primaria.
- Insert: Inserta filas en la base de datos, y devuelve la ultima fila insertada.
- Delete: Elimina filas de la base de datos.
- Update: Actualiza filas de la base de datos
- SaveAll: Realiza todas las operaciones: eliminación, inserción, y actualización. En una sola llamada.
- Una función SelectBy por cada uno de los campos que sean necesarios que representen claves foráneas de la tabla. En el caso del DALC de User tendrá también la función SelectByEmail, y una función de DoLogin.

Para la implementación de la aplicación prototipo son necesarios los siguientes DALCs: (para el código completo del Dalc referirse al CD que acompaña al texto)

- CatalogDalc
- OrderDetailsDalc
- OrdersDalc
- UserDalc
- UserTaskDalc

5.2.6. Implementación de mensajes

Los mensajes en la aplicación son los objetos enviados entre capas, en estos se encapsula todos los datos necesarios para realizar la llamada a una capa inferior, concretamente al Business Action. Los mensajes deben ser objetos serializables ya que su transferencia entre capas se la realiza mediante serialización. (Ver 4.2.6 Mensajes -7-).

El diagrama de clases de un mensaje es el siguiente:

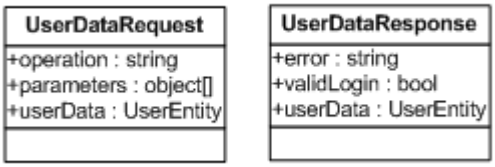


Figura 5-48: Diagrama de clases de un mensaje[215]

La implementación de los mensajes se realiza de la siguiente manera:

Creamos una nueva clase en Business Layer, dentro de la carpeta Services, dentro de la carpeta Reference Service, en el proyecto Reference.Messages.

[215] Elaboración Propia

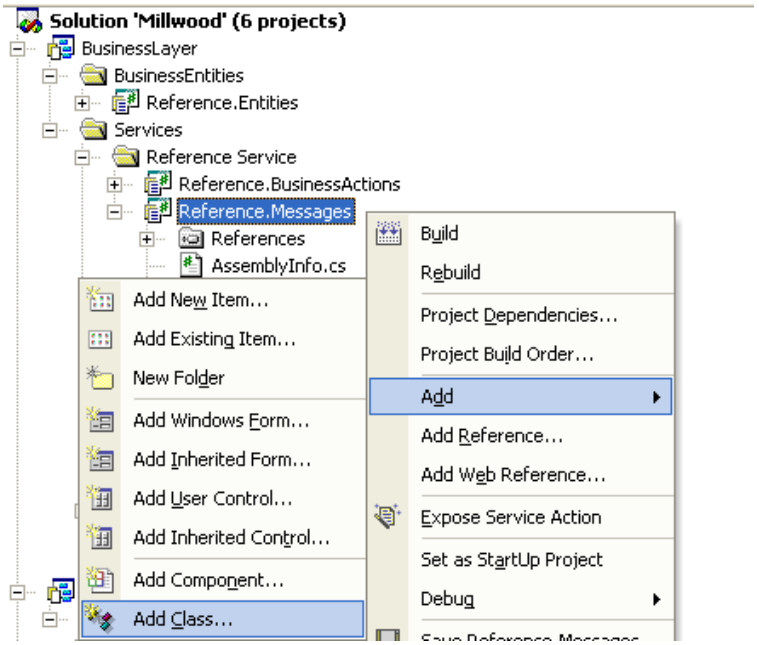


Figura 5-49 Crear una nueva clase mensaje[216]

Los mensajes deberán ser llamados de acuerdo con el Business Action por el que serán invocados y de acuerdo con su rol en la invocación. Primero se crea el mensaje de Request (petición) para el Business Action de User, la clase del mensaje deberá ser llamada: UserDataRequest.

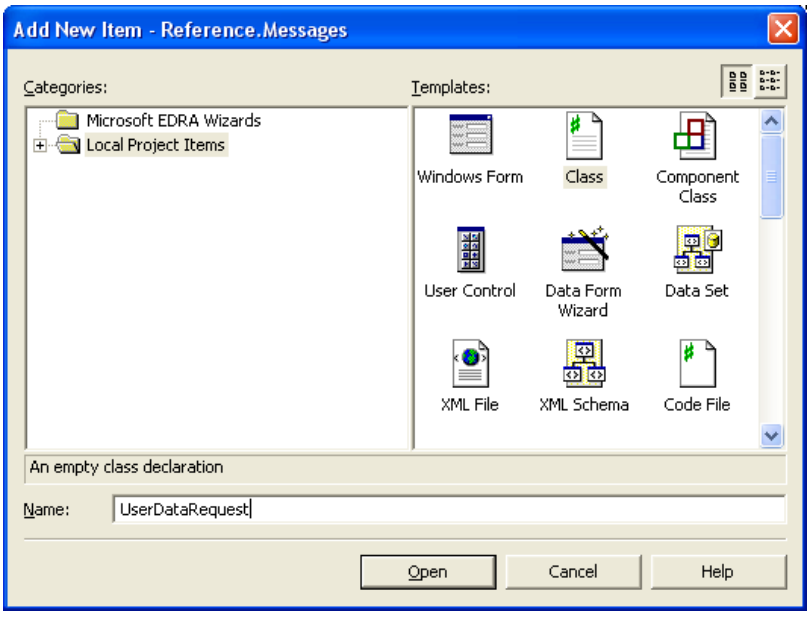


Figura 5-50 Creación Clase Mensaje Request[217]

[216] Elaboración Propia
[217] Elaboración Propia

Una vez creada la clase, agregamos los includes necesarios. En este caso solamente debemos agregar: System.Xml.Serialization, debido a que se utilizará serialización en XML para el procesamiento del mensaje.

Luego debemos agregar los atributos de serialización a la clase:

Código 5-5 Atributos de Serialización

```
/// <summary>
/// ExampleDataRequest. Request Message for User Business Action
///
/// Authors: Ivan Fernandez
///          Juan Jalil
///
/// This class represents a message that is going to be transmitted to the
/// User Business action.
/// </summary>
[XmlRoot(Namespace="http://g5corp.com/",IsNullable=false)]
[Serializable()]
public class UserDataRequest
```

Por último, agregamos todos los campos necesarios para el procesamiento del Business Action. A continuación la clase completa:

Código 5-6 Clase User Data Request

```
//
// UserDataRequest.cs
//
//
// Version:  1.0.0.1
//
//

using System;
using System.Xml.Serialization;

namespace Reference.Messages
{
      /// <summary>
      /// ExampleDataRequest. Request Message for User Business Action
      ///
      /// Authors: Ivan Fernandez
      ///          Juan Jalil
```

```
      ///
      /// This class represents a message that is going to be transmitted
to the
      /// User Business action.
      /// </summary>
      [XmlRoot(Namespace="http://g5corp.com/",IsNullable=false)]
      [Serializable()]
      public class UserDataRequest
      {
            /// <summary>
            /// An string representing the operation that should be
                performed by
            /// the business action
            /// </summary>
            public string operation;

            /// <summary>
            /// The parameters needed to perform the operation
            /// </summary>
            public object[] parameters;

            /// <summary>
            /// any data related to the operation that has to be performed
            /// </summary>
            public Entities.UserEntity userData;
      }
}
```

Luego, deberemos crear de la misma manera el mensaje correspondiente al response del Business Action, mediante el siguiente código del mensaje de response (respuesta):

Código 5-7 Clase User Data Response

```
//
// UserDataResponse.cs
//
//
// Version:  1.0.0.1
//
//

using System;
using System.Xml.Serialization;

namespace Reference.Messages
{
      /// <summary>
      /// UserDataRequest. Request Message for User Business Action
      ///
      /// Authors: Ivan Fernandez
      ///          Juan Jalil
      ///
```

```
        /// This class represents a message that is going to be transmitted
        /// to the User Business action.
        /// </summary>
        [XmlRoot(Namespace="http://g5corp.com/",IsNullable=false)]
        [Serializable()]
        public class UserDataResponse
        {
              /// <summary>
              /// An string representing if an error ocurred in the
                  procesing
              /// </summary>
              public string error;

              /// <summary>
              /// A bool indicating if the login was succesfull
              /// </summary>
              public bool validLogin;

              /// <summary>
              /// Any data related to the operation that was performed
              /// </summary>
              public Entities.UserEntity userData;
        }
}
```

Para la implementación de la aplicación prototipo son necesarios las siguientes clases: (para el código completo de los mensajes referirse al CD que acompaña al texto)

- CatalogDataRequest
- CatalogDataResponse
- UserDataRequest
- UserDataResponse
- UserTaskDataRequest
- UserTaskDataResponse

5.2.7. Implementación de la capa de negocio

5.2.7.1. Creación de un business action

Para crear un business action (Ver 4.2.7.1 Business actions -8-) deberemos seguir uno de los Wizards de EDRA (Ver 5.2.1.2 Instalación de requisitos), para esto seleccionamos Add New Item, en Business Layer, dentro de la carpeta Services, Reference Service, en el proyecto Reference.BusinessActions. Para el ejemplo de la implementación tomaremos el ejemplo de “Usuario” de la aplicación prototipo.

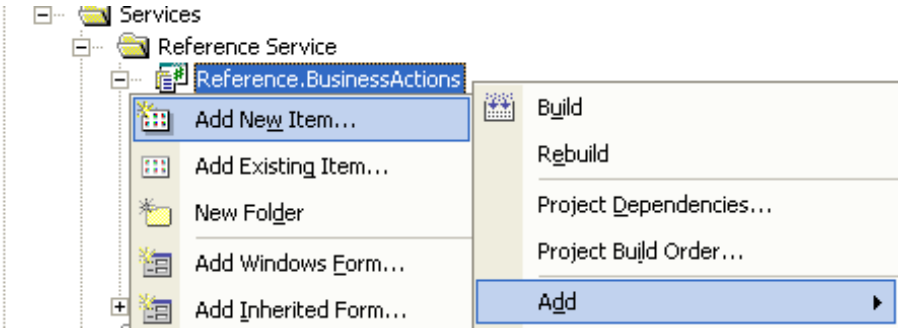


Figura 5-51 Crear un Business Action[218]

Luego, seleccionamos dentro de la carpeta Microsoft EDRA Wizards -> Business Action y le damos el nombre de UserDataOperations

[218] Elaboración Propia

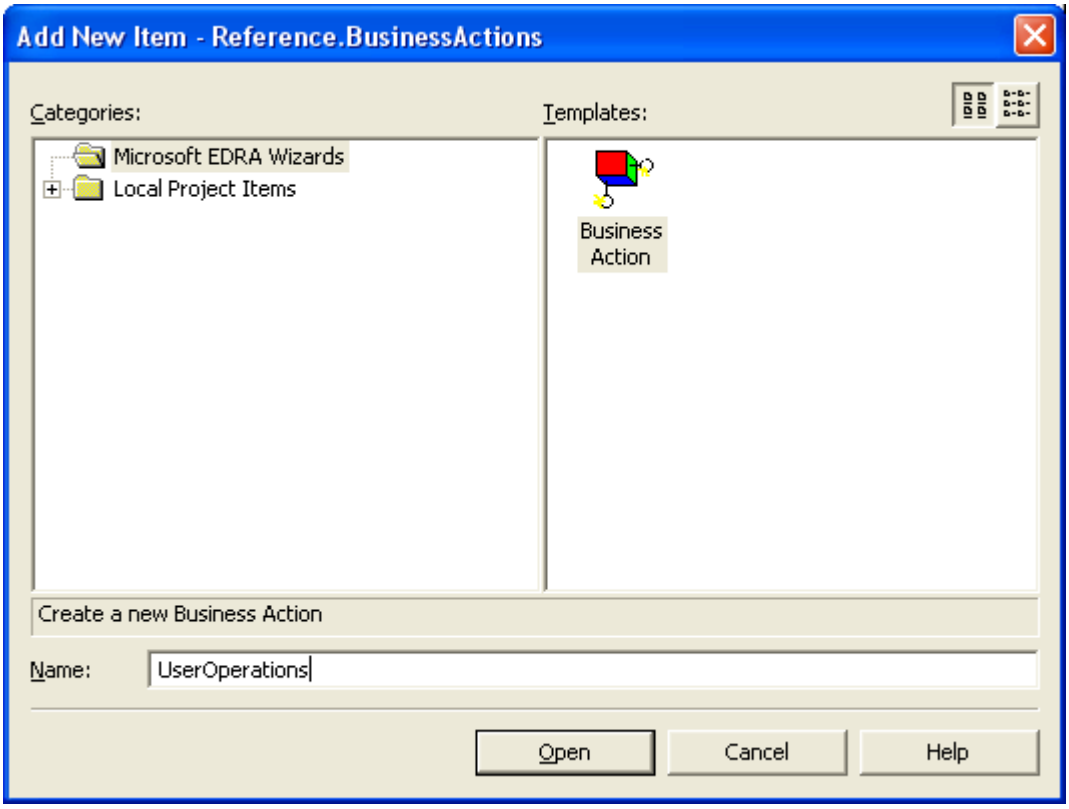


Figura 5-52 Creación Business Action[219]

Luego, al hacer clic en Open se nos presenta el Wizard de EDRA de creación de un Business Action. Seleccionamos next, ingresamos los siguientes nombres:

- Service Action Name: UserOperations

- Business Action Class: CuserOperations

- Business Action Method: UserOperations

[219] Elaboración Propia

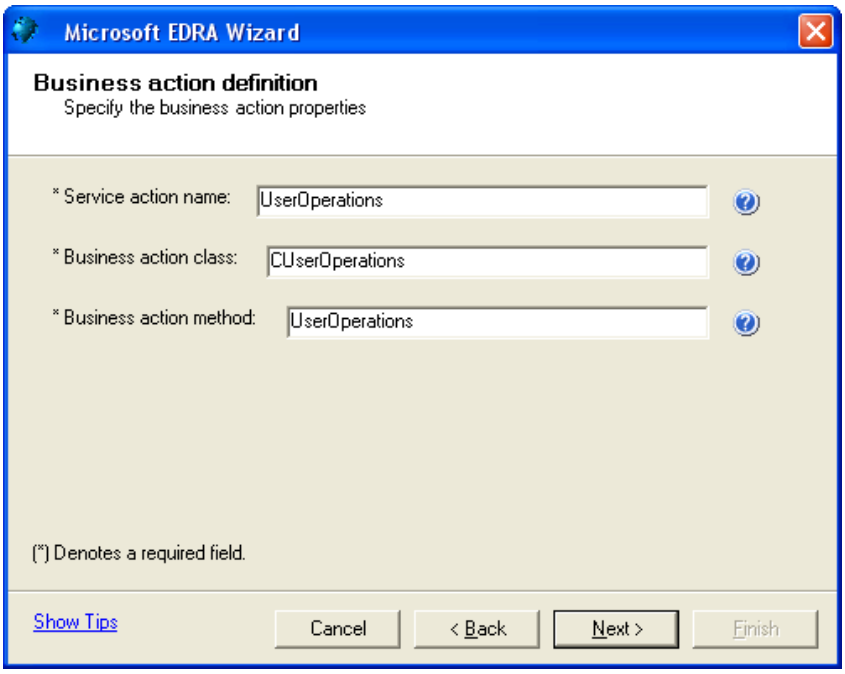


Figura 5-53 Creación de un Business Action[220]

En el siguiente paso del Wizard elegimos el proyecto Messages Project donde creamos los mensajes. (Ver 5.2.6 Implementación de mensajes) y dejamos vacíos los campos de los schemas.

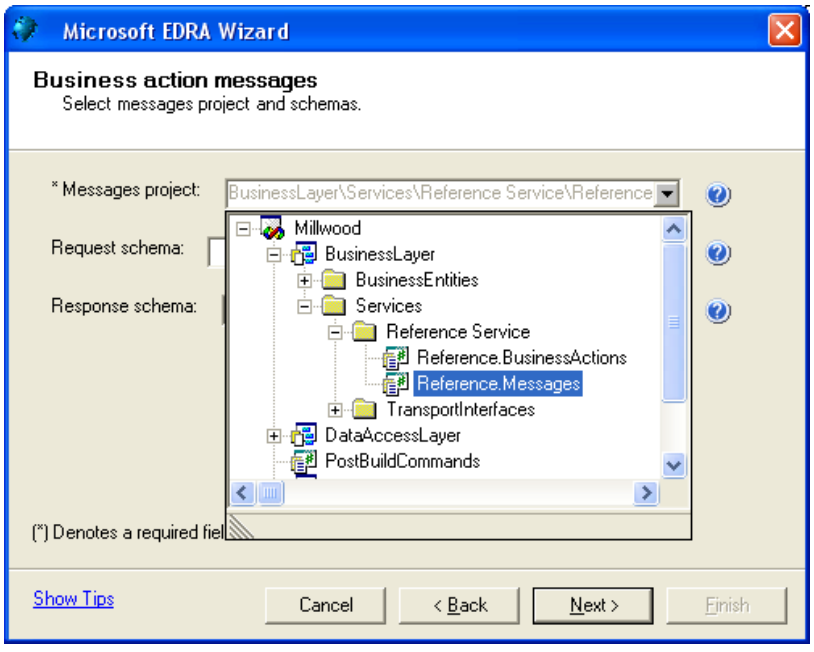


Figura 5-54 Creación de un Business Action (2)[221]

[220] Elaboración Propia
[221] Elaboración Propia

En el siguiente paso del Wizard seleccionamos las clases creadas de los mensajes.

