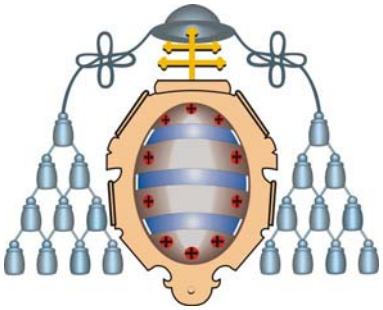
UNIVERSIDAD DE OVIEDO
Departamento de Informática



TESIS DOCTORAL

TALISMAN: Desarrollo ágil de Software con Arquitecturas Dirigidas por Modelos

Begoña Cristina Pelayo García-Bustelo

Director: Dr. Juan Manuel Cueva Lovelle

Oviedo, 2007

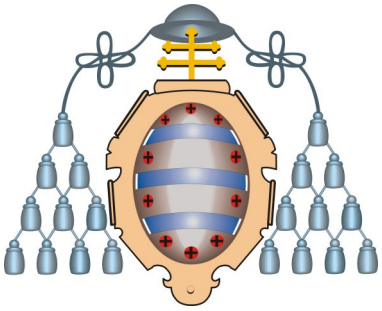
Universidad de Oviedo

TALISMÁN:
Desarrollo ágil de Software con Arquitecturas Dirigidas por Modelos

Begoña Cristina Pelayo García-Bustelo

ISBN: 978-84-694-9020-4
Depósito Legal: AS.00484-2011
http://hdl.handle.net/10803/35683

UNIVERSIDAD DE OVIEDO
Departamento de Informática



TESIS DOCTORAL

TALISMAN: Desarrollo ágil de Software con Arquitecturas Dirigidas por Modelos

Presentada por

Dña. Begoña Cristina Pelayo García-Bustelo

Para la obtención del título de Doctora por la Universidad de Oviedo

Dirigida por el

Profesor Doctor D. Juan Manuel Cueva Lovelle

Oviedo, Mayo de 2007

AGRADECIMIENTOS

A Berta Pellón Solana, in memoriam

Este trabajo ha sido realizado gracias al apoyo de muchas personas que con su contribución directa o indirecta, lo han hecho posible.

Especialmente quiero agradecer la ayuda al Director de esta Tesis, Dr. D. Juan Manuel Cueva Lovelle, por su disposición a ayudarme y orientarme a lo largo de toda la investigación.

A mis compañeros del grupo del OOTLab por sus ánimos y su esfuerzo para la realización de esta tesis.

A Ramón y Mara, por haberme apoyado en todos los momentos de mi vida, los buenos, los malos y también los otros. Os quiero.

A mi familia, por estar siempre ahí.

TALISMAN existe gracias a mi "talismán" personal, sin tenerte a mi lado, nada de esto habría sido posible. Eres mi luz.

RESUMEN

La especificación *Model Driven Architecture* (MDA), es una especialización del desarrollo dirigido por modelos que separa la lógica del negocio del software y las plataformas tecnológicas. Para ello MDA define tres tipos de modelos. Los CIM, Computation Independent Model, asociados al dominio del negocio, los PIM, Platform Independent Model, asociados a modelos abstractos del software, y los PSM, Platform Specific Model, relacionados con modelos de software específicos de plataformas tecnológicas.

Sin embargo MDA no detalla cómo deben ser los modelos CIM y tampoco describe cómo deben ser transformados a modelos PIM. Como solución a dicho problema, esta tesis presenta una recomendación que propone un proceso de desarrollo de software basado en la creación de modelos de procesos del negocio, clasificados como CIM, que son asociados a los modelos iniciales del software, considerados PIM.

Partiendo de una interpretación válida de MDA, la recomendación propuesta se apoya además en la aplicación de otras disciplinas de gran actualidad. Entre ellas destacamos el uso de *desarrollo ágil de software*, para la definición adecuada de los procesos del negocio.

Palabras clave: Model Driven Architecture (MDA), Computation Independent Model (CIM), Platform Independent Model (PIM), desarrollo ágil de software.

ABSTRACT

Model Driven Architecture (MDA) specification is a specialization of model driven development that separates business logic from software and technological platforms. In this sense it MDA defines three types of models. CIM, Computation Independent Model, related to business domain, PIM, Platform Independent Model, associated to abstract software models, and PSM, Platform Specific Model, related specific to software models of technological platforms.

Nevertheless MDA does not detail how CIM models must be and it does not describe either how they must be transformed to PIM models. Like a solution to this problem, this thesis presents a recommendation that proposes a development process based on the creation of business processes models, classified like CIM, and associated to initial software models, considered PIM.

Starting on a valid MDA interpretation, proposed recommendation is based as well on the application of other important present disciplines. Between them we emphasized, *agile software development*, for adequate definition of business processes.

Keywords: Model Driven Architecture (MDA), Computation Independent Model (CIM), Platform Independent Model (PIM), Agile Software Development.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS FUENTES

1. INTRODUCCIÓN A LA INVESTIGACIÓN

En la sociedad actual la demanda de software es cada vez más elevada, pero el proceso de desarrollo del software es también más costoso. Ello lleva a intentar aumentar la productividad en la construcción de software. Para ello se utilizan dos técnicas fundamentales: elevar el nivel de abstracción y elevar el nivel de reutilización.

La Arquitectura dirigida por modelos (MDA) [Sol00] es una iniciativa del Object Management Group [OMG] que asume las ideas de elevar el nivel de abstracción y reutilización y además introduce una nueva idea: interoperabilidad en tiempo de diseño. La esencia de esta iniciativa es que el proceso de crear software debería ser dirigido por la formulación de modelos en lugar de por la escritura manual de código fuente.

Para ello se definió una arquitectura de estándares y un conjunto de directrices que permiten expresar las especificaciones de software como modelos. El núcleo de la arquitectura de estándares engloba Unified Modeling Language [UML], Common Warehouse Metamodel [CWM] y Meta-Object Facility [MOF06].

Dichos modelos [RFW+04] representarían todos los aspectos del programa final ejecutable, por lo que deberían ser formales, precisos y con una semántica bien definida. La iniciativa MDA y los estándares que engloba tienen como objetivo situar los modelos en un nivel de abstracción tal que su realización en múltiples plataformas sea posible [MRM03].

Para esto se basa en el concepto de transformación [KWB03]. El proceso de transformar una especificación software en un programa ejecutable será automático. El código fuente de las aplicaciones se debería generar a partir de los modelos en un proceso de transformación, del mismo modo que el código máquina se genera a partir del código fuente escrito en un lenguaje de alto nivel en un compilador tradicional.

El desarrollo ágil de software [Lar03] define nuevas metodologías para afrontar el desarrollo del software de una forma más eficiente, y por tanto menos costosa. Estos métodos, llamados inicialmente ligeros [AJ02], por contraposición a los tradicionales métodos pesados, asumen el cambio como algo inevitable, para lo cual se hace necesaria una nueva forma de abordar el desarrollo de software que facilite el acomodo a los nuevos requisitos a medida que éstos surjan, en vez de pretender analizar inicialmente el dominio a modelar de forma tan exhaustiva que ya no se produzca luego ningún cambio o estos sean mínimos.

Estos métodos ligeros están centrados en el código, de tal forma que este se convierte en la principal documentación del proyecto. Esta ausencia de documentación pone de manifiesto dos diferencias mucho más significativas [Fow05]:

- Los métodos ágiles son adaptables, más que predictivos.
- Los métodos ágiles se centran más en las personas que en el proceso.
- Las características de estos métodos se describieron en el Manifiesto para el Desarrollo de Software Ágil, firmado en febrero de 2001 [Bec01].

Muchos de los principios de los métodos ágiles suponen procesos y relaciones con el cliente y su mantenimiento, no código. En este sentido, los principios ágiles de desarrollo de código pueden ser aplicables a la construcción de modelos ejecutables [MSU+04]. Bajo esta perspectiva podemos asumir que un modelo ejecutable es código [Amb04].

El objetivo principal de esta tesis es la adaptación de la Arquitectura dirigida por modelos a los métodos ágiles de desarrollo del Software [PCJ06]. Para ello se desarrolla una metodología y un prototipo necesario para probarla empleando modelos ejecutables como primer artefacto, dado que tienen mayor nivel de abstracción que el código, lo que significa que estos modelos aumentan la comunicación con los clientes y mejora la interacción con el dominio del cliente.

1.1. JUSTIFICACIÓN DE LA INVESTIGACIÓN E HIPÓTESIS DE PARTIDA

Tras analizar la propuesta MDA de OMG se han detectado dos importantes carencias relacionadas con el tratamiento de los modelos iniciales:

- No se aclara cómo deben manejarse los modelos ligados al negocio, lo que se conoce como modelos independientes de la computación ó CIM.
- No se describe cómo deben asociarse esos modelos CIM con los primeros modelos del software a desarrollar, es decir, con los PIM.

Debido a las carencias mencionadas la mayoría de trabajos relacionados con el desarrollo de software basado en MDD ó MDA suelen enfocarse desde una perspectiva puramente tecnológica. Salvo contadas excepciones no se trabaja partiendo desde un punto de vista orientado al negocio y relacionado con los objetivos empresariales.