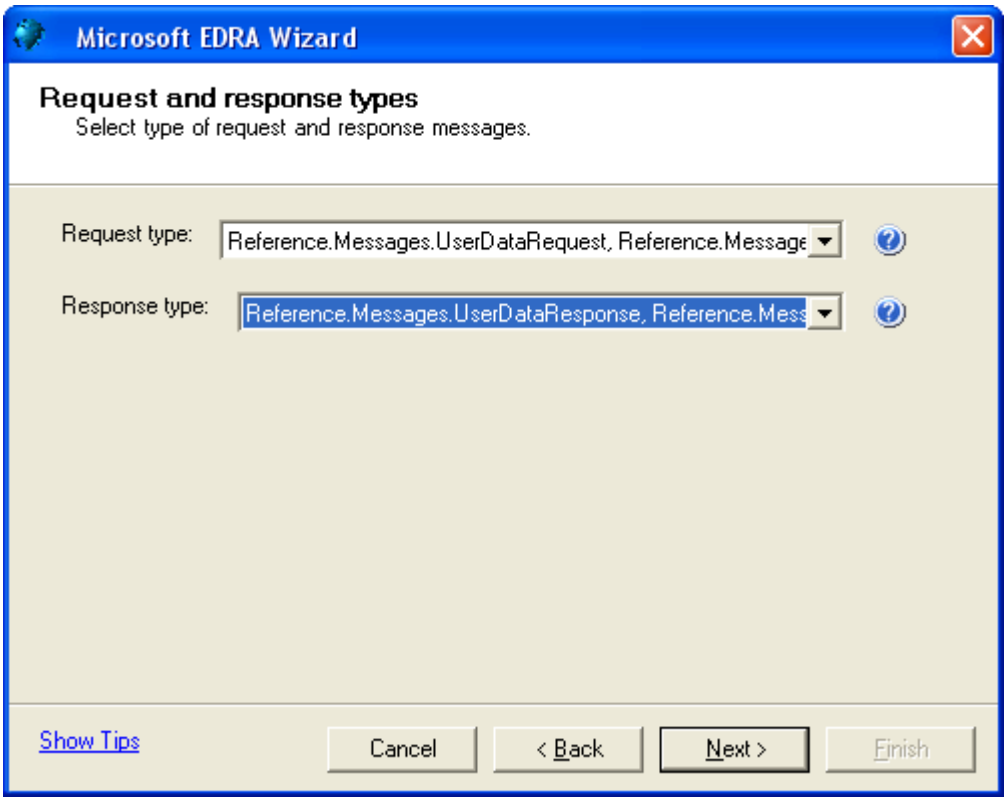


Figura 5-55 Creación de un Business Action (3)[222]

Al hacer clic en siguiente, se nos presenta la pantalla final del wizard, seleccionamos Finish para terminar la creación del Business Action.

5.2.7.1.1. Implementación del Business Action

Luego de creado el business action debemos agregar el código respectivo para que el business action creado realice el proceso de negocio elegido. En este caso el proceso que el Business Action realizará es el manejo del acceso a datos de la tabla “Usuario” de la aplicación prototipo.

[222] Elaboración Propia

Abrimos la clase del Business Action CUserOperations que se encuentra ubicada en: Business Layer en la carpeta Services -> Reference Service dentro del proyecto Reference.BusinessActions.

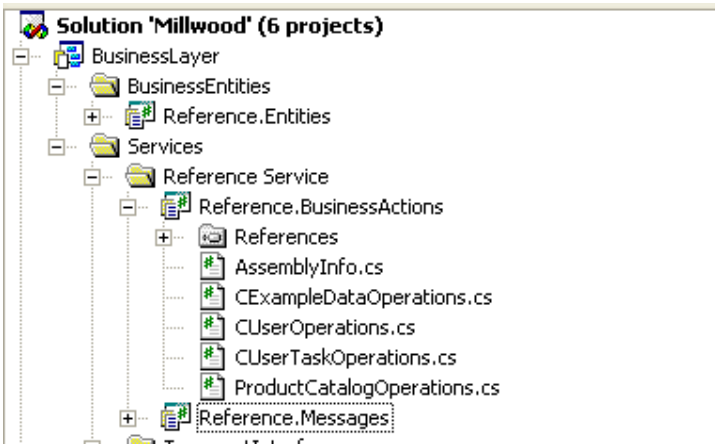


Figura 5-56 Creación de un Business Action (4)[223]

En la clase generada por los wizards de EDRA tenemos el esqueleto de la clase ya creado. Agregamos en la cabecera las sentencias include necesarias: Reference.Data.Dalcs y Reference.Entities ya que vamos a hacer uso de los Dalcs y las Entidades Empresariales. Agregamos el código necesario en la función del Business Action para el procesamiento de los datos:

Código 5-8 Creación de un Business Action (5)

```
using Reference.Data.Dalcs;
using Reference.Entities;
```

[223] Elaboración Propia

```
public Reference.Messages.UserDataResponse UserOperations(
      Reference.Messages.UserDataRequest request)
{
      Reference.Messages.UserDataResponse response =
            new Reference.Messages.UserDataResponse();
      UserDalc dalc=new UserDalc();
      response.validLogin=false;
      try
      {
            switch(request.operation)
            {
                  case "SelectAll":
                        response.userData=dalc.selectAll();
                        break;
                  case "SelectById":

      response.userData=dalc.selectById(request.parameters);
                        break;
                  case "Update":
                        response.userData=dalc.Update(request.userData);
                        break;
                  case "Delete":
                        dalc.Delete(request.userData);
                        request.userData.AcceptChanges();
                        response.userData=request.userData;
                        break;
                  case "Insert":
                        response.userData=dalc.Nuevo(request.userData);
                        break;
                  case "ActualizarTodo":
                        response.userData=dalc.SaveAll(request.userData);
                        break;
                  case "SelectByEmail":

      response.userData=dalc.selectByEmail(request.parameters);
                        break;
                  case "DoLogin":
                        // Keys must contain 2 parameters
                        // 1: Username
                        // 2: Password
                       response.userData=dalc.doLogin(request.parameters);
                        if(response.userData.User.Rows.Count==1)
                              response.validLogin=true;
                        break;
            }
      }
      catch (Exception ex)
      {
            response.userData=null;
            response.error=ex.ToString();
      }
      return response;
}
```

En la aplicación final serán necesarios varios Business Actions con sus respectivos mensajes para que se pueda implementar toda la funcionalidad requerida. Los Business Actions requeridos son listados a continuación:

- CUserOperations
- CUserTaskOperations
- ProductCatalogOperations

El código completo de estos Business actions se encuentra en el CD que acompaña al texto.

5.2.7.2. Publicación de Business Actions en interfaces de transporte

Luego de creado el Business Action debemos exponer el Business action en una interfase de transporte (Ver 4.2.7.3 Interface transports -10-) para que sea consumido por las capas superiores, esto se realiza mediante el wizard de EDRA: Exponer un Business Action.

Para acceder al wizard de exponer un Business Action wizard hacemos clic derecho sobre el proyecto de Business Layer-> Services-> TransportInterfaces -> WebServiceInterface y seleccionamos Expose Service Action.

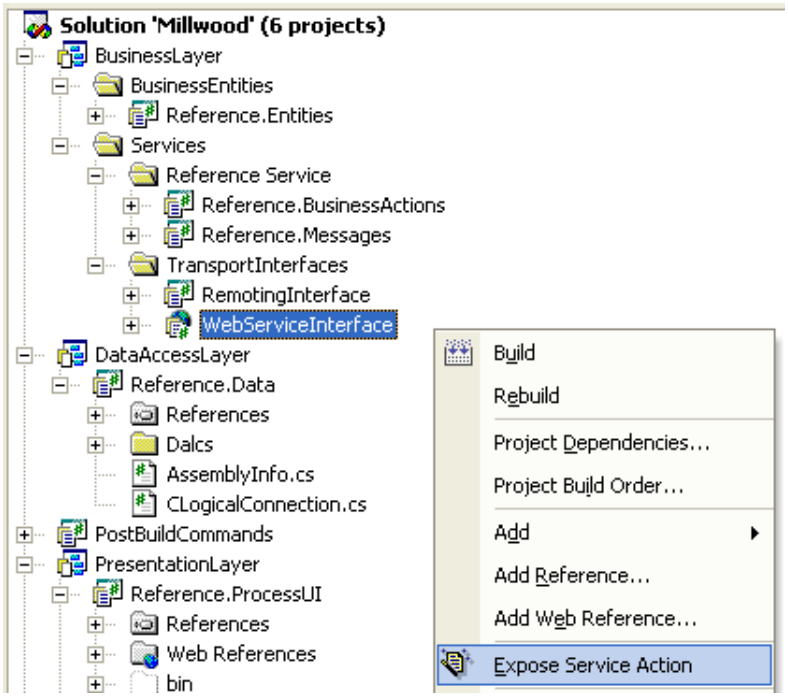


Figura 5-57 Publicar un Business Action (1)[224]

En la pantalla seleccionamos el Business Action a publicar y el transporte. Seleccionamos UserOperations y por transporte elegimos Remoting.

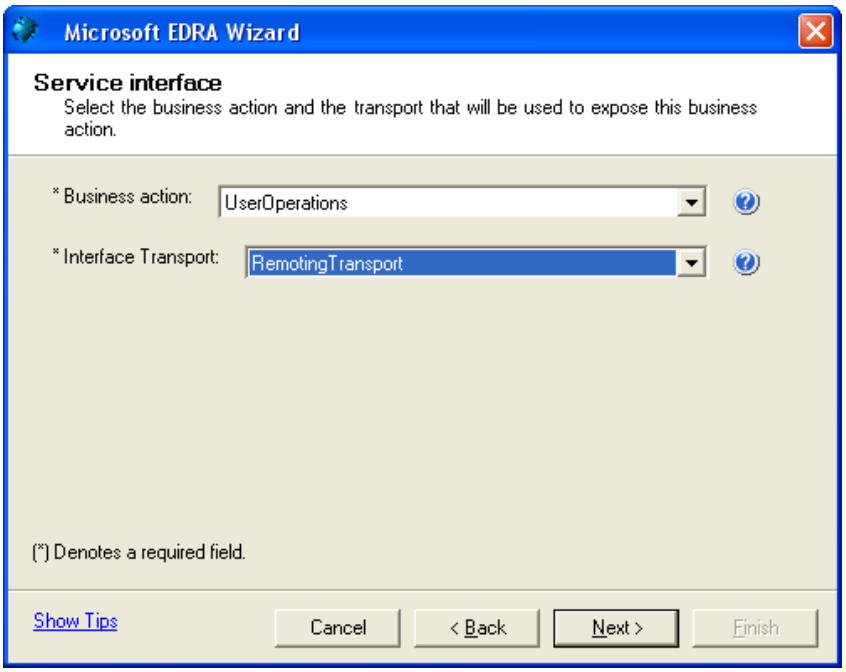


Figura 5-58 Publicar un Business Action (2)[225]

[224] Elaboración Propia
[225] Elaboración Propia

En la siguiente pantalla se acepta los valores por defecto.

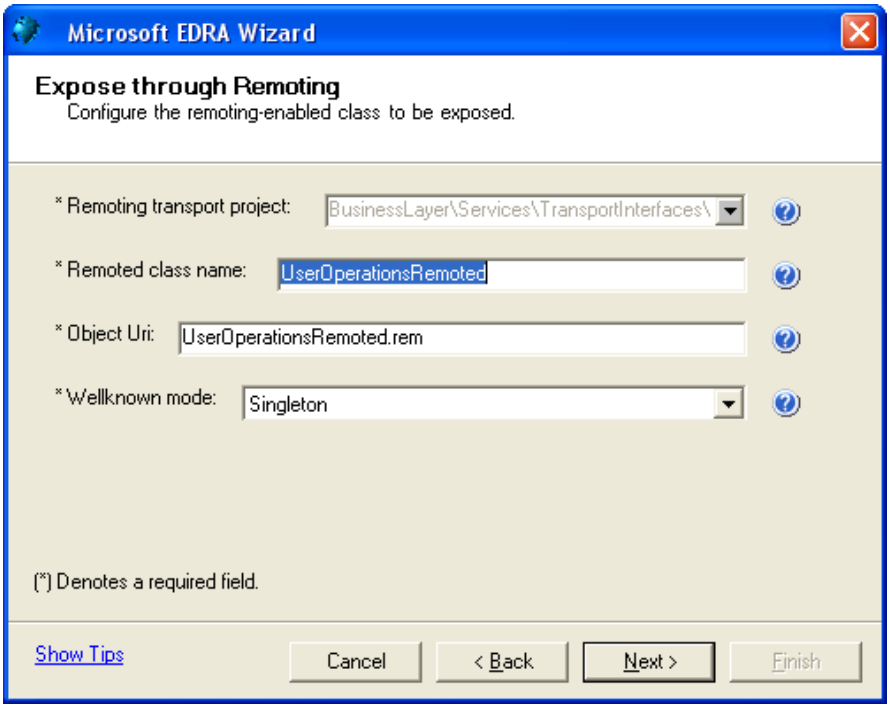


Figura 5-59 Publicar un Business Action (3)[226]

Hacemos clic en siguiente con los valores por defecto de la siguiente pantalla.

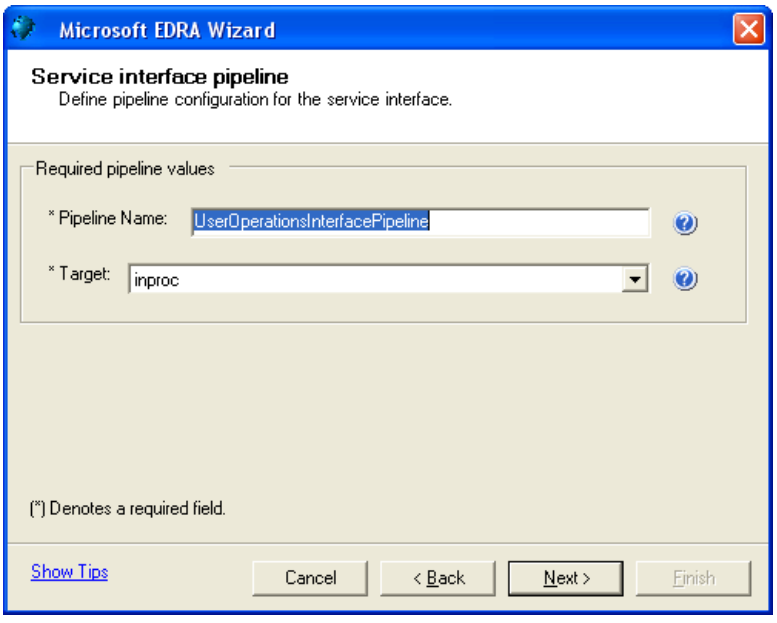


Figura 5-60 Publicar un Business Action (4)[227]

[226] Elaboración Propia
[227] Elaboración Propia

En el siguiente paso elegimos, crear un nuevo pipeline de implementación, y le damos el nombre: UserOperationsImplPipeRemoting, y hacemos clic en Finish, para terminar el wizard.

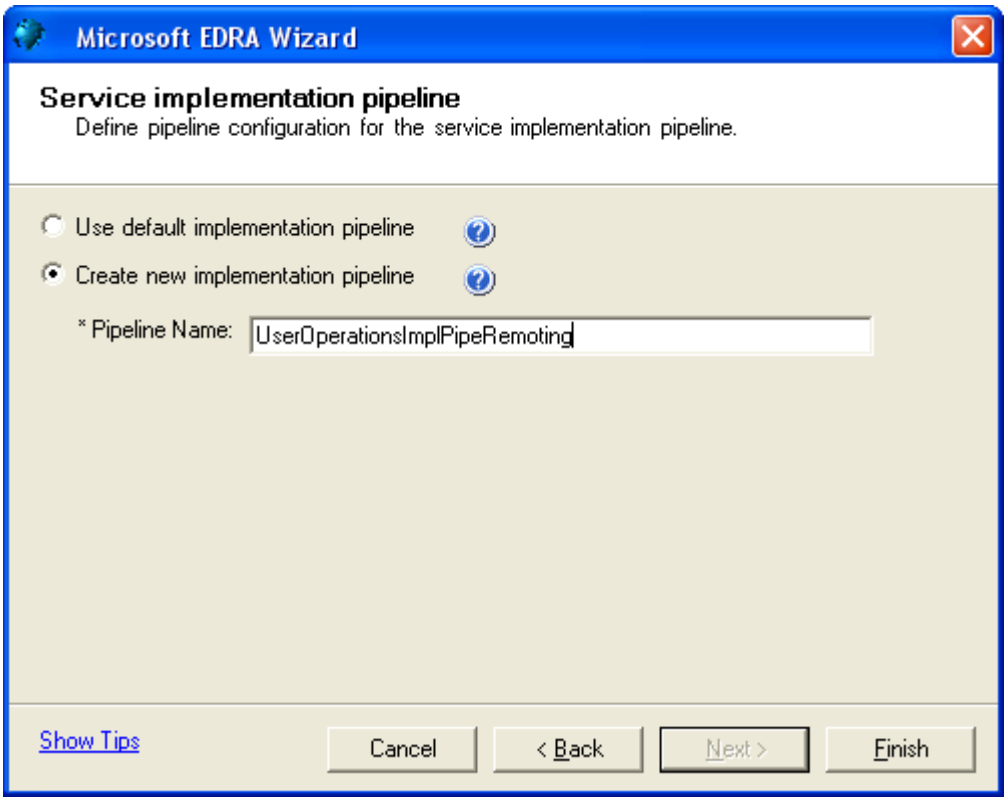


Figura 5-61 Publicar un Business Action (5)[228]

Este proceso debe ser repetido por cada uno de los transportes que se requieran para la publicación de un Business Action. Se recomienda por lo menos la utilización de dos transportes: Remoting y Web Services que son los más comunes dentro del ambiente de .net.