La mayor parte de las investigaciones actuales sobre este campo están centradas en aspectos técnicos que pueden mejorar la obtención de código a partir de modelos más o menos abstractos. Esos modelos suelen estar directamente relacionados con el software, como son los modelos de requisitos, de información, de casos de uso, de análisis, de diseño, etc.

De todo lo anterior se deduce que la falta de conexión real entre el dominio del negocio y el tecnológico provoca que, en la mayoría de los casos, los modelos del negocio no sean utilizados como punto de partida de los procesos de desarrollo de software. Este problema se aprecia claramente en el ámbito de MDA, siendo evidente la falta de asociación entre modelos CIM y PIM.

Las metodologías de desarrollo ágil de software tiene como punto fuerte la conexión de los aspectos asociados al negocio con los tecnológicos. Sin embargo, no se ajusta a la especificación MDA y por lo tanto no puede aportar los beneficios inherentes, relacionados con el desarrollo de software dirigido por modelos.

Por consiguiente, una vez expuestas de forma resumida las ventajas, desventajas y deficiencias de los modelos de desarrollo mencionados y con el objetivo de solventar sus puntos débiles y aprovechar sus aportaciones, planteamos la siguiente hipótesis de partida en esta tesis:

> ***La combinación de la especificación MDA con la idea del desarrollo ágil resultaría muy ventajosa de cara a un desarrollo de software flexible y asociado a los procesos del negocio.***

Para que esta hipótesis pueda cumplirse consideramos que el desarrollo de software debería realizase a partir del establecimiento de un contexto de desarrollo ágil de software. Sólo de esa forma creemos que será posible partir de modelos de tipo CIM, asociados a los procesos del negocio, y transformarlos a modelos de software independientes de las plataformas, de tipo PIM.

Si se consigue la mencionada combinación entre MDA y desarrollo ágil de software se estará contribuyendo a resolver el problema mencionado sobre la falta de conexión entre el dominio del negocio y el tecnológico. Además se favorecerá la definición de procesos de desarrollo de software más rápidos y flexibles, que reaccionen rápidamente ante los cambios en el negocio y que no dependan tanto de las tecnologías.

1.2. OBJETIVOS

Partiendo de la hipótesis que acabamos de plantear en el apartado anterior, el objetivo principal de la tesis es el siguiente:

> ***Definir una metodología para el desarrollo de software que permita la creación de software a partir del modelado de procesos del negocio de acuerdo a la especificación MDA y el desarrollo de software ágil.***

Es importante aclarar que nuestra investigación se centra en las etapas iniciales del proceso de desarrollo, aquellas relacionadas con los modelos CIM y PIM de MDA, dado que, tal y como se ha señalado anteriormente, esa parte es la que aún no ha sido suficientemente estudiada dentro de la comunidad científica.

El objetivo principal se concreta en los siguientes objetivos parciales:

- Realizar un estudio de las áreas relacionadas con el objetivo principal, es decir, MDA, desarrollo ágil de software, patrones de diseño y estándares.
- Definir una metodología para el desarrollo de software ágil de acuerdo a la especificación MDA.
- Crear un prototipo que implemente la metodología.
- Desarrollar una aplicación usando la metodología y utilizando el prototipo.

1.3. METODOLOGÍA DE LA INVESTIGACIÓN

El proceso metodológico que se ha utilizado en esta investigación está basado en un método científico y puede dividirse en varias fases generales. Concretamente las descritas a continuación:

1. Formulación del problema: En esta primera etapa se realiza un planteamiento del problema, la definición de la hipótesis de partida y de los objetivos principales de la investigación. La descripción de toda esta información se recoge en este capítulo.

2. Recopilación y estudio de documentación: Durante esta fase se ha hecho acopio de toda la información necesaria para elaborar la base de conocimientos básicos necesarios para afrontar la investigación. Toda esa documentación ha sido organizada y estructurada en función a su contenido y relación con nuestra investigación de forma que permita una cómoda consulta y actualización.
3. Documentar la base teórica. Una vez asimilada la información recopilada en el apartado anterior, se procede a documentar en los primeros capítulos aquellas disciplinas que están directamente relacionados con el área de esta investigación, es decir, MDA, desarrollo ágil de software y modelado del negocio. Esta información servirá de soporte para el análisis y las investigaciones posteriores.
4. Analizar trabajos relacionados con la investigación. En esta etapa se realiza un estudio de trabajos y propuestas muy relacionados con los objetivos a alcanzar. Tras dicho análisis se pueden plantear, para cada uno de esos trabajos, los puntos a favor y en contra en relación a cómo pueden ser utilizados para la resolución del problema formulado y la hipótesis de partida.
5. Desarrollo de la investigación. Con los conocimientos y el material adquiridos en las etapas anteriores se desarrollará la investigación. Este desarrollo desemboca en la documentación de los capítulos principales de la tesis donde se especifica la metodología aportada.
6. Construcción de un prototipo. Una vez que la metodología ha sido definida, se desarrolla un prototipo para probar la utilización de la misma en casos prácticos.
7. Conclusiones. Esta etapa se expresan los resultados y aportaciones de la investigación, fundamentados en las discusiones reflejadas en el resto de la memoria de la tesis.

1.4. ORGANIZACIÓN DE LA MEMORIA DE TESIS

En consonancia con la metodología que acabamos de describir, la memoria de tesis sigue la siguiente estructura:

- **Introducción a la investigación.**

Esta primera parte consta únicamente de este primer capítulo de introducción en el que se realiza el planteamiento del problema, se presenta la hipótesis de partida para la investigación y se definen los objetivos principales de la tesis.

- **Estado del arte**

En esta segunda parte se describen las áreas relacionadas con la investigación que, en este caso, son extensas y relativamente complejas.

El capítulo 2 se presenta una introducción al desarrollo de Software ágil. En el capítulo 3 se hace un estudio de la Arquitecturas Dirigidas por Modelos, además se presentan algunas implementaciones de Software libre y comercial.

El capítulo 4 trata de los patrones de diseño, cómo elementos básicos del diseño Orientado a Objetos.

En los siguientes capítulos se estudian los estándares de OMG, Unified Modeling Language (UML) capítulo 5, Object Constraint Language (OCL) capítulo 6, Meta-Object Facility (MOF) capítulo 7 y XML Metadata Interchange (XMI) capítulo 8.

Se realiza una investigación sobre las distintas alternativas para construir Sistemas de Persistencia de Objetos en el capítulo 9.

Otra parte fundamental en los MDAs es la generación automática de Interfaces, que se estudia en el capítulo 10.

Para finalizar el estudio de las tecnologías, en el capítulo 11, se muestran las plataformas J2EE y .NET.

- **Desarrollo de la investigación**

En esta parte se han incluido varios capítulos para describir de forma progresiva el desarrollo de la investigación, esto es, las partes de la metodología propuesta para el desarrollo de software ágil de acuerdo a MDA.

El capítulo 12 contiene una descripción general de la metodología así como de sus pasos principales. Además, en el capítulo 13 se describe el prototipo asociado a la metodología.

En el capítulo 14 se describen los resultados obtenidos tras la aplicación de la metodología y utilización del prototipo para el desarrollo de una aplicación Web.

- **Conclusiones**

Esta parte, del capítulo 15 se describe los resultados obtenidos tras la investigación realizada en la parte anterior. Además plantea posibles líneas de investigación a seguir dentro de las áreas abordadas en esta tesis.

- **Anexos**

Al final de esta memoria se han incluido varios anexos cuyo objetivo es contener código fuentes del prototipo y de la aplicación desarrollada con el mismo.

- **Lista de acrónimos**

Se acompaña una lista de referencia con los acrónimos que son empleados a lo largo de esta memoria, para facilitar al lector la consulta y comprensión de los diferentes pasajes de esta tesis.

- **Bibliografía**

Esta última parte contiene la relación de referencias bibliográficas y de la Web que han sido empleadas en el desarrollo de esta tesis y la investigación asociada. Se presentan debidamente formateadas de acuerdo a estándares bibliográficos.

2. DESARROLLO DE SOFTWARE ÁGIL

La comunidad de la ingeniería del software siempre se encuentra a la búsqueda de nuevas formas de desarrollar software en este contexto hicieron su aparición los llamados *métodos ágiles de desarrollo*. El caso paradigmático, y también el más conocido es, sin duda, **Extreme Programming,** también conocido como XP. [Bec99b]

Estos métodos, llamados inicialmente *ligeros*, por contraposición a los tradicionales métodos pesados o burocráticos, asumen el cambio como algo inevitable, para lo cual se hace necesaria una nueva forma de abordar el desarrollo de software que facilite el acomodo a los nuevos requisitos a medida que éstos surjan, en vez de pretender ser capaces de analizar inicialmente el dominio a modelar de forma tan exhaustiva que ya no se produzca luego ningún cambio o éstos sean mínimos (el tradicional modelo en cascada o, a lo sumo, con unas pocas iteraciones de los métodos pesados) [Fer02]

El desarrollo de software dista cada vez más de ser algo precedible, por ello éste debe ser cada vez más flexible. Para esto se debe dotar a los participantes en los proyectos informáticos (clientes, jefes de proyecto, programadores…) de las herramientas y técnicas necesarias para que dichos cambios continuos en las especificaciones no sean traumáticos, sino que puedan llevarse a cabo de manera natural (o ágil).