[228] Elaboración Propia

5.2.8. Configuración de EDAF

Para cubrir las necesidades de una aplicación, EDAF provee una serie de helpers o ayudantes de configuración. Estos deben ser registrados en el archivo ServicesReferenceArchitecture.config ubicado dentro de la carpeta Solution Items de la solución del Framework. En este archivo definimos la cadena de conexión a la base de datos que luego será utilizada por la aplicación. Para configurar esta cadena de conexión, debemos modificar el valor del atributo ConnectionString dentro de la sección frameworkHelperSettings del archivo antes mencionado.

Código 5-9 Sección FrameworkHelperSettings del Archivo de configuración

```
<helper name="ReferenceDatabaseHelper"
      type="Microsoft.ReferenceArchitecture.Services.Support.Core.
      FrameworkDatabaseHelper, Microsoft.ReferenceArchitecture.
      Services.Support.Core, Version=1.0.0.0, Culture=neutral,
      PublicKeyToken=7191858c9959a33b"
      isReusable="true">

        <configuration ConnectionString="Initial Catalog=<BasedeDatos>;
                  Data Source=<servidor>\<instancia>;
                  user id=<usuario_servidor>;
                  Password=<password>" />


</helper>
```

5.2.9. Implementación de la capa de Process UI

La capa de ProcessUI es la capa de enlace entre la aplicación cliente y los objetos de negocio como son los Business Actions. En esta capa se realiza todo el procesamiento de datos para luego, con los datos procesados, realizar el enlace por los transportes con la capa de negocios. Esta capa en el caso de las aplicaciones Windows, es la primera capa en la que los procesos y assemblies son cargados en el cliente y no en un servidor de componentes, como es el caso

de todas las capas anteriores. En el caso de Web Forms los componentes de esta capa son los que se encontrarán en el proceso de la aplicación Web, mientras que todos los componentes de capas inferiores se encontrarán en otro proceso del servidor o de otros servidores.

5.2.9.1. Transport proxies

Los proxys de transporte son los encargados del gestionamiento de las conexiones a los Business Actions de manera transparente a cualquier transporte que sea utilizado. El código fuente de las clases que manejan los proxys de transporte se encuentra en el proyecto de Presentation Layer, Reference.ProcessUI, dentro de la carpeta TransportProxy. (Ver 4.2.8.1 4.2.8.1 Transport proxies -11-)

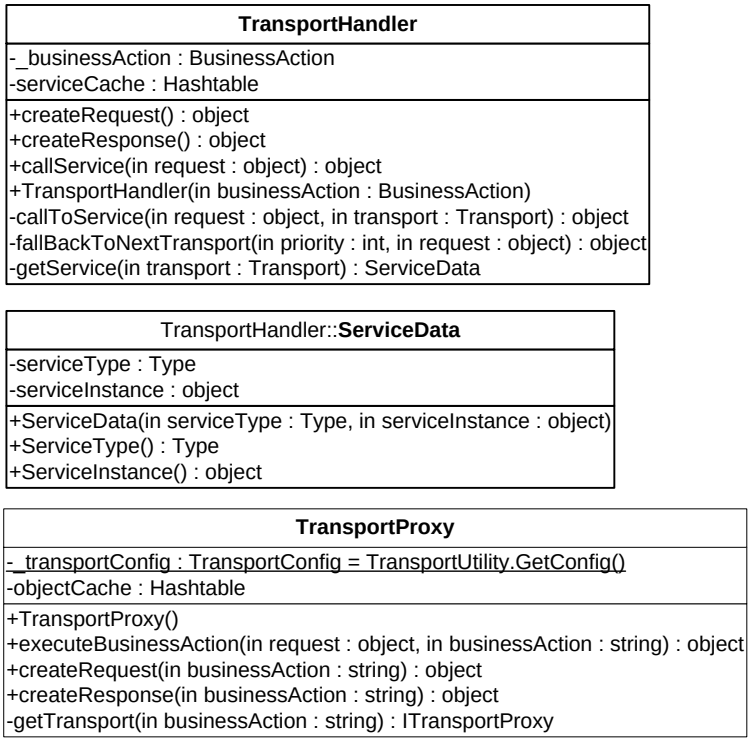


Figura 5-62: Diagrama de clases de Transport Proxy[229]

[229] Elaboración Propia

5.2.9.1.1. Creación y configuración de transportes

La configuración de los Proxys de transporte se la realiza en el archivo de configuración de la aplicación. Para el caso de las aplicaciones Windows se debe modificar el archivo app.config en el proyecto de Presentation Layer: Reference.Windows.Launcher para Windows Forms. Para las aplicaciones Web de debe modificar el archivo Web.config en el proyecto de Presentation Layer: Reference.Web en el caso de Web Forms.

La configuración necesaria es igual para los dos casos (web y windows) y se la realiza en XML, donde se especifica:

- Nombre del Business Action: El nombre con el que se va a conocer el Business Action en la aplicación.
- Tipos de datos para los mensajes de Request y Response, los tipos de datos para los mensajes, y los assemblys en los que residen. Los tipos de datos deben estar especificados de la forma “Fully Qualified Name (FQN)”, esto quiere decir que se debe especificar el nombre completo del tipo de dato, incluido su Assembly, por ejemplo: el FQN de la clase TextBox es System.Windows.Forms.TextBox.
- La clase Handler para el Business Action, el Framework da la apertura necesaria para que sean implementados nuevos proxys de transporte en caso de ser requeridos por el usuario. Ya que el Framework provee ya un Proxy de transporte es necesario el indicar

en configuración cual es la clase que será la encargada de realizar el procesamiento de los transportes.

- Por cada transporte se deberá especificar:
    - Nombre del transporte, pueden ser remoting, webService o msmq
    - La prioridad del transporte, para seleccionar el orden de procesamiento de los transportes. Esta prioridad debe ser única dentro del Business Action, y se evaluará en orden ascendente. Es decir el menor número tendrá la mayor prioridad como transporte.
    - El tipo de la clase que maneja el transporte, la interface de transporte creada por EDRA. (Ver 5.2.7.2 Publicación de Business Actions en interfaces de transporte).
    - El assembly en el que se encuentra la interface de transporte.
    - El URL del objeto remoto, definido en el wizard de publicación de un Business action. (Ver 5.2.7.2 Publicación de Business Actions en interfaces de transporte)
    - El nombre del método del BusinessAction que se debe ejecutar, definido en el wizard de publicación de un Business action. (Ver 5.2.7.2 Publicación de Business Actions en interfaces de transporte)

A continuación se encuentra la forma general de la sección de configuración de la sección de transportes:

Código 5-10 Forma General de Sección de Configuración de Transportes

```
<transports>
            <businessAction name="Nombre"
                    requestType="FQN.Request.Class"
                    requestAssembly="Messages.Assembly"
                    responseType="FQN.Response.Class"
                    responseAssembly="Messages.Assembly"
                    handlerClass="Transport.Handler.FQN">

                    <transport name="transportName"
                              priority="1"
                              type="FQN.Transport.Interface"
                              assembly="Transport.Interface.Assembly"
                              URL="URL://for/the/remote/object:port"
                              methodName="MethodName" />
            </businessAction>

</transports>
```

Dentro de la sección transports se definirán todos los Business actions que sean necesarios, y dentro de cada Business action, se podrán definir tantos transportes como el Business action tenga definidos.

A continuación se presenta un ejemplo de configuración con el Business action de User.

Código 5-11 Sección de Configuración para el Business Action de UserOperations

```
<businessAction name="CUserOperations"
       requestType="Reference.Messages.UserDataRequest"
       requestAssembly="Reference.Messages"
       responseType="Reference.Messages.UserDataResponse"
       responseAssembly="Reference.Messages"
       handlerClass="Reference.ProcessUI.TransportProxy.TransportHandler" >

       <transport name="webService"
              priority="2"
              type="Reference.ProcessUI.UserOperations.UserOperationsWebService"
              assembly="Reference.ProcessUI"
              URL="http://localhost/MillwoodWebService/UserOperationsWebService.asmx"
              methodName="UserOperations" />

       <transport name="remoting"
              priority="1"
              type="Reference.Interface.Remoting.UserOperationsRemoted"
              assembly="RemotingTransport"
              URL="tcp://localhost:9000/UserOperationsRemoted.rem"
              methodName="UserOperations" />

</businessAction>
```

De esta manera se deberá configurar todos los Business Actions con sus respectivos transportes en el archivo de configuración de la aplicación.

5.2.9.2. Operaciones

Las operaciones realizan el procesamiento de los mensajes enviados a los transportes para su posterior procesamiento por los Business actions y la extracción de datos de los mensajes de respuesta. Estos componentes son los encargados del manejo de los proxys de transporte, y de los mensajes, de manera que si se desea cambiar la implementación de los componentes de negocio definidos en capas superiores, o si se desea cambiar las capas subyacentes, modificando la implementación de los mensajes o transportes sea lo más sencillo posible, y transparente para los componentes de otras capas. (Ver 4.2.8.2 Operaciones -12-)

Todas las operaciones se encuentran en el proyecto Reference.ProcessUI dentro de Presentation Layer. Para crear una nueva operación debemos realizar las siguientes tareas:

Crear una nueva clase dentro de la carpeta Operations del proyecto.

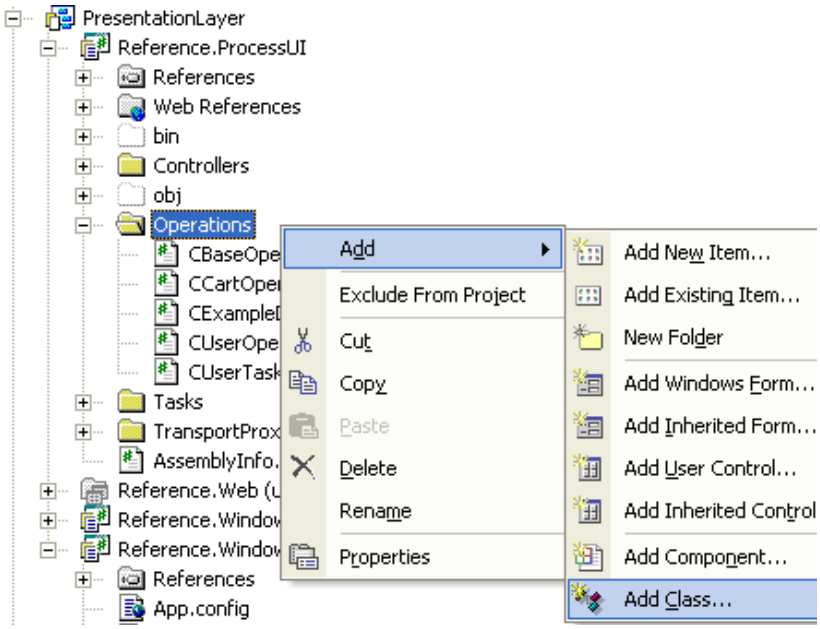


Figura 5-63 Crear una Operación[230]

La clase creada va a manejar el Business Action de User por lo que será llamada CUserOperations.

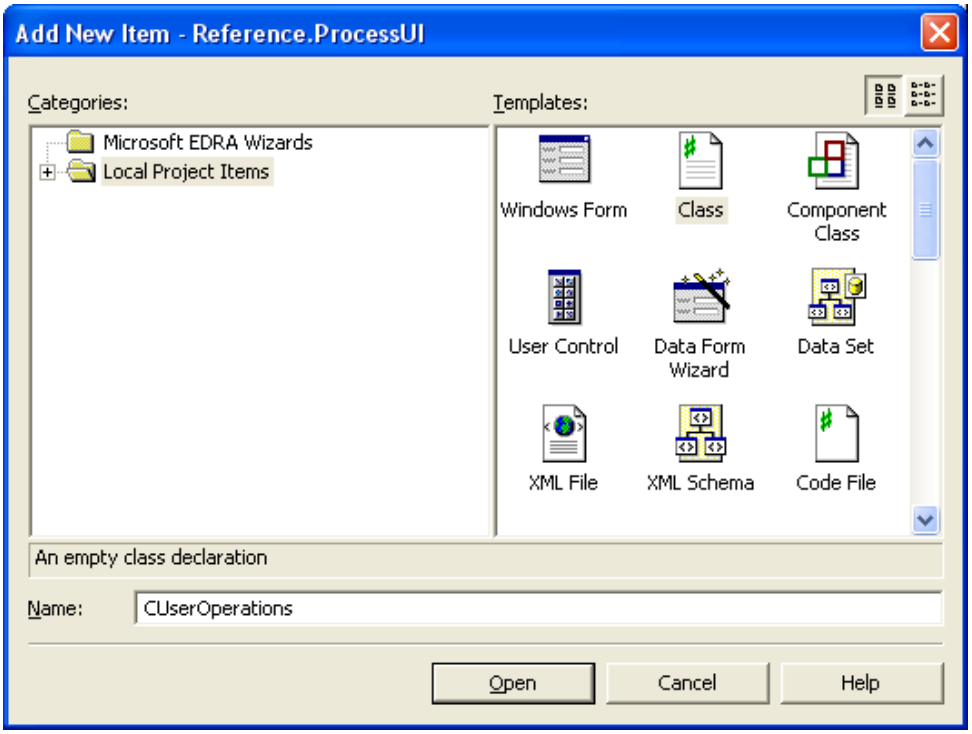


Figura 5-64 Crear una Operación (2)[231]

[230] Elaboración Propia

Luego debemos agregar las cabeceras de la clase necesarias, y heredar la clase base. En el caso de la clase de operaciones de Usuario, solo necesitaremos las entidades empresariales.

Código 5-12 Clase Operación

```
//
// CBaseOperation.cs
//
//
// Version:  1.0.0.1
//
//

using System;
using Reference.Entities;

namespace Reference.ProcessUI.Operations
{
      /// <summary>
      /// CUserOperation. Class for Operations related to User Business
      /// Action
      ///
      /// Authors: Ivan Fernandez
      ///          Juan Jalil
      ///
      /// This class contains all the functionality related to the User
      /// Business Action
      /// </summary>
      public class CUserOperations : CBaseOperation
```

Luego definiremos todas las funciones necesarias para realizar todas las tareas definidas en el Business Action (Ver 5.2.7.1.1 Implementación del Business Action). En este caso las operaciones necesarias son:

- SelectAll
- SelectById
- Nuevo
- Update
- Eliminar

[231] Elaboración Propia

- Sincronize
- DoLogin

Además debemos agregar una función que reciba el mensaje de respuesta del Business Action, y lo revise para verificación de los datos retornados. A continuación el código de una de las funciones implementadas.

Código 5-13 Creación de una Operación (3)

```
/// <summary>
/// Handles the select By Id of the Business Action
/// </summary>
/// <param name="id">UserId to search</param>
/// <returns>An Entity containing the result of the Business Action
/// process</returns>
public static Reference.Entities.UserEntity selectById(int id)
{
      Reference.Messages.UserDataRequest request=
            proxy.createRequest("CUserOperations")
            as Reference.Messages.UserDataRequest;

      request.operation="SelectById";
      request.parameters = new object[] {id};

      Reference.Messages.UserDataResponse response=
            proxy.executeBusinessAction(request,"CUserOperations")
            as Reference.Messages.UserDataResponse;

      return checkResponse(response);
}
```

De esta manera se deberá realizar la implementación de todas las funciones antes definidas. Para la aplicación completa se deberá implementar una clase de operaciones por cada uno de los Business Actions, con la misma estructura que la presentada.

El código completo de las clases de operaciones se encuentra en el CD que acompaña al texto.

El diagrama de clases de una operación es el siguiente:

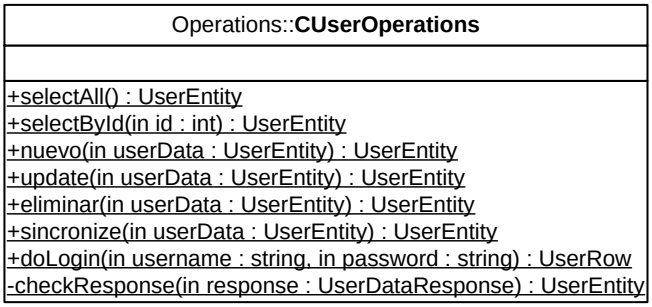


Figura 5-65: Diagrama de clases de una operación[232]

5.2.9.3. UI Process Application Block

UI Process Application Block es un componente desarrollado por Microsoft que implementa las características de UI Process. (Ver 4.2.8.3 UI process components).

5.2.9.4. Controladores de aplicación

Los controladores de la aplicación son el enlace de la aplicación con UIProcess, estos controladores son enlazados a las vistas de la aplicación, para proveer acceso desde la vista a la funcionalidad definida en la capa de negocio, los controladores deben contener todas las funciones de negocio que una vista requiere, ya que desde una vista solamente se debería llamar a métodos y funciones del controlador definido para esa vista, para que de esa manera, UI Process pueda controlar el estado, session y flujo de la aplicación. (Ver 4.2.8.3.1 Controllers -14-)

[232] Elaboración Propia

Para la implementación de un controlador debemos primero crear una clase, esta clase debe ubicarse dentro del proyecto de Reference.ProcessUI en la carpeta Controllers dentro de la capa de Presentation Layer.

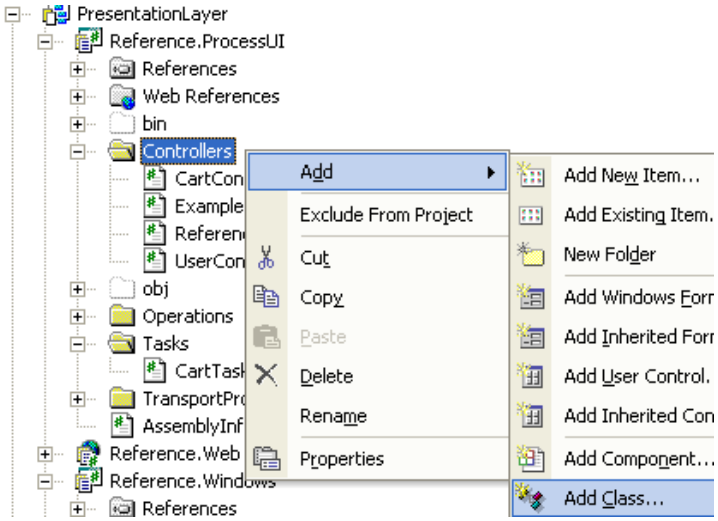


Figura 5-66 Creación de un Controlador[233]

Esta clase será la encargada de manejar las tareas referentes a un proceso de usuario específico, que en este caso será el proceso de administración de usuarios, por lo que llamaremos a esta clase UserController.

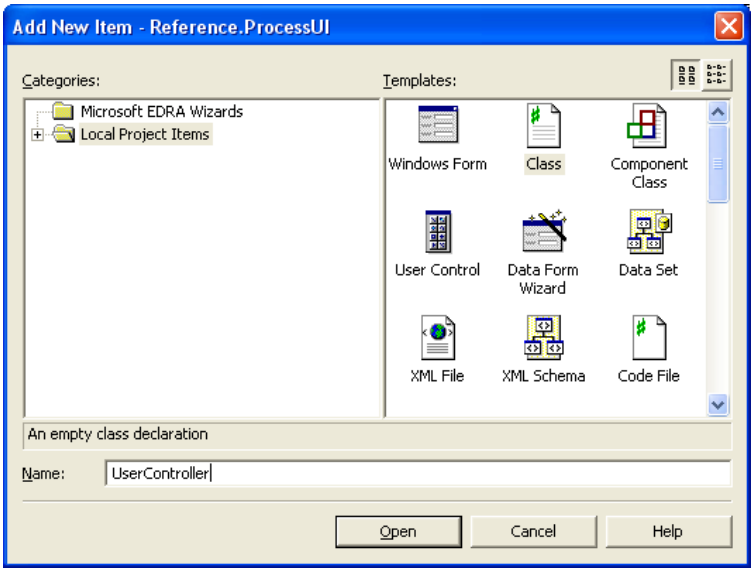


Figura 5-67 Creación de un Controlador (2)[234]

[233] Elaboración Propia
[234] Elaboración Propia

El application block de process UI requiere que todo controlador tenga un constructor con un parámetro Navigator que llame a su vez al constructor de la clase base con este parámetro, y herede de la clase BaseController de UIP. En la aplicación se ha creado un controlador base donde se podría, si existiera, agregar funcionalidad común para los controladores de toda la aplicación, este controlador se denomina ReferenceController y cumple con la regla del constructor y de la herencia impuesta por UIProcess. Para implementar nuestro controlador definimos las cabeceras necesarias, que son los namespaces de el bloque de UIProcess, las entidades empresariales y las operaciones. Además heredamos de la clase ReferenceController y creamos el constructor requerido.

Código 5-14 Clase UserController

```
//
// UserController.cs
//
//
// Version:  1.0.0.1
//
//

using System;
using Microsoft.ApplicationBlocks.UIProcess;
using Reference.Entities;
using Reference.ProcessUI.Operations;

namespace Reference.ProcessUI.Controllers
{

      /// <summary>
      /// UserController. Controller for the User Process
      ///
      /// Authors: Ivan Fernandez
      ///          Juan Jalil
      ///
      /// This class is responsible for managing all the operations of
      /// the User process for Administering Users.
      /// </summary>
      public class UserController : ReferenceController
      {
            /// <summary>
            /// Contructor required by UIP, calls the constructor of the
            /// base with the navigator passed as parameter.
            /// </summary>
            /// <param name="navigator">The navigator to wich this
```

```
            /// controller belongs to, passed by the UIP AB</param>
            public UserController(Navigator navigator) : base(navigator)

            {}
```

Luego de esto creamos todas las funciones que deseamos que sean accesibles desde la vista, que manejaran el proceso del usuario, y el flujo de datos. En este caso la función más importante es la de Login.

Código 5-15Clase UserController

```
/// <summary>
/// Verifies the login information of a user
/// </summary>
/// <param name="username">Username provided by the user</param>
/// <param name="password">Password provided by the user</param>
/// <returns>Empty row if the information is an invalid login, a Row
/// with the data of the user if the user exists</returns>
public UserEntity.UserRow doLogin(string username, string password)
{
      return CUserOperations.doLogin(username,password);
}
```

El código completo de la clase se puede encontrar en el CD que acompaña al texto.

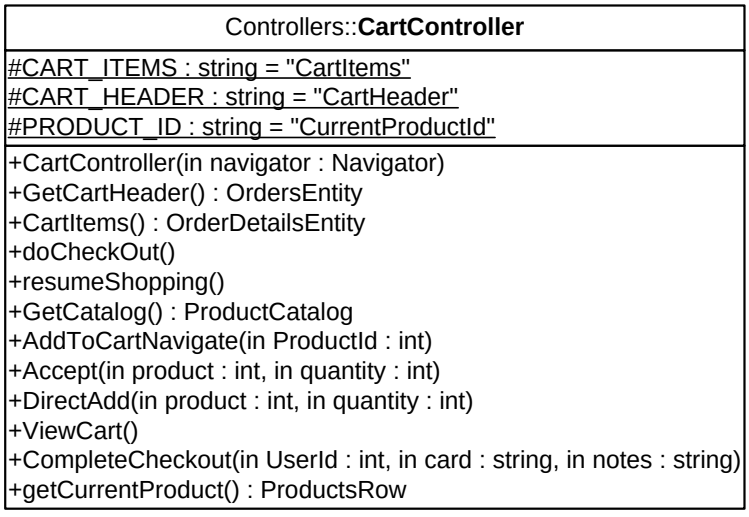


Figura 5-68: Diagrama de clases de un controlador[235]

[235] Elaboración Propia

5.2.9.5. Tareas

Las tareas son objetos que definen un proceso de un usuario. (Ver 4.2.8.3.2 Tasks -15-). Para la implementación de tareas debemos seguir los siguientes pasos:

Crear la clase que representará la tarea dentro de Presentation Layer en el proyecto de Reference.ProcessUI, dentro de la carpeta Tasks.

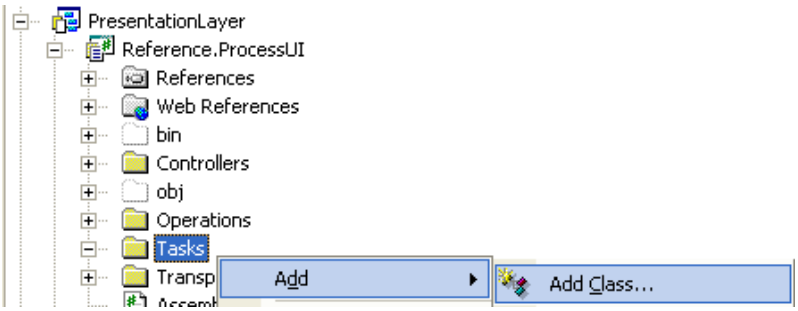


Figura 5-69 Creación de una Tarea

Todas las tareas por convención deberán llamarse de la forma ***<Nombre del Proceso>Task,*** así llamamos a esta clase CarTask.

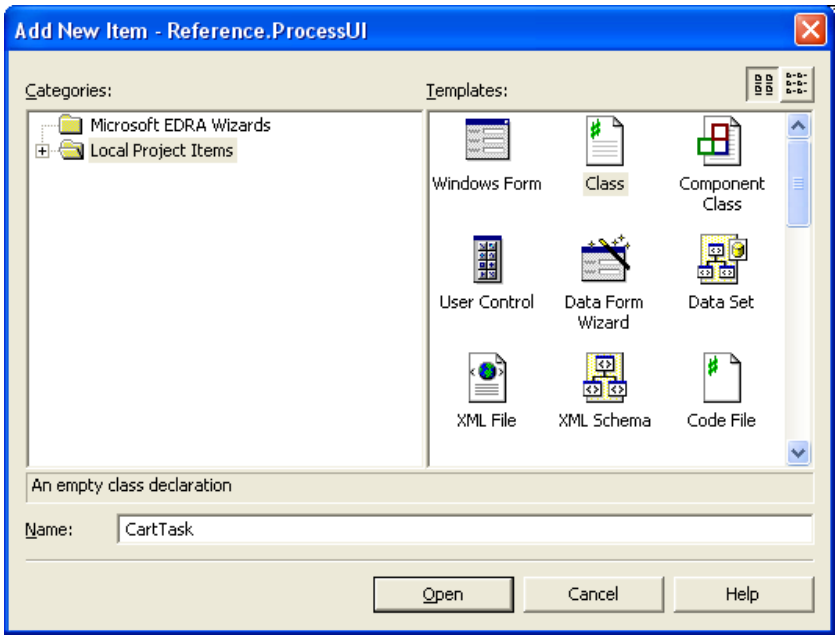


Figura 5-70 Creación de una Tarea (2)[236]

[236] Elaboración Propia

Agregamos las sentencias include necesarias, en este caso se utilizarán las Operaciones, las entidades empresariales y el Application Block UIProcess de Microsoft.

Código 5-16 Includes de la Tarea

```
using Reference.Entities;
using Microsoft.ApplicationBlocks.UIProcess;
using Reference.ProcessUI.Operations;
```

Luego de esto en la clase implementamos las interface de UIProcess ITask, con esto se crearán las funciones necesarias para la integración con UIProcess, debido a que esta clase debe realizar el enlace, entre los datos del usuario que tenemos nosotros en nuestra aplicación y los datos de la tarea asignada por Process UI, declaramos 2 variables de la clase, una para almacenar el usuario del proceso, y otra para almacenar el taskID asignado por UIProcess.

Las funciones que se deben implementar de la interface son:

- Create: Recibe de parámetro un Guid, es el taskID creado por UIProcess, se debe relacionar este taskID, con el userID entregado en la creación del task. Para en caso de ser necesario sea recuperado después para restaurar una sesión de un usuario.
- Get: Retorna un Guid, el guid almacenado en la variable de la clase.

En la constructora se recibe el userID del usuario del proceso, y se hace el proceso respectivo, para recuperarlo, en caso de existir, del almacén de datos, o asignar uno nulo, para que UIProcess lo cree en caso de ser necesario.

A continuación el código de la clase CartTask.

Código 5-17 Clase CartTask

```
public class CartTask : ITask
{
	private UserEntity.UserRow userData;
	private Guid taskId;

	/// <summary>
	/// Constructor for the task
	/// </summary>
	/// <param name="user">The userID that this task belongs
	    to</param>
	public CartTask(UserEntity.UserRow user)
	{
		userData=user;
		taskId=CUserTaskOperations.selectTaskByUserId(user.id);
	}

	#region ITask Members

	public void Create(Guid taskId)
	{
		this.taskId=taskId;
		CUserTaskOperations.createTask(userData.id,taskId);
	}

	public Guid Get()
	{
		return taskId;
	}

	#endregion
}
```

El diagrama de clases de una tarea es el siguiente:

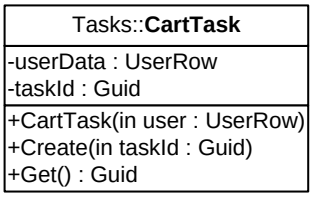


Figura 5-71: Diagrama de clase de una tarea[237]

[237] Elaboración Propia

5.2.10. Implementación de la capa de presentación

5.2.10.1. Windows Forms

Las aplicaciones Windows son implementadas utilizando los denominados Windows Forms que nos provee .net, un Windows Form no es más que una pantalla que está dentro de una aplicación que corre dentro de un sistema Windows. Para que una aplicación Windows hecha en .net funcione en cualquier PC se debe instalar en el cliente un redistribuible de .net si es que éste lo requiere aunque por lo general en la actualidad ya está instalado en la mayoría de los PC.

La aplicación Windows estará compuesta básicamente por dos proyectos: uno para la creación de los formularios Windows denominado Referece.Windows y el otro proyecto Reference.Windows.Launcher el cual contiene la clase inicializadora de la aplicación la cual se encarga de cargar el formulario inicial, contendrá también los archivos necesarios para la configuración de la aplicación y de los controladores de Process UI.

Se toma como base el proyecto denominado Reference.Windows y el otro proyecto Reference.Windows.Launcher que se encuentran en el conjunto de proyectos de la capa de presentación. (Ver 5.2.2.2.3 Proyectos de la Capa de Presentación). Dicho proyecto contendrá un formulario de ejemplo y los archivos necesarios para la configuración de la aplicación y a partir de esto implementaremos la aplicación de administración definida en la especificación de requerimientos de la aplicación prototipo. (Ver 5.1.3.1.2 Especificación de Requerimientos: Aplicación Windows).

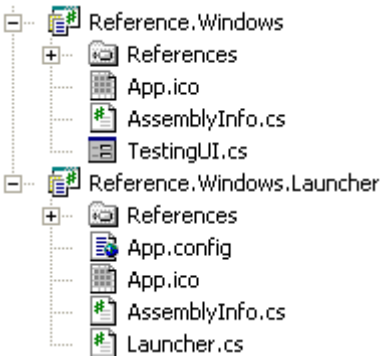


Figura 5-72: Formularios y archivos de configuración base para la aplicación de escritorio[238]

Los pasos para la implementación de la aplicación Windows son los siguientes:

5.2.10.1.1. Creación de los formularios Windows

Para crear un nuevo formulario Windows hacemos clic derecho sobre el proyecto Reference.Windows y hacemos clic en Add – Add Web Form.

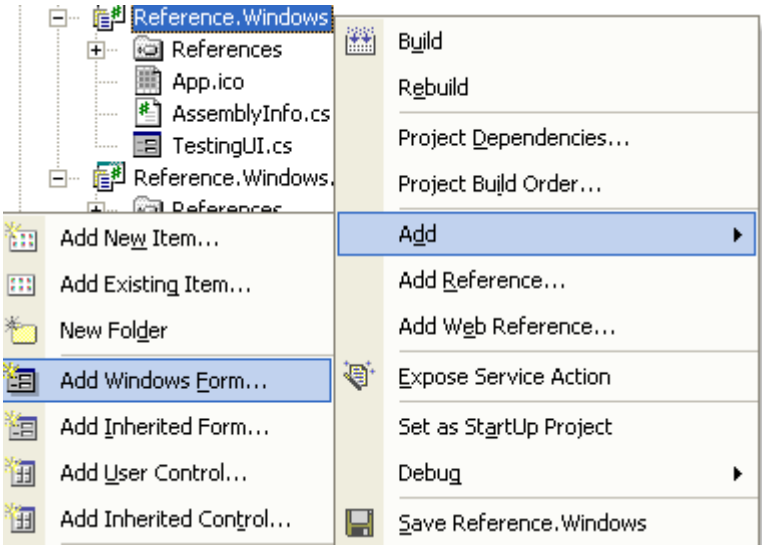


Figura 5-73: Creación de un Windows Form[239]

[238] Elaboración Propia
[239] Elaboración Propia

En la siguiente pantalla escribimos como nombre del archivo AdminProductos.cs que representará la pantalla de administración de productos.