Así, estos métodos ágiles de desarrollo constituyen por sí mismos una respuesta a dicha realidad cambiante; pero, como métodos que son, lo hacen fundamentalmente desde la perspectiva de la gestión de proyectos y, en todo caso, del análisis y el diseño.

La orientación a objetos ha contribuido a incrementar significativamente la flexibilidad en el desarrollo de software, pero sigue sin ofrecer el grado de flexibilidad requerido para abordar la complejidad (requisitos cambiantes) de las aplicaciones modernas. Se hace necesario un nuevo paradigma de programación que venga a suavizar ese salto todavía muy grande que existe entre el diseño y el código. Se trata, en definitiva, de ir un paso más allá e incrementar aún más el nivel de abstracción de los lenguajes de programación, de forma que resulten mucho más cercanos al dominio a modelar.

La aparición de los Patrones de Diseño [*véase* Capítulo 4] es una respuesta a esa falta de flexibilidad dado que muchos de ellos responden más bien a determinadas carencias de los lenguajes de programación orientados a objetos en los que son implementados. En este sentido, en presencia de ciertos mecanismos de programación con los que sí cuentan otros lenguajes (no necesariamente, como se verá, orientados a objetos) muchos de los patrones existentes verían notablemente simplificada su implementación, llegando incluso en algunos casos a desaparecer, por triviales.

En este ámbito, los MDA ágiles se basan en que el código y los modelos ejecutables son operacionalmente lo mismo. Un modelo ejecutable puede ser construido, ejecutado, probado y modificado en ciclos cortos e incrementales.

2.1. MÉTODOS ÁGILES DE DESARROLLO DE SOFTWARE

Los métodos ágiles o ligeros se encuentran en contraposición a los métodos pesados tradicionales. Este tipo de métodos surgieron como respuesta al caos en el que estaba sumido el desarrollo de software, hasta el punto de que muchas veces la única planificación existente consistía en "codificar primero y arreglar después". Frente a esa actitud, estos métodos trataron de imponer una disciplina en el proceso de desarrollo, con el objetivo de que éste fuese así más predecible. Para ello tomaron como fuente de inspiración otras ingenierías y crearon procesos con un fuerte componente de planificación y de documentación.

Frente a estos métodos pesados, han aparecido una serie de métodos denominados ligeros o ágiles que tratan de una solución de compromiso entre la ausencia de proceso y el exceso de éste. La diferencia más obvia entre este tipo de métodos y los pesados es que generan mucha menos documentación que aquellos. Estos métodos ligeros están centrados en el código, de tal forma que éste se convierte en la principal documentación del proyecto. Sin embargo, no debemos quedarnos con esta llamativa característica como la principal diferencia entre ambos. Lo que esta ausencia de documentación pone de manifiesto son dos diferencias mucho más significativas, como afirma Fowler [Fow05]:

- Los métodos ágiles son adaptables, más que predictivos. Los métodos pesados tienden a planificar con gran detalle todo el proceso de desarrollo, pero esto sólo funciona mientras las cosas no cambien. Frente a esta resistencia al cambio, los métodos ágiles asumen que éste se va a producir, y se preparan para recibirlo.
- Los métodos ágiles se centran más en las personas que en el proceso. A nuestro juicio, ésta constituye una de las principales características de este tipo de métodos, que explicitan así la necesidad de tener en cuenta la naturaleza de las personas en vez de ir contra ella.

Las características de estos métodos se muestran en la Tabla 2.1 Estas características están extraídas del *Manifiesto para el Desarrollo de Software Ágil* [Bec01], firmado en febrero de 2001 por representantes de Extreme Programming, SCRUM, DSDM y otros métodos ágiles.

Se prefiere	Frente a
Las personas y las relaciones	Los procesos y herramientas
El software que funciona	La documentación
La colaboración del cliente	Los contratos
Responder a los cambios	Seguir un plan

Tabla 2.1. Características de los métodos ágiles.

El nuevo proceso -iterativo, incremental, involutivo y oportunista- alentado por la tecnología de objetos, los entregables evolutivos y las variaciones del ciclo de vida en espiral pretenden conseguir un refinamiento continuo del sistema software en proceso de modelado, de forma que que la construcción por bloques (la basada en fases terminales aisladas y estrictamente secuenciales) se ha trocado en una creación de software "sobre plastilina", en la que en cualquier momento pueden matizarse detalles o insertar cambios sin que se produzcan rupturas o impactos de quiebra.

Muchos de los principios de XP suponen procesos y relaciones con el cliente y su mantenimiento, no código. En este sentido, los principios ágiles de desarrollo de código pueden ser aplicables a la construcción de modelos ejecutables [MSU+04]. Bajo esta perspectiva podemos asumir que un modelo ejecutable es código.

Los MDAs ágiles emplean modelos ejecutables como primer artefacto, que tienen mayor nivel de abstracción que el código, lo que significa que estos modelos aumentan la comunicación con los clientes y mejora la interacción con el dominio del cliente.

Existen métodos ágiles existentes son:

- Extreme Programming (XP)
- Scrum
- Adaptive Software Development (ASD)
- Crystal Clear y otras metodologías de la familia Crystal
- DSDM
- Feature Driven Development
- Lean software development

En este trabajo de investigación con objeto de comprender la forma de actuar con estos métodos vamos a centrar el estudio en el más representativo de ellos: Extreme Programming (XP).

2.2. EXTREME PROGRAMMING (XP)

Extreme Programming es un método ágil de desarrollo de software pensado para equipos pequeños o medianos que se enfrentan a requisitos vagos o cambiantes.

Como sugiere Beck [Bec99a], XP lleva un conjunto de prácticas de sentido común al extremo, de tal manera que:

- El código será revisado continuamente, mediante la programación en parejas (dos personas por máquina).
- Se harán pruebas todo el tiempo, no sólo de cada nueva clase (pruebas unitarias) sino que también los clientes comprobarán que el proyecto va satisfaciendo los requisitos (pruebas funcionales).
- Las pruebas de integración se efectuarán siempre, antes de añadir cualquier nueva clase al proyecto, o después de modificar cualquiera existente (integración continua), para lo que nos serviremos de frameworks de pruebas, como JUnit [Jun].
- Se (re)diseñará todo el tiempo ("refactoring"), dejando el código siempre en el estado más simple posible.
- Las iteraciones serán radicalmente más cortas de lo que es usual en otros métodos, de manera que nos podamos beneficiar de la retroalimentación tan a menudo como sea posible.

2.2.1. El coste del cambio

El coste del cambio en el desarrollo de software se incrementaba exponencialmente en el tiempo (Figura 2.1.). En concreto, el coste de detectar y corregir un error se multiplica por un factor de aproximadamente 10 al pasar de la fase de análisis a la especificación de requisitos, de esta al diseño y así sucesivamente. Por tanto, los esfuerzos de los métodos pesados han ido encaminados a invertir grandes cantidades de tiempo y documentación durante las primeras fases de desarrollo (especialmente en el establecimiento de los requisitos), en la creencia de que este esfuerzo bien valdría la pena al ahorrar un tiempo mucho mayor en fases posteriores.

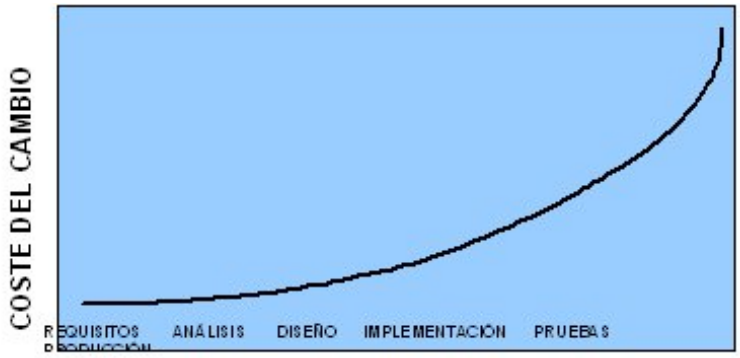


Figura 2.1. Coste del cambio tradicional en ingeniería del software.

Sin embargo, una de las asunciones más innovadoras de XP es que esta curva ya no es válida, y que con una adecuada combinación de herramientas y tecnología (velocidad de los ordenadores, disminución del ciclo de compilación y pruebas de días a segundos, bases de datos, orientación a objetos, etc.) es posible obtener la curva de coste del cambio que se muestra en la Figura 2.2.