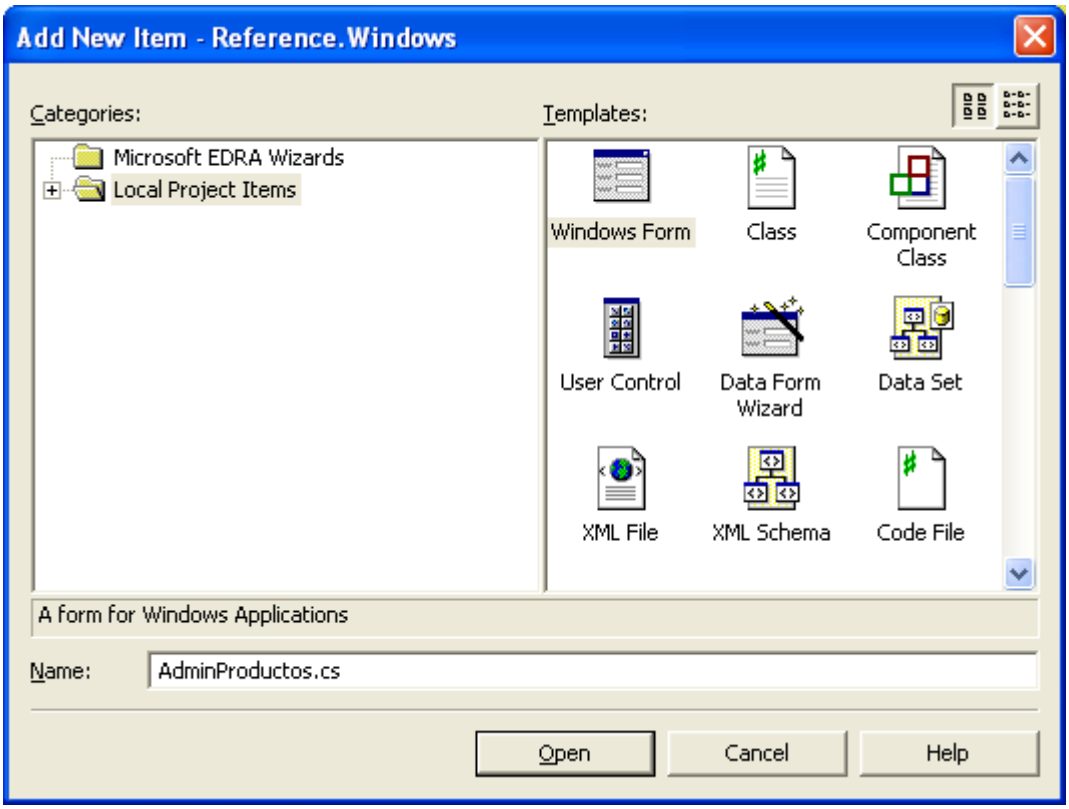


Figura 5-74 Creación de un Windows Form(2)[240]

En la pantalla creada se agregará controles para que se pueda implementar la funcionalidad de la administración de productos. El código completo de la implementación y diseño de cada pantalla se encuentra en la carpeta “aplicación Prototipo\PresentationLayer\Reference.Windows” del CD que acompaña al texto.

5.2.10.1.2. Configuración del proyecto inicializador

El proyecto Reference.Windows.Launcher contiene la clase inicializadora de la aplicación, esta clase se encarga de cargar el formulario inicial de la aplicación

[240] Elaboración Propia

e iniciar y sirve de contenedor para que el resto de formularios Windows se carguen durante la ejecución de la aplicación. Para especificar el formulario de inicio se debe abrir la clase Launcher.cs dentro del proyecto Reference.Windows.Launcher, en el método Main() se crea una variable nueva del tipo del formulario dentro de la función Application.Run() tal como se especifica a continuación:

```
public static void Main(string[] args)
            {
                  Application.ThreadException += new
System.Threading.ThreadExceptionEventHandler(Application_ThreadException);
                  Application.EnableVisualStyles();
                  Application.Run(new LoginForm());
            }
```

Otra de las funciones que brinda este proyecto es el de realizar la configuración para que pueda funcionar la capa de Process UI.

5.2.10.1.3. Configuración de Process UI dentro de la aplicación Windows

La aplicación Windows utilizará los componentes de Process UI implementados en el punto 5.2.9 Implementación de la capa de Process UI. Para que la aplicación pueda consumir y funcionar con los componentes será necesario definirlos en el archivo de configuración de la aplicación denominado app.config que se encuentra en el proyecto de Reference.Windows.Launcher. En este archivo existen varias secciones de configuración definidas mediante etiquetas XML. La sección <uipConfiguration> corresponde a todo lo relacionado con Process UI.

Configuración de los controladores de aplicación

Dentro de la sección <uipConfiguration> del archivo app.config agregamos las referencias a los controladores de Usuario (“UserController”) y del manejo del catálogo de productos (“CartController”) mediante las siguientes etiquetas:

Código 5-18: XML de configuración de controladores

```
            <controller
                name="UserController"
                type="Reference.ProcessUI.Controllers.UserController,
Reference.ProcessUI, Version=1.0.0.0,
Culture=neutral,PublicKeyToken=null" />


              <controller
                name="CartController"
                type="Reference.ProcessUI.Controllers.CartController,
Reference.ProcessUI, Version=1.0.0.0,
Culture=neutral,PublicKeyToken=null" />
```

Definición de las vistas

Los controladores utilizan las denominadas “vistas” que no son más que los formularios o pantallas Windows. Para cada vista deberá existir una entrada dentro de la sección <uipConfiguration> del archivo app.config. Agregamos las siguientes etiquetas.

Código 5-19: XML de configuración de vistas

```
 <view name="TestingUI"
      type="Reference.Windows.TestingUI, Reference.Windows,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
      controller="ExampleController" />

<view name="AdminProductos
      type="Reference.Windows.AdminProductos, Reference.Windows,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
      controller="CartController" />
```

Cada una de las etiquetas “view” relaciona al formulario Windows mediante el atributo “type” y mediante el atributo “controller” al controlador que será el manejador del formulario. Cada vista puede contener solo un controlador.

Utilización y referencia al controlador desde un formulario Windows (plumbing)

Para utilizar un controlador dentro de un formulario Web se debe realizar un proceso denominado “plumbing” colocando dentro del código fuente de los formularios Web el siguiente código:

Código 5-20: Código para UI Process Plumbing

```
#region "UIProcess Plumbing"

private NombreControlador LocalController
{
      get{ return (NombreControlador)this.Controller; }
}

#endregion
```

5.2.10.2. Web Forms

Las aplicaciones Web son implementadas en la tecnología ASP.net (ver 2.4.11 ASP.NET). La aplicación estará compuesta por un conjunto de pantallas o formularios Web denominados Web forms. Existe un Web form por cada punto funcional de la aplicación; por ejemplo existirá un Web form para ver la lista de productos, otro para ver el actual estado del carrito de compras etc.

Se toma como base el proyecto denominado Reference.Web que se encuentra en el conjunto de proyectos de la capa de presentación. (Ver 5.2.2.2.3 Proyectos de la Capa de Presentación). Dicho proyecto contendrá un formulario de ejemplo y los archivos necesarios para la configuración de la aplicación y a partir de esto implementaremos el portal de compras definido en la especificación de requerimientos de la aplicación prototipo.

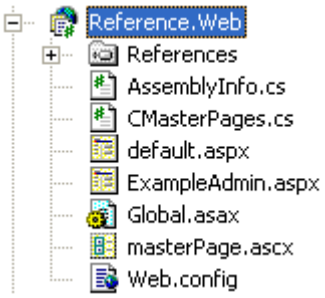


Figura 5-75: Formularios y archivos de configuración base para la aplicación Web[241]

Los pasos para la implementación de la aplicación Web son los siguientes:

5.2.10.2.1. Creación de los Web Forms

Para crear un nuevo Web Form hacemos clic derecho sobre el proyecto Reference.Web y hacemos clic en Add – Add Web Form.

[241] Elaboración Propia

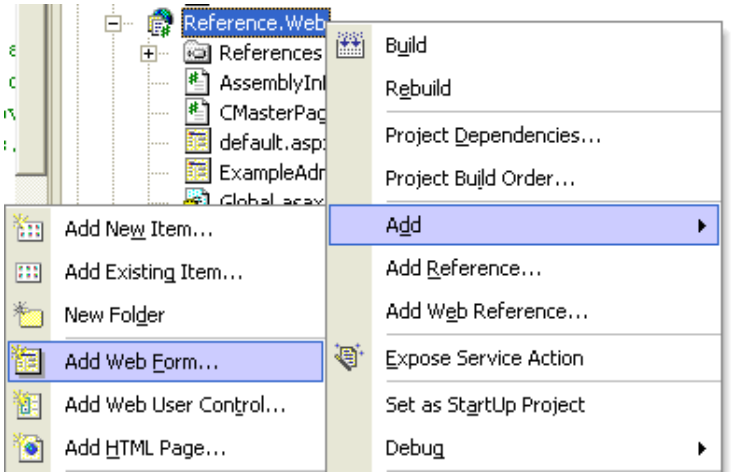


Figura 5-76: Creación de un Web Form[242]

En la siguiente pantalla escribimos como nombre del archivo catalog.aspx que representará la pantalla inicial de selección del catalogo de productos.

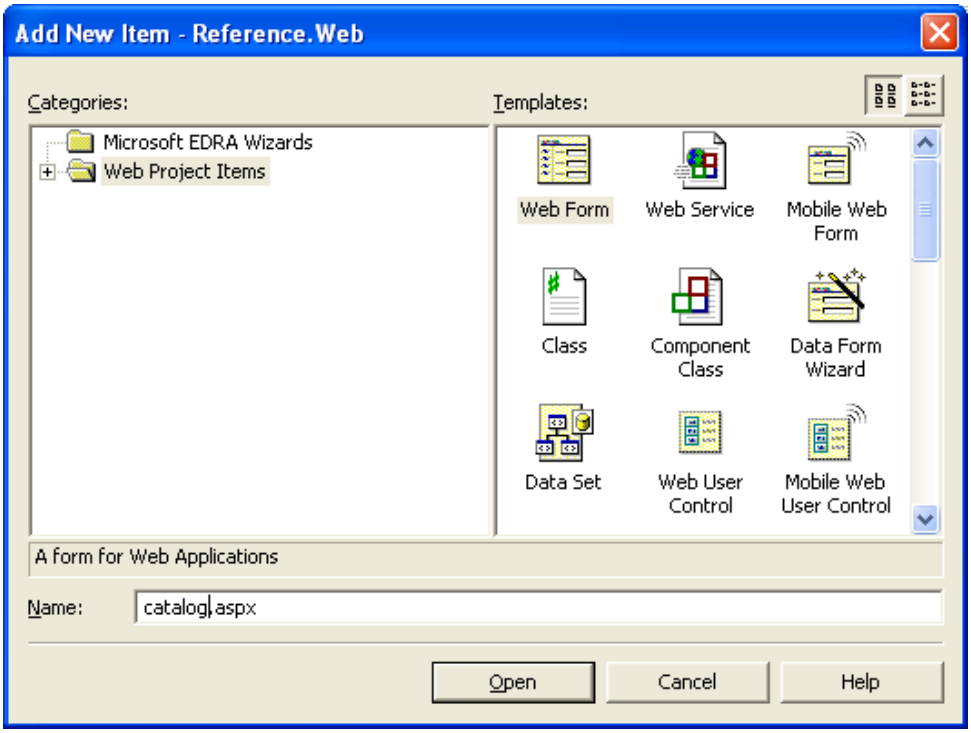


Figura 5-77: Creación de un Web Form(2)[243]

[242] Elaboración Propia
[243] Elaboración Propia

En la página creada se agregará controles para que se pueda implementar la funcionalidad del catalogo de productos. El código completo de la implementación y diseño de cada Web form se encuentra en la carpeta “Aplicacion Prototipo\PresentationLayer\ReferenceWeb” del CD que acompaña al texto.

5.2.10.2.2. Configuración de Process UI dentro de la aplicación Web

La aplicación Web utilizará los componentes de Process UI implementados en el punto 5.2.9 Implementación de la capa de Process UI. Para que la aplicación pueda consumir y funcionar con los componentes será necesario definirlos en el archivo de configuración global denominado Web.config que se encuentra en el proyecto de Reference.Web. En este archivo existen varias secciones de configuración definidas mediante etiquetas XML. La sección <uipConfiguration> corresponde a todo lo relacionado con Process UI.

Configuración de los controladores de aplicación

Dentro de la sección <uipConfiguration> del archivo web.config agregamos las referencias a los controladores de Usuario (“UserController”) y del manejo de compras y pedidos (“CartController”) mediante las siguientes etiquetas:

Código 5-21: XML de configuración de controladores

```
            <controller
                name="UserController"
                type="Reference.ProcessUI.Controllers.UserController,
Reference.ProcessUI, Version=1.0.0.0,
Culture=neutral,PublicKeyToken=null" />


              <controller
                name="CartController"
                type="Reference.ProcessUI.Controllers.CartController,
Reference.ProcessUI, Version=1.0.0.0,
Culture=neutral,PublicKeyToken=null" />
```

Definición de las vistas

Los controladores utilizan las denominadas “vistas” que no son más que los archivos aspx de los web forms. Para cada vista deberá existir una entrada dentro de la sección <uipConfiguration> del archivo web.config. Agregamos las siguientes etiquetas.

Código 5-22: XML de configuración de vistas

```
<views>
      <view
            name="TestingUI"
            type="ExampleAdmin.aspx"
            controller="ExampleController" />

      <view
            name="Catalog"
            type="catalog.aspx"
            controller="CartController" />

      <view
            name="CheckOut"
            type="CheckOut.aspx"
            controller="CartController" />

      <view
            name="AddToCart"
            type="AddToCart.aspx"
            controller="CartController" />

      <view
            name="Login"
            type="login.aspx"
            controller="UserController" />

      <view
            name="CartView"
            type="cartView.aspx"
            controller="CartController" />

      <view
            name="Congratulations"
            type="congratulations.aspx"
            controller="CartController" />


</views>
```

Cada una de las etiquetas “view” relaciona al archivo aspx mediante el atributo “type” y mediante el atributo “controller” al controlador que será el manejador del web form. Cada vista puede contener solo un controlador.

Configuración del control de navegación

El controlador de navegación nos brindará el flujo de navegación de las pantallas web. Cada pantalla puede ir a otra pantalla de forma controlada; es decir, no se puede ir deliberadamente a una pantalla que no esté definida en el flujo, por ejemplo, si estoy en la pantalla de selección de productos no puedo ir directamente a la página de confirmación de pago ya que esto no esta definido en el flujo. Para el control de flujo agregamos el siguiente código al archivo web.config.

Código 5-23: XML de configuración de controladores

```
<navigationGraph
                        iViewManager="WebFormViewManager"
                        name="Shop"
                        state="State"
                        statePersist="SqlServerPersistState"
                        startView="Catalog"
                        >

                        <node view="Catalog">
                              <navigateTo navigateValue="CheckOut"
view="CheckOut" />
                              <navigateTo navigateValue="AddToCart"
view="AddToCart" />
                              <navigateTo navigateValue="CartView"
view="CartView" />
                        </node>

                        <node view="AddToCart">
                              <navigateTo navigateValue="Resume"
view="Catalog" />
                              <navigateTo navigateValue="Add"
view="Catalog" />
                        </node>

                        <node view="CartView">
                              <navigateTo navigateValue="Resume"
view="Catalog" />
                              <navigateTo navigateValue="CheckOut"
```

```
view="CheckOut" />
                        </node>

                        <node view="CheckOut">
                              <navigateTo navigateValue="Resume"
view="Catalog" />
                              <navigateTo navigateValue="Congratulations"
view="Congratulations" />
                        </node>

                        <node view="Congratulations">
                              <navigateTo navigateValue="Resume"
view="Catalog" />
                        </node>


        </navigationGraph>
```

Utilización y referencia al controlador desde un web form (plumbing)

Para utilizar un controlador dentro de un Web Form se debe realizar un proceso denominado “plumbing” colocando dentro del código fuente de los archivos aspx el siguiente código:

Código 5-24: Código para UI Process Plumbing

```
            #region "UIProcess Plumbing"

            private NombreControlador LocalController
            {
                  get{ return (NombreControlador)this.Controller; }
            }

            #endregion
```

5.2.10.2.3. Configuración de la seguridad de la aplicación

La aplicación prototipo plantea que para ingresar al portal de compras es necesario identificar al usuario por medio de un usuario y contraseña. Para ello creamos un web form con el nombre login.aspx (Ver paso 5.2.10.2.1) agregamos dos cajas de texto y un botón. (Para el diseño completo del formulario refiérase al archivo “Prototipo\PresentationLayer\ReferenceWeb\login.aspx” del CD que

acompaña al texto. Hacemos doble clic sobre el botón “Login” y agregamos el siguiente código:

Código 5-25: Código de la función de Inicio de Sesión

```
private void cmdLogin_Click(object sender, System.EventArgs e)
            {
                  UserEntity.UserRow userData=
UserController.doLogin(txtUsername.Text,txtPassword.Text);
                  if(userData!=null)
                  {
                        FormsAuthentication.SetAuthCookie(
userData.id.ToString() , false );
                        CartTask startingTask=new CartTask(userData);

      UIPManager.StartNavigationTask("Shop",startingTask);
                  }
                                          }
```

Este código utiliza el controlador de Usuarios para realizar el proceso de validación contra la base de datos del nombre de usuario y contraseña. Si el usuario existe se autentifica al usuario y se inicia el proceso de Control de Navegación definida en el punto 5.2.10.2.2 utilizando también un Task creado en el punto 5.2.9.5. La utilización del controlador de navegación unido con el task nos brindará un proceso en el cual el usuario, una vez identificado, podrá iniciar un proceso de compra o continuar uno existente guardando toda la actividad que realice, como por ejemplo, guardar los elementos del carrito de compras con la opción de poder realizar el pedido en ese momento o en una futura ocasión sin perder lo que ya había realizado previamente.

Para continuar con la configuración de la seguridad de la aplicación se debe configurar la aplicación para que funcione con el modelo de Autenticación basada en formularios que nos provee ASP.net. Agregamos al archivo de configuración web.config las siguientes etiquetas XML:

Código 5-26: XML de configuración de seguridad

```
<authentication mode="Forms">
            <forms loginUrl="login.aspx" protection="All"></forms>
</authentication>
<authorization>
            <deny users="?" />
</authorization>
```

5.2.10.2.4. Utilización de los controladores y del mapa de navegación

Los archivos aspx utilizarán los controladores y el mapa de navegación para implementar la funcionalidad que debe proveer la capa de presentación. Un controlador provee de todo lo necesario para que el web form funcione; por ejemplo devolver los datos necesarios para recuperar una lista de productos y presentarla en pantalla o simplemente realizar la navegación hacia otra página.

Para utilizar un controlador es necesario que el web form haya referenciado al controlador mediante el proceso de "plumbing" definido en el punto 5.2.10.2.2, luego de eso simplemente se hace una llamada directa al controlador por ejemplo:

- `LocalController.ViewCart(); (Ir a la pagina del carrito de compras)`
- `LocalController.resumeShopping(); (Ir a la pagina del catalogo)`
- `LocalController.GetCatalog(); (Devuelve un DataSet con el catálogo de productos)`

5.3. Ejecución y funcionalidad de la aplicación prototipo

Una vez que la aplicación prototipo fue implementada en los puntos anteriores se va a demostrar su funcionalidad básica una vez puesta en ejecución.

5.3.1. Aplicación Windows

La aplicación windows tendrá el objetivo de realizar la administración de la aplicación prototipo. Para los detalles del alcance y los requerimientos ver 5.1.3.1.2 Especificación de Requerimientos: Aplicación Windows.

5.3.1.1. Ejecución de la aplicación Windows

La aplicación Windows puede ser ejecutada de dos formas:

- Ejecución desde el IDE de Visual Studio: esta opción se utiliza cuando la aplicación aún se encuentra en el ambiente de desarrollo. Presionamos la tecla F5 dentro del IDE de Visual Studio y se ejecuta la aplicación Web junto con la consola de transporte de los Interface Transports.
- Ejecución desde el programa compilado: esta opción se puede utilizar para cuando la aplicación ya está en un ambiente de producción y ya ha sido compilada previamente, o si se quiere abrir manualmente en el ambiente de desarrollo sin utilizar Visual Studio. Para abrir la aplicación se debe realizar dos pasos:
    - Ejecutar la consola de transporte ubicada en: "%ProgramFiles%Microsoft EDRA\CS\ReferenceArchitecture\Transports\Hosts\Console\bin\Debug"
    - Ejecutar la aplicación Reference.Windows.exe desde la carpeta de la aplicación prototipo

"C:\development\Millwood\FrameworkBase\Source\Millwood\PresentationLayer\Reference.Windows\bin\Debug".

5.3.1.2. Descripción de la funcionalidad de la aplicación windows

Una vez ejecutada la aplicación Windows podemos realizar las siguientes tareas de administración:

5.3.1.2.1. Inicio de sesión en la aplicación

La primera pantalla que podemos visualizar es la de inicio de sesión. Aquí escribimos un usuario y contraseña con permisos de administrador. La cuenta del administrador de ejemplo es: usuario: admin contraseña: millwood.

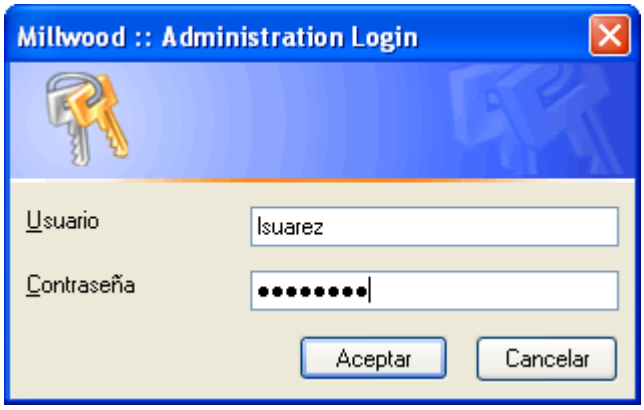


Figura 5-78: Inicio de Sesión – Aplicación Windows[244]

5.3.1.2.2. Administración de categorías de productos

Una vez ingresada a la pantalla de administración podemos ver la lista de categorías existentes, en el control de árbol de la parte izquierda.

[244] Elaboración Propia

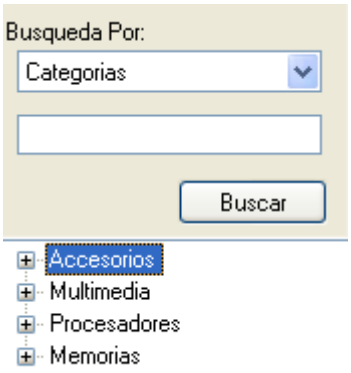


Figura 5-79: Categorías de Productos[245]

Para agregar una nueva categoría hacemos clic en el botón Nueva Categoría dentro del panel Administración de Categorías. Escribimos en el cuadro de texto el nombre de la nueva categoría y hacemos clic en el botón grabar. Para cancelar el cambio hacemos clic sobre Cancelar.

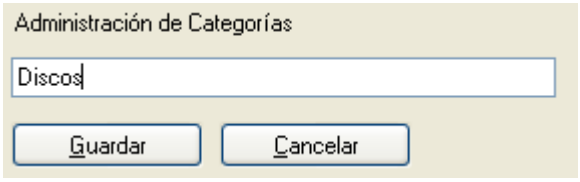


Figura 5-80: Nueva Categoría[246]

Para editar una categoría existente hacemos clic sobre el control de árbol de la parte izquierda y hacemos clic sobre el botón editar. Escribimos el nuevo nombre de la categoría y hacemos clic sobre guardar. Para cancelar el cambio hacemos clic sobre Cancelar.

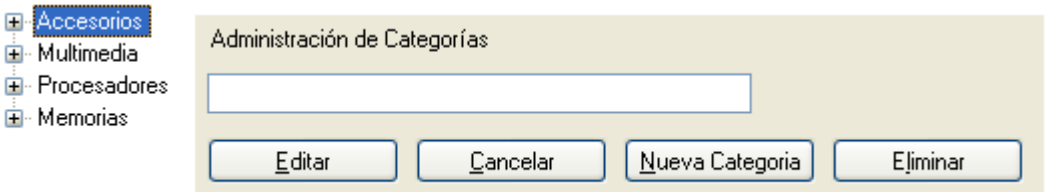


Figura 5-81: Edición de Categorías[247]

[245] Elaboración Propia
[246] Elaboración Propia
[247] Elaboración Propia

Para eliminar una categoría existente, que no tenga productos relacionados, hacemos clic sobre el control de árbol de la parte izquierda y hacemos clic en eliminar categoría.

5.3.1.2.3. Administración de productos

Una vez ingresada a la pantalla de administración podemos ver la lista de productos existentes, en el control de árbol de la parte izquierda. Para filtrar los productos por categoría seleccionamos una categoría de la lista desplegable.

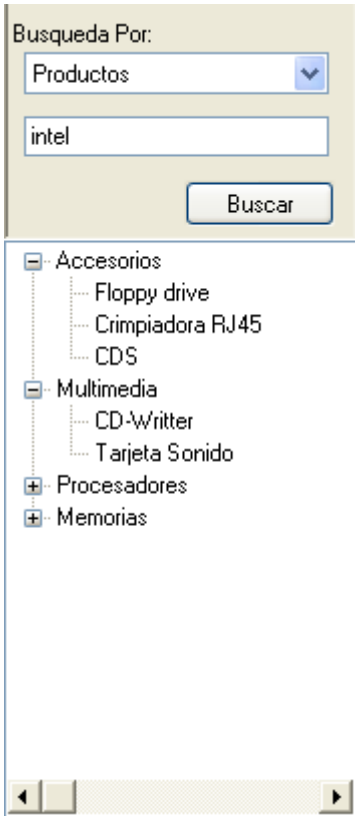


Figura 5-82: Lista de Productos[248]

Para agregar un nuevo producto hacemos clic en el botón Nuevo Producto dentro del panel Administración de Productos. Seleccionamos de la lista la

[248] Elaboración Propia

categoría, escribimos en el cuadro de texto el nombre del producto, la descripción y el precio unitario. Hacemos clic en el botón grabar. Para cancelar el cambio hacemos clic sobre Cancelar.

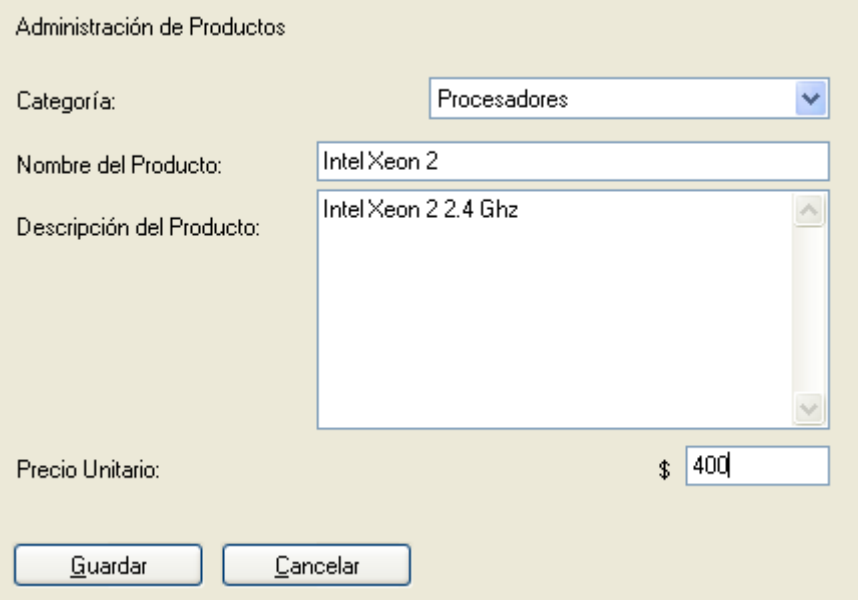


Figura 5-83: Nuevo Producto[249]

Para editar un producto existente, hacemos clic sobre el control de árbol de la parte izquierda, seleccionamos el producto a editar. Hacemos clic sobre el botón Editar dentro del panel Administración de Productos. Editamos los valores: categoría, nombre, descripción y / o precio unitario. Hacemos clic en el botón grabar. Para cancelar el cambio hacemos clic sobre Cancelar.

Para eliminar un producto, que no tenga ventas relacionadas, hacemos clic sobre el control de árbol de la parte izquierda y hacemos clic en eliminar producto.

[249] Elaboración Propia

5.3.2. Aplicación Web

La aplicación web representa la funcionalidad del catálogo virtual de compras. Para los detalles del alcance y los requerimientos.

5.3.2.1. Ejecución de la aplicación Web

La aplicación Web puede ser ejecutada de dos formas:

- Ejecución desde el IDE de Visual Studio: esta opción se utiliza cuando la aplicación aún se encuentra en el ambiente de desarrollo. Presionamos la tecla F5 dentro del IDE de Visual Studio y se ejecuta la aplicación Web junto con la consola de transporte de los Interface Transports.
- Ejecución desde un explorador de Internet: esta opción se puede utilizar para cuando la aplicación ya está en un ambiente de producción y ya ha sido compilada previamente, o si se quiere abrir manualmente en el ambiente de desarrollo sin utilizar Visual Studio. Para abrir la aplicación se debe realizar dos pasos:
    - Ejecutar la consola de transporte ubicada en: "%ProgramFiles%Microsoft EDRA\CS\ReferenceArchitecture\Transports\Hosts\Console\bin\Debug"
    - Abrir un explorador de Internet y abrir la dirección: http://<servidor>/ReferenceWeb.

5.3.2.2. Descripción de la funcionalidad de la aplicación web

Una vez ejecutada la aplicación Web podemos realizar las siguientes tareas en el catalogo virtual de compras.

5.3.2.2.1. Inicio de sesión en la aplicación

La primera pantalla que podemos visualizar es la de inicio de sesión. Aquí escribimos un usuario y contraseña, para este caso, proporcionado por algún vendedor o agente de ventas. El usuario de pruebas sería: test y la contraseña millwood.

Llenamos la información de la cuenta en las cajas de texto y presionamos el botón iniciar sesión.

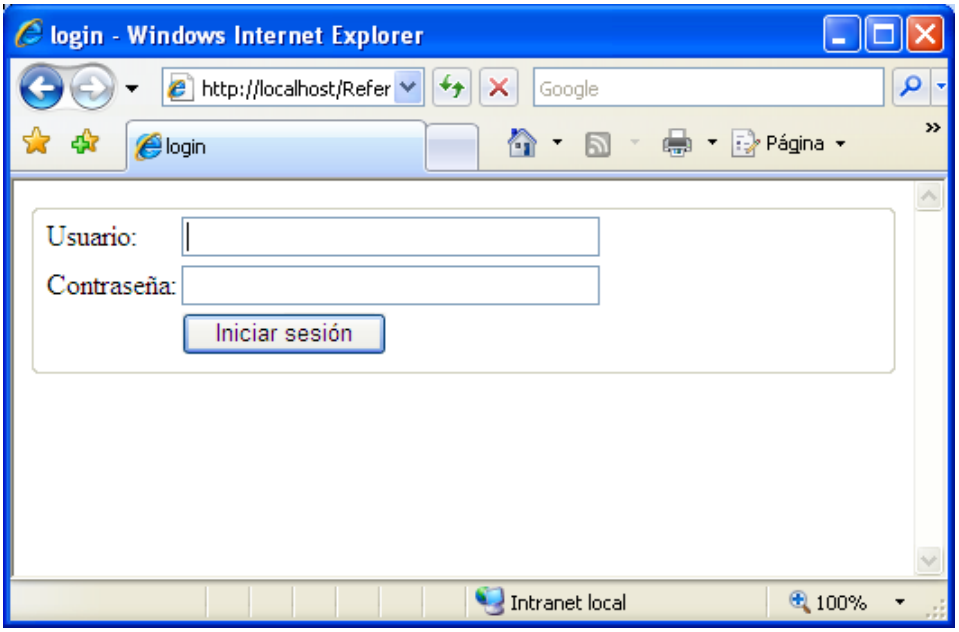


Figura 5-84: Inicio de Sesión – Aplicación Web[250]

[250] Elaboración Propia

5.3.2.2.2. Consulta de los productos del catálogo

Si es la primera vez que abrimos la tienda virtual de compras, se nos presentará la lista de productos. Los productos están agrupados o filtrados por categoría. Para ver los productos de una categoría seleccionamos de la lista desplegable la opción deseada.

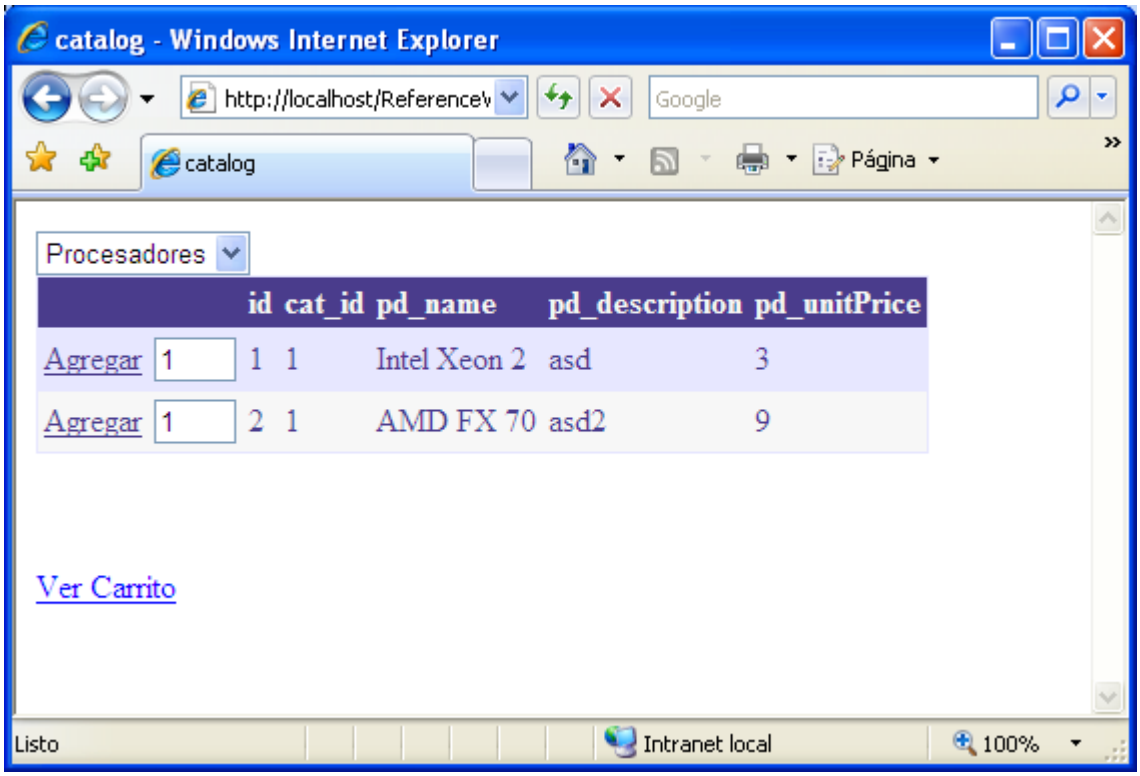


Figura 5-85: Consulta de Productos[251]

Podremos ver los productos que consten en esa categoría

[251] Elaboración Propia

5.3.2.2.3. Carrito de Compras

El carrito de compras representa una lista temporal de compras antes de realizar un pedido, este puede ser consultado y modificado a cualquier momento dentro del proceso e compras.