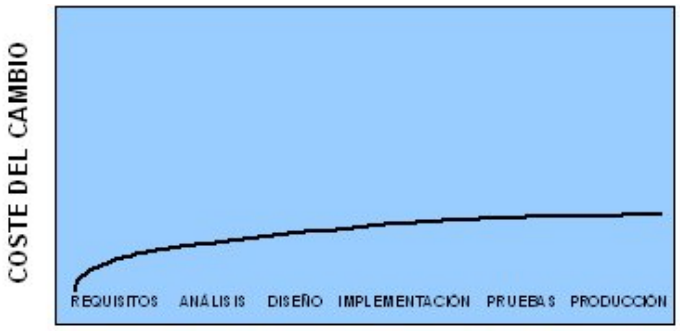


Figura 2.2. Coste del cambio en XP.

Pero esta afirmación [Bec99b] no es demostrada y además hay quien, como Cockburn [Coc00], no está de acuerdo con ella y sostiene que la curva tradicional sigue siendo válida, pero que eso no supone una crítica a XP, sino todo lo contrario, ya que, según él, XP no sólo no depende de la ausencia de dicha curva exponencial, sino que sus ventajas provienen precisamente de la existencia de la misma y de cómo este hecho es manejado por el método mucho mejor que los tradicionales métodos pesados.

No obstante, lo realmente importante aquí no es si la curva sigue siendo o no exponencial, sino cuál debería ser nuestra actitud al enfrentarnos al desarrollo de software si la curva fuese la propuesta originalmente por XP.

2.2.1.1. Consecuencias del nuevo coste del cambio

Si resulta que el coste del cambio no crece drásticamente en el tiempo, entonces:

- No será necesario hacer gran cantidad de análisis inicial, eliminando así gran parte de las suposiciones -muchas de ellas erróneas- sobre las futuras necesidades del proyecto.
- No trataremos de adivinar el futuro, planificando algo que posiblemente nunca sea necesario.
- Trataremos de retrasar todas las decisiones hasta el último momento posible, de manera que el cliente sólo pagará por aquello que realmente vaya a usar.

Es decir, la idea fundamental aquí es que, en vez de diseñar para tratar de anticiparnos al cambio, diseñaremos tan sencillo como sea posible, para hacer sólo lo que sea imprescindible en un momento dado, pues la propia simplicidad del código, junto con la factorización, y, sobre todo, las pruebas y la integración continua, hacen posible que los cambios puedan ser llevados a cabo tan a menudo como sea necesario. Pero nos estamos anticipando al hacer referencia a algunas de las prácticas del método que aún no han sido mencionadas, y que se irán describiendo en las siguientes secciones. Por tanto XP es un método que reduce el riesgo de tomar decisiones de diseño erróneas en las fases más tempranas del proyecto, con el consiguiente coste, no sólo del tiempo empleado en programar una funcionalidad que no será utilizada, sino sobre todo el coste de oportunidad de no haber dedicado ese tiempo a programar lo realmente necesario.

2.2.2. Ciclo de Vida

XP es un método centrado en el código pero sobre todo es un método de gestión de proyectos software [BF01]. Si se acepta que el desarrollo de software es un proceso caótico, XP no trata de buscar un determinismo inexistente y sí de poner los medios necesarios para manejar esa complejidad, aceptándola. En definitiva, lo que XP propugna es que la planificación no consiste en predecir el futuro, como hacen los métodos pesados, porque para cuando el software hubiera sido desarrollado el cliente ya no querría lo planificado: sus necesidades habrían cambiado entretanto.

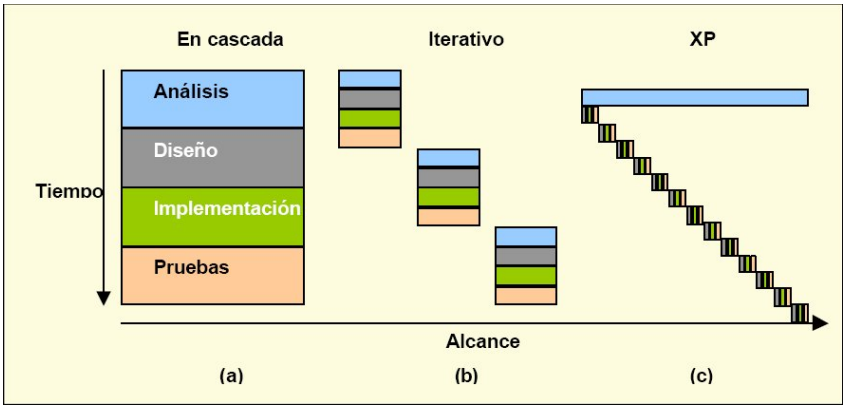


Figura 2.3. Evolución de los ciclos de desarrollo.