Añadir un producto al carrito de compras

Se puede añadir uno o varios productos del catálogo al carrito de compras; se puede además especificar la cantidad deseada de cada producto.

Para añadir un producto al carrito especificamos la cantidad deseada, mediante la caja de texto a la izquierda del producto, y presionamos el botón Agregar. Una vez hecho esta operación podremos ver los elementos del carrito de compras.

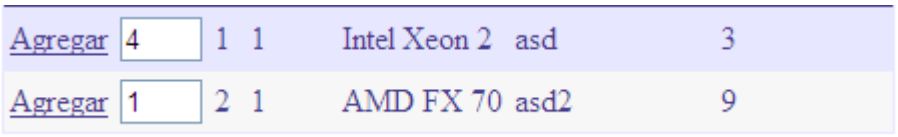


Figura 5-86: Carrito de Compras[252]

Consulta / modificación al carrito de compras

Se puede eliminar o modificar las cantidades de un producto que esté añadido al carrito de compras. Para modificar una cantidad de uno o varios

[252] Elaboración Propia

productos, usamos la caja de texto que está alado de cada producto y especificamos la nueva cantidad. Una vez modificadas las cantidades hacemos clic en el botón actualizar.

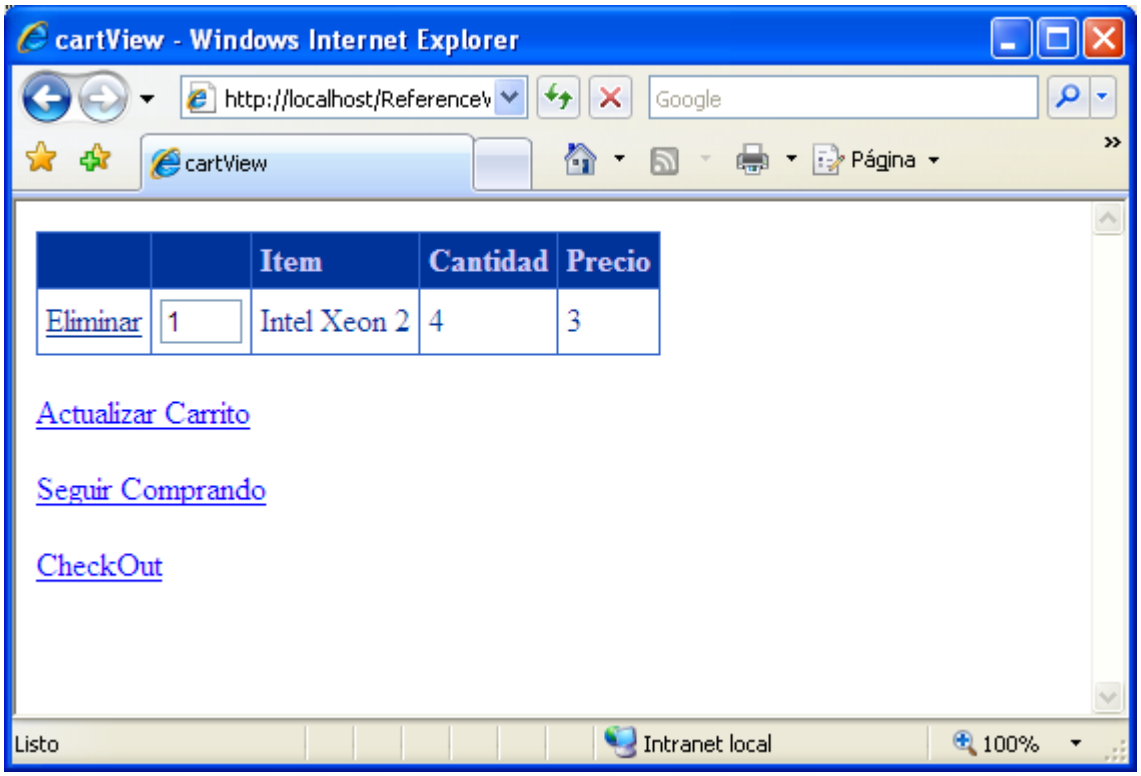


Figura 5-87: Edición del carrito de compras[253]

Para eliminar un producto del carrito hacemos clic sobre el botón eliminar ubicado al lado derecho de cada producto.

Una vez consultado o modificado el carrito de compras se puede seguir explorando el catálogo de productos y añadir nuevos elementos.

[253] Elaboración Propia

5.3.2.2.4. Confirmación del pedido

Una vez que hayamos se haya añadido los productos deseados al carrito de compras se puede realizar el pedido. Hacemos clic sobre el botón “Confirmar Pedido” y se nos presentará una pantalla de confirmación.

En la pantalla del pedido revisamos que los productos sean los elegidos, llenamos la información o medio de pago y opcionalmente las observaciones. Hacemos clic sobre el botón comprar y el pedido quedará finalizado.

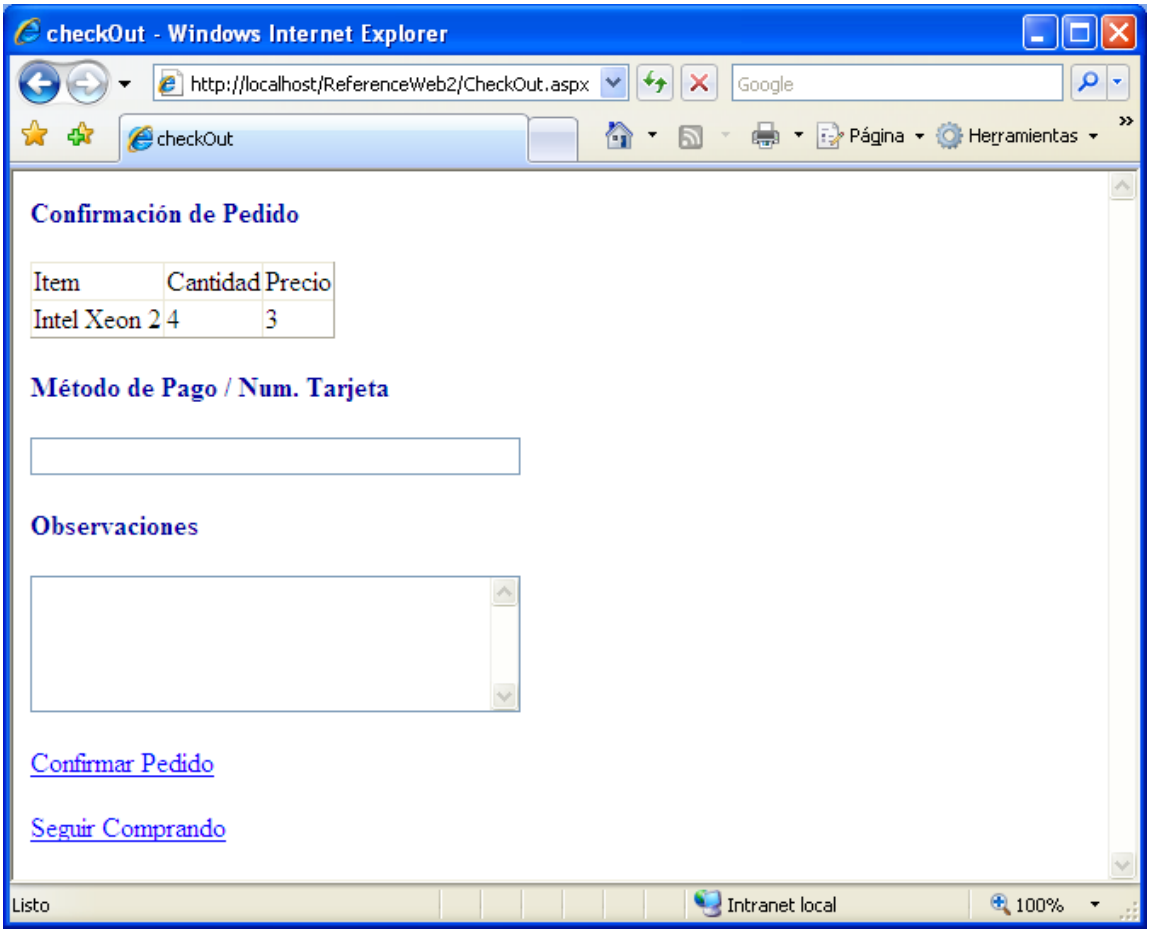


Figura 5-88: Confirmación de Pedido[254]

[254] Elaboración Propia

Podremos ver una pantalla de confirmación de que la orden ha sido registrada y será procesada por un agente de ventas.

CAPÍTULO VI

5.4. CONCLUSIONES Y RECOMENDACIONES

- El uso de los patrones de arquitectura en la construcción de una solución empresarial supone que la gente que desarrolla la aplicación obtenga muchos beneficios de calidad tanto en el producto, en el proceso y en la fase de mantenimiento. Sin embargo se debe considerar algunos puntos de importancia:
    - En referencia al ciclo de aprendizaje, a un desarrollador con una experiencia media le va a tomar mucho más tiempo en entender la forma de construir la aplicación e incluso, una vez entendido el problema, tardará mucho tiempo hasta que esta persona adquiera las destrezas para realizar la tarea de la construcción en un tiempo razonable.
    - En las fases de mantenimiento, si se cuenta con las mismas personas que diseñaron y construyeron la aplicación, se va a ver una real ventaja de haber aplicado los patrones, ya que los cambios para incorporar un nuevo requisito son mínimos.

- Durante el estudio se pudo notar que los patrones de software ayudan a mejorar la administración de la memoria, la reutilización de código y otros factores que ayudan a la calidad del software; sin embargo hay que notar que mientras más complejo sea el modelo, se incorporen más capas y se implementen más patrones, el rendimiento de la

aplicación también se degrada ya que el procesamiento es mayor, los componentes deben pasar por más capas, lo que demanda más tiempo y recursos del servidor. Un buen diseño y arquitectura de una solución empresarial supone que no se debe utilizar todos los patrones posibles sino más bien los necesarios para cumplir con las expectativas de la aplicación y que el rendimiento sea óptimo.

- Existen patrones de diseño cuya implementación sola requiere un gran trabajo en diseño y análisis. Patrones que son realmente confusos o difíciles de encontrar una aplicación concreta en un sistema. Estos patrones como por ejemplo Flyweight o Interpreter, pueden ser muy utilizables en cierto dominio de aplicaciones y en otros simplemente ser inadecuados. No todos los patrones son útiles en el mismo tipo de aplicaciones ni aplicables tampoco y no todos son posibles de utilizar todo el tiempo. Debe existir un diseño y un análisis de por qué utilizar un patrón en una situación determinada o de que patrón es el más adecuado.

- El framework propuesto brinda a una organización de desarrollo de software todos los elementos para la construcción de una solución empresarial. Sin embargo, se debe analizar primero los requisitos de dicha solución y ver si es factible de implementar sobre este framework. Una aplicación empresarial muy pequeña no justificaría que se implemente con esta solución ya que el modelo no es lo

suficientemente complejo para que justifique dicha implementación.

- Una aplicación mediana o grande va a poder obtener todas las ventajas que ofrece el framework y va a ayudar a que el modelo se implemente correctamente. Esto no solamente a nivel del sistema completo si no también en el desarrollo de un componente utilizando patrones de diseño en donde su construcción resulta más complicada si el desarrollador no está acostumbrado a trabajar de esta manera. Esto se debe a que en un componente que normalmente sería compuesto por una sola clase “monolítica” en la que se tiene todo el código asociado al componente, utilizando un patrón de diseño como Bridge, terminaría compuesto de por lo menos dos clases y una interfase o posiblemente muchos más componentes.

  El aumento en la dificultad de desarrollo de un componente debido al aumento de clases, interfaces o tiempo de diseño y análisis se ve justificado con un componente mucho más flexible y una aplicación mucho más adaptable a un nuevo componente, con un efecto mínimo en las capas superiores de la aplicación que han sido desarrolladas de acuerdo a esa interfaces. Obviamente esto sucede siempre y cuando la interface del componente no cambie, y aún así es en parte solucionable utilizando otros patrones de diseño como por ejemplo Adaptadores o Decoradores.

- Si bien el framework trae un conjunto de componentes y capas pre-establecidas, queda libre para el arquitecto o desarrollador que implemente la solución que pueda incorporar o modificar el framework para el caso y la necesidad específica de dicha organización o de la aplicación a construir. Estos cambios suponen que no se deba alterar la calidad del producto y la facilidad de mantenerlo.

- El orden en la codificación y un estándar de nomenclatura son dos partes que se considerarían básicas en un proyecto de desarrollo, especialmente en proyectos grandes un código que es bastante extenso es mucho más simple de comprender y estudiar, incluso de modificar y mantener si está pegado a un estándar de codificación y de nomenclatura. Por ejemplo, si nos referimos siempre a las interfaces con la misma nomenclatura es posible el comprender un código que utilice interfaces en un menor tiempo, pues al ver el código nos damos cuenta de inmediato de lo que se trata, al contrario de código que no utilice ningún tipo de estándar en el que deberemos mirar las definiciones o ver las referencias para darnos cuenta que una variable hace referencia a una interfase y no a una clase concreta.

- El desarrollo de sistemas utilizando patrones de software resulta, en general, en un código más compacto en las clases del sistema, sin embargo estas clases que resultan ser más pequeñas que clases normales en las que se codificaría todo en un mismo lugar tienen un

mayor número de dependencias, clases asociadas, interfaces, o componentes utilizados. Por lo que para entender el código por completo es necesario revisar un mayor número de componentes.

Al desarrollar componentes de un sistema con patrones de diseño se deberá realizar un análisis mucho más estricto del componente, su estructura y su funcionalidad, pues si no se lo hace de una manera correcta es muy posible que la implementación de este componente resulte mucho más complicada de lo que debería ser debido a errores de diseño, como se dice en el libro “Design Patterns: Elements of Reusable Object-Oriented Software”:

> *“Reusable and flexible design is difficult if not impossible to get "right" the first time.”*

El trabajo involucrado en el diseño y desarrollo de componentes no siempre es justificado pues en un proyecto de desarrollo pequeño el esfuerzo del análisis de un solo componente de esta manera podría ser demasiado alto.

Anexos

Anexo A: Estándar de codificación

Anexo B: Descripción de Casos de Uso

Glosario

A

ADO.NET (Active Data Object .Net)

Constituye el modelo primario relacional de acceso a datos de las aplicaciones basadas en .net. Brinda ambientes conectados y desconectados hacia las fuentes de datos.

Algoritmo

Conjunto de sentencias / instrucciones en lenguaje nativo, los cuales expresan la lógica de un programa.

ASP (Active Server Pages)

Tecnología desarrollada por Microsoft para sus servidores de páginas Web. Permite crear dinámicamente páginas Web mediante HTML, scripts, y componentes de servidor ActiveX reutilizables.

ASP.NET

Active Server Pages (ASP) es una tecnología del lado servidor de Microsoft para páginas web generadas dinámicamente que ofrece varias ventajas importantes de seguridad, escalabilidad y rendimiento acerca de los modelos de programación ASP anteriores.

Assembly

Un ensamblado o assembly, consiste en un conjunto de clases reunidos para formar la unidad más elemental de código que puede ejecutar el entorno de .NET Framework.

.

B

Base de datos

Es la colección de información, que está organizada de forma tal que su contenido sea fácilmente accesado, administrado y actualizado. Los tipos más comunes de Base de Datos son las "relacionales" donde la información está definida de una forma en que pueda reorganizarse y accesarse de múltiples formas distintas.

BLOB (Binary Large Object)

El formato de datos binarios almacenados en una base de datos relacional. Representa un tipo de datos que puede almacenar hasta 4 GB, generalmente utilizado para almacenar información digital como imágenes, audio y video.

C

Caché

Es un conjunto de datos duplicados de otros originales, con la propiedad de que los datos originales son costosos de acceder, normalmente en tiempo, respecto a la copia en el caché. Cuando se accede por primera vez a un dato, se hace una copia en el caché; los accesos siguientes se realizan a dicha copia, haciendo que el tiempo de acceso aparente al dato sea menor.

C#

Lenguaje de programación orientado a objetos, evolución del lenguaje C++, desarrollado por Microsoft para la tecnología .Net.

CGI (Common Gateway Interface)

Es una importante tecnología de la World Wide Web que permite a un cliente (explorador web) solicitar datos de un programa ejecutado en un servidor web. CGI especifica un estándar para transferir datos entre el cliente y el programa.

Clave / Clave Primaria

En el mundo de las bases de datos, se conoce como clave al valor de es capaz de distinguir un registro de otro de forma fiable, como podría ser la CI o el pasaporte para el caso de una persona.

CLOB (Character Large Object)

Archivo o conjunto grande de caracteres almacenados en una base de datos.

CLR (Common Language Runtime)

El CLR es uno de los principales componentes del .NET Framework. Provee el ambiente de ejecución del código y varios servicios para las aplicaciones.

CLS (Common Language Specification)

El Common Language Specification pretende dar los lineamientos y reglas que deben cumplir todos los lenguajes de .net para lograr interoperabilidad.

Cobol / Cobol.net (Common Business-Oriented Language)

Lenguaje de programación de tercera generación, desarrollado en los años cincuenta y sesenta, muy empleado para aplicaciones comerciales y financieras diseñadas para grandes ordenadores. Cobol .net constituye otro de los lenguajes del .net framework.

Código fuente

Programa en su forma original, tal y como fue escrito por el programador, el código fuente no es ejecutable directamente por el computador, debe convertirse en lenguaje de maquina mediante compiladores, ensambladores o intérpretes.

CTS (Common Type System)

El sistema común de tipos de datos de .net framework que provee los tipos de datos necesarios, valores y tipos de objetos, los mismos que son necesarios para el desarrollo de aplicaciones en diferentes lenguajes.

Clase

Es un tipo de datos definido por el usuario que especifica un conjunto de objetos que comparten las mismas características.

COM (Component Object Model)

Es un tipo de datos definido por el usuario que especifica un conjunto de objetos que comparten las mismas características.

D

DALC (Data access logic components)

Son las clases encargadas de realizar el mapeo Objeto Relacional de la base de datos hacia un modelo de objetos.

DataSet

Son estructuras de datos del ADO.Net que permiten almacenar las consultas recibidas de una fuente de datos, guardando la estructura y tipos de datos de la tabla o conjunto de registros de dicha fuente de datos.

DCOM (Distributed Component Object Model)

Modelo de objetos componentes distribuido. Extensiones del Modelo de objetos componentes (COM) que facilita la distribución transparente de objetos a través de redes y de Internet. DCOM forma parte de la especificación administrada por The Open Group para la distribución en plataformas heterogéneas.

Delphi / Delphi.net

Delphi es un lenguaje de programación y un entorno de desarrollo rápido de software diseñado para la programación de propósito general con énfasis en la programación visual. Es producido comercialmente por la empresa estadounidense Borland Software Corporation. En sus diferentes variantes, permite producir ejecutables binarios para Windows y Linux; y también para la plataforma .NET de Microsoft.

DLL (Dynamic Linking Library)

Archivos con código ejecutable que se cargan bajo demanda del programa por parte del sistema operativo. Contiene funciones que son ser utilizadas desde los programas. Es un tipo de fichero muy frecuente en Windows.

E

EDAF (Enterprise Development Application Framework)

Es un framework de desarrollo de aplicaciones distribuidas empresariales que forma parte de EDRA.

EDRA (Enterprise Development Reference Architecture)

EDRA es una guía Arquitectural que una organización puede utilizar para estandarizar el desarrollo de aplicaciones distribuidas. Es un producto soportado por Microsoft y la comunidad de Patterns & Practices.

Encapsular

En programación orientada a objetos, se denomina encapsulación al ocultamiento del estado, es decir, de los datos miembro, de un objeto de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

Encriptación

Procedimiento mediante el cual se hace ilegible un mensaje para evitar que lo lean personas no autorizadas.

Ensamblador / Assembler

Lenguaje de programación bajo nivel, muy cercano al código máquina. Su sintaxis depende por completo del tipo de ordenador que se esté usando.

F

Framework

Un Framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

G

GAC (Global Assembly Cache)

Es un repositorio centralizado de assemblies utilizado por el CLR de .net framework. El gac representa una librería compartida de clases que evita los conflictos de los DLL.

GOF (Gang of four)

Compuesta por Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, considerados como los padres de los Patrones de Software. Publicaron el libro “Design Patterns: Elements of Reusable Object-Oriented Software”

H

Herencia

Concepto de orientación a objetos que supone crear clases derivadas de otras existentes, que heredan sus tipos y métodos y pueden contener otros nuevos.

Hibernate

Solución de software libre en JAVA para el mapeo Objeto-Relacional de una base de datos hacia un modelo de objetos.

HTML (Hyper Text Markup Language)

Es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.

I

IDE (Integrated development environment)

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes.

IIS (Internet Information Services)

Servidor Web de Microsoft que corre sobre plataformas Windows. Los servicios que ofrece son: FTP, SMTP, NNTP y HTTP/HTTPS.

IL (Intermediate Language)

Constituye un parte fundamental del CLR de .net framework. Es un lenguaje intermedio entre el código fuente y el código de maquina en el cual el código fuente escrito en cualquier lenguaje de .net siempre es traducido a código intermedio para ser interpretado por el CLR.

Interface

Conexión entre dos componentes de hardware, entre dos aplicaciones o entre un usuario y una aplicación.

Interoperabilidad

La interoperabilidad es la condición mediante la cual sistemas heterogéneos pueden intercambiar procesos o datos.

Intranet

Se puede considerar como un Internet privado que funciona dentro de una organización. Normalmente, dicha red local tiene como base el protocolo TCP/IP de Internet y utiliza un sistema firewall (cortafuegos) que no permite acceder a la misma desde el exterior.

ISAM (Indexed Sequential Access Method)

Es un método para controlar la forma en que una computadora accede a registros y archivos almacenados en un disco duro. Un sistema ISAM provee acceso directo a datos específicos a través de un índice, lo cual resulta es un acceso veloz a los datos, independientemente de si los datos se leen secuencial o aleatoriamente.

IT (Information Technologies)

Tecnologías de la Información. Una forma de denominar al conjunto de herramientas, habitualmente de naturaleza electrónica, utilizadas para la recolección, almacenamiento, tratamiento, difusión y transmisión de la información.

.

J

JAVA

Java es una plataforma de software desarrollada por Sun Microsystems, de tal manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales.

JIT Compiler (Just in time)

Es un compilador que a partir del lenguaje Intermedio IL de .net genera el código máquina real que se ejecuta en la plataforma que tenga la computadora, consiguiendo cierta independencia de plataforma ya que cada plataforma puede tener su compilador JIT y crear su propio código máquina a partir del código IL.

JSP (Java Server Pages)

Es la tecnología para generar páginas web de forma dinámica en el servidor, desarrollado por Sun Microsystems, basado en scripts que utilizan una variante del lenguaje java.

L

Layer

Los layer o capas son utilizadas para dividir en partes más pequeñas un software o sistema complejo. Cada capa en encapsula su propia funcionalidad y cuándo un capa cambia las otras también lo harán en cascada.

Lenguaje de programación

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático.

Lógica de Negocio

Es utilizado en la arquitectura de un sistema como un componente o capa que implementa las reglas de un negocio manipulando los datos que vienen de la capa de vista o de almacenamiento.

M

Mensajes

Mensaje en el sentido más general, es el objeto de la comunicación. Está definido como la información que el emisor envía al receptor a través de un canal

determinado o medio de comunicación. En el caso de el Framework son los objetos que se transfieren entre los distintos componentes.

Metadatos

Metadatos (Meta+datos) es un término que se refiere a datos sobre los propios datos.

MSMQ (Microsoft Message Queuing)

Es una tecnología y protocolo desarrollada por Microsoft implementada en un servidor Windows que permite el paso de mensajes mediante aplicaciones.

MVC (Model View Controller)

Es una arquitectura de software que separa el modelo de datos, la interfaz de usuario y la lógica de control en tres distintas partes para lograr independencia entre cada una de las partes.

MySQL

MySQL es una de las bases de datos más populares desarrolladas bajo la filosofía de código abierto.

N

.net

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones.

.net framework

El .net framework es una infraestructura que reúne un conjunto de lenguajes y servicios, librerías de clases y un ambiente común de ejecución altamente

distribuido y constituye la plataforma y elemento principal sobre el que se fundamenta la tecnología .NET.

Namespace

Un namespace es la organización lógica y jerárquica de las clases dentro de .net. Una clase pertenece a un namespace.

O

Objeto

Un objeto es una representación detallada, concreta y particular de un "algo". Tal representación determina su identidad, su estado y su comportamiento particular en un momento dado.

Oracle

Oracle es un sistema de administración de base de creado por Oracle Corporation. Un servidor Oracle posee una base de datos Oracle y la instancia Oracle y soporta acceso por SQL y por lenguajes de programación.

P

Patrón

Canon o medida de referencia de las cosas.

Patrón de diseño

Los Patrones de Diseño (Design Patterns) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el

problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).

PEAA (Patterns of Enterprise Application Architecture)

Grupo de patrones orientados a la arquitectura de aplicaciones empresariales descritos por Martin Fowler

Perl / Perl .net (Practical Extraction and Report Language)

Un lenguaje script desarrollado por Larry Wall utilizado principalmente para el desarrollo de programas CGI. Perl .net es otro lenguaje para la plataforma .net.

PHP (Hypertext Pre Processor)

Lenguaje de programación tipo script para entornos Web utilizado sobre todo en servidores Linux con el fin de personalizar la información que se envía a los usuarios que acceden a un sitio web. Es un programa de software libre con unas funciones muy semejantes a las de ASP y JSP.

Polimorfismo

Es un concepto de orientación a objetos que permite al programador generar componentes reutilizables de alto nivel que puedan adaptarse a diferentes aplicaciones mediante el cambio de sus partes de bajo nivel.

Portabilidad

Característica de ciertos programas que les permite ser utilizados en distintos ordenadores sin que precisen modificaciones de importancia.

POSA (Pattern Oriented Software Architecure)

Libro de referencia de patrones de arquitectura escrito por Frank Buschmann.

Protocolo

Descripción técnica de una estándar, incluyendo las reglas de diseño y funcionamiento. Descripción formal de formatos de mensaje y de reglas que dos ordenadores deben seguir para intercambiar dichos mensajes.

Prototipo

Un prototipo es un ejemplar original o primer molde en que se fabrica una figura u otra cosa. Un prototipo puede ser un ejemplar perfecto y un modelo de referencia. Un prototipo también se puede referir a cualquier tipo de máquina en pruebas, o un objeto diseñado para una demostración de cualquier tipo.

Proxy

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

R

Remoting

Es una arquitectura de Microsoft diseñada para simplificar la comunicación entre dos objetos que se encuentran ejecutando en diferentes dominios de aplicación.

S

Serialización / Deserialización

Es el procedimiento de codificar una estructura de datos en una secuencia de bytes, el proceso inverso se lo llama deserialización.

Servlet

Los servlets son objetos que corren dentro del contexto de un servidor de aplicaciones JAVA (ej. Tomcat) y extienden su funcionalidad.

SOA (Service-Oriented Architecture)

Arquitectura Orientada a Servicios. Es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

SOAP (Simple Object Access Protocol)

Es un protocolo estándar creado por el W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios Web.

SQL (Structured Query Language)

Lenguaje de consulta de bases de datos relacionales desarrollado por IBM.

SQL Server

Microsoft SQL Server es un servidor de gestión y administración de bases de datos relacionales basada en el lenguaje SQL, que incluye también un potente entorno gráfico de administración.

Struts

Framework abierto para la construcción de aplicaciones web en JAVA.

T

TCP (Transmission Control Protocol)

Conjunto básico de protocolos de comunicación de redes, popularizado por Internet, que permiten la transmisión de información en redes de computadoras.

U

UI (User Interface)

Representa la comunicación entre una persona y un sistema de de software o sistema operativo.

UML (Unified Modelling Language)

Es el lenguaje de modelado de sistemas de software más conocido en la actualidad; aún cuando todavía no es un estándar oficial.

UIPAB (User Interface Process Application Block)

El User Process Application Block prove un framework extensible para simplificar el proceso de separar el código de lógica de negocio de la interfase de usuario.

V

Visual Basic .net

Es un lenguaje de programación orientado a objetos creado con la plataforma .net de Microsoft como evolución del lenguaje Visual Basic.

Visual Studio .net

IDE desarrollado por Microsoft en el año 2002 como herramienta para la construcción de aplicaciones en la tecnología .net. Tiene gran soporte para XML y para la construcción de aplicaciones y servicios web.

VSAM (Virtual Storage Access Method)

Es un sistema de administración de archivos utilizados en mainframes de IBM. VSAM agiliza el acceso a los datos en archivos al usar un índice invertido (conocido como B+tree) de todos los registros añadidos a cada archivo.

W

Web Service

Es una forma de servicio parte de la infraestructura SOA que contiene un conjunto de tecnologías que proveen protocolos y estándares independientes de la plataforma para el intercambio de datos entre aplicaciones.

WYSIWYG (What You See Is What You Get)

.Se aplica a los procesadores de texto y otros editores de texto con formato (como los editores de HTML) que permiten escribir un documento viendo directamente el resultado final, frecuentemente el resultado impreso.

X

XML (eXtensible Markup Language)

Desarrollado por el World Wide Web Consortium (W3C) es lenguaje de marcado extensible que puede usarse para almacenar datos en un formato estructurado, basado en texto y definido por el usuario. Constituye un lenguaje universal de marcado para documentos estructurados y datos en la web, más amplio, más rico y más dinámico que HTML.

XSLT (Extensible Stylesheet Language Transformations)

XSLT o XSL Transformaciones es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML.

Bibliografía

(1) Fowler, Martin: Enterprise Application Architecture Patterns, Addison Wesley (2003)

(2) Gamma E., Helm, R., Johnson, R., Vlissides J.: Design Patterns: Elements of Reusable Object Oriented Software, Addison Wesley (1995).

(3) Microsoft Corporation, Patterns & Practices. Enterprise Development Reference Implementation, Versión 1.1 (2003)

(4) Microsoft Corporation, Patterns & Practices. Enterprise Solution patterns using Microsoft .net, Versión 2.0 (2003)

(5) Lutz Prechelt, An Experiment on the Usefulness of Design Patterns: Detailed Description and Evaluation> [en línea] <http://wwwipd.ira.uka.de/%7Eprechelt/Biblio/jakk_pattern09-1997.ps.gz> [Consulta: 15 Octubre 2005]

(6) Miscrosoft Corporation, Patterns & Practices. Application Architecture for .NET: Designing Applications and Services, (2004)

(7) Blanco, Luis Miguel, Introducción a .NET Framework, Grupo EIDOS Consultaría y Documentación Informática, S.L., 2002

(8) Microsoft Corporation, Enterprise Development Reference Architecture, Microsoft Press, 2004.

(9) Moisés Diaz, Daniel, “Cómo desarrollar una arquitectura de software: los lenguajes de patrones”. [en línea] <http://www.programacion.com/articulo/lenguajes_patrones>. [Consulta: 9 Octubre 2006]

(10) Barbara Unger, Peter Broessler, Walter F. Tichy, A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions> [en línea] <http://wwwipd.ira.uka.de/%7Eprechelt/Biblio/patmain.ps.gz> [Consulta: 15 Octubre 2005]

(11) Gonzalez Haaron, Inicios con EDRA (Enterprise Development Reference Architecture) [en línea] <http://weblogs.golemproject.com/hgonzalez/posts/2027.aspx> [Consulta: 20 Octubre 2005]

(12) Microsoft – GotDotNet, Comunidad - WorkSpace de patterns & practices: Enterprise Development Reference Architecture) [en línea] <http://www.gotdotnet.com/codegallery/codegallery.aspx?id=9c29a963-594e-4e7a-9c45-576198df8058> [Consulta: 4 Octubre 2005]

(13) Desarrollo Web, Ventajas de .NET [en línea] <http://www.desarrolloweb.com/articulos/1329.php?manual=48> [Consulta: 3 Marzo 2006]

(14) León Welicki, MSDN, Patrones y Antipatrones: una Introducción [en línea] < http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_ 3317.asp> [Consulta: 4 Abril 2006]

(15) John Earles, Framework Evolution [en línea] < http://www.cbd-hq.com/articles/2000/000401je_frameworks2.asp> [Consulta: 23 Abril 2006]

(16) Mohamed Fayad, Douglas C. Schmidt, Object-Oriented Application Frameworks [en línea] < http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html> [Consulta: 23 Abril 2006]

(17) Creangel, UML [en línea] <http://www.creangel.com/uml/home.php> [Consulta: 25 Septiembre 2006]

(18) Microsoft, MSDN, Arquitectura de aplicaciones de .NET: Diseño de aplicaciones y servicios [en línea] <http://www.microsoft.com/spanish/msdn/arquitectura/das/distapp.asp> [Consulta: 10 Septiembre 2006]

(19) Nick Harrison, Understanding Reflection Part 2 [en linea] http://www.ondotnet.com/pub/a/dotnet/2003/11/17/reflectionpt2.html [Consulta: 05 Febrero 2007]

(20) Alejandro Ramírez, Introducción a los Patrones de Diseño [en línea] http://www.1x4x9.info/files/patrones/html/online-chunked/index.html [Consulta 05 Febrero 2007]

(21) Anónimo, Computación y Patrones [en línea] http://www.dcc.uchile.cl/~lmateu/CC10A/Apuntes/patrones/index.html [Consulta 05 Febrero 2007]

HOJA DE LEGALIZACIÓN DE FIRMAS

ELABORADO POR

_____	_____
Iván Aurelio Fernández Silva	Juan Gabriel Jalil Moreno

COORDINADOR DE CARRERA

INGENIERIA EN SISTEMAS E INFORMATICA

_____

Ing. Ramiro Delgado

Lugar y fecha: _____