El ciclo de vida de XP se basa en ciclos de desarrollo más cortos, de hecho es una de las ideas centrales de XP. En la

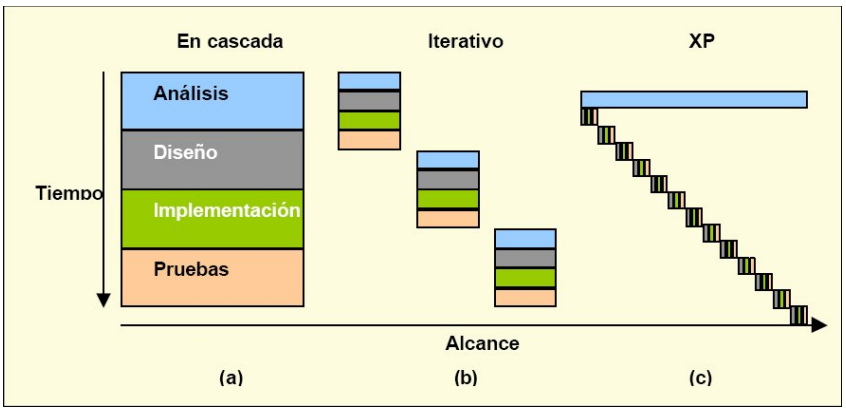


Figura 2.3.puede verse la evolución de los largos ciclos de desarrollo del modelo en cascada (a) a los ciclos iterativos más cortos de, por ejemplo, el modelo en espiral (b) y, finalmente, a la mezcla que XP (c) hace de todas estas actividades a lo largo de todo el proceso de desarrollo de software [Bec01].

XP define cuatro variables para cualquier proyecto software: **coste, tiempo, calidad y alcance**. Además, especifica que, de estas cuatro variables, sólo tres de ellas podrán ser fijadas por las fuerzas externas al proyecto (clientes y jefes de proyecto), mientras que el valor de la variable libre será establecido por el equipo de desarrollo en función de los valores de las otras tres. Normalmente los clientes y jefes de proyecto se creen capaces de fijar de antemano el valor de todas las variables: ``Quiero estos requisitos satisfechos para el día uno del mes que viene, para lo cual contáis con este equipo. ¡Ah, y ya sabéis que la calidad es lo primero!'' [Fer02]. Pero habitualmente la calidad es lo primero que se olvida dado que nadie es capaz de trabajar bien cuando está sometido a presión.

XP hace a las cuatro variables visibles para todo el mundo (programadores, clientes y jefes de proyecto), de manera que se pueda jugar con los valores de la entrada hasta que la cuarta variable tenga un valor que satisfaga a todos.

Además las cuatro variables no guardan entre sí una relación tan obvia como a menudo se quiere ver. XP hace especial énfasis en equipos de desarrollo pequeños (diez o doce personas como mucho) que, naturalmente, se podrán ir incrementando a medida que sea necesario, pero no antes, o los resultados serán generalmente contrarios a lo esperado. Sí es bueno, en cambio, incrementar el **coste** del proyecto en aspectos como máquinas más rápidas, más especialistas técnicos en determinadas áreas o mejores oficinas para el equipo de desarrollo.

Frecuentemente, aumentar la **calidad** conduce a que el proyecto pueda realizarse en menos tiempo. En efecto, en cuanto el equipo de desarrollo se habitúe a realizar pruebas intensivas y se sigan estándares de codificación, poco a poco comenzará a avanzar mucho más rápido de lo que lo hacía antes, mientras la calidad del proyecto se mantiene asegurada (por las pruebas) al 100\%, lo que conlleva mayor confianza en el código y, por tanto, mayor facilidad para adaptarse al cambio, sin estrés, lo que hace que se programe más rápido… y así sucesivamente.

Existe la tentación de sacrificar la calidad interna del proyecto (la apreciada por los programadores) para reducir el tiempo de entrega del proyecto, en la confianza de que la calidad externa (la que notan los clientes) no se vea demasiado afectada. Sin embargo ésta es una apuesta a muy corto plazo que suele ser una invitación al desastre, pues obvia el hecho fundamental de que todo el mundo trabaja mucho mejor cuando le dejan hacer trabajo de calidad. No tener esto en cuenta conduce a la desmoralización del equipo y, con ello, a la larga, a la ralentización del proyecto mucho más allá del **tiempo** que hubiera podido ganarse al principio con esta reducción de calidad.

En cuanto al **alcance** del proyecto, es una buena idea dejar que sea ésta la variable libre, de manera que, una vez fijadas las otras tres, el equipo de desarrollo determinaría el alcance mediante:

- La estimación de las tareas a realizar para satisfacer los requisitos del cliente
- La implementación de los requisitos más importantes primero, de manera que el proyecto tenga en cada instante tanta funcionalidad como sea posible

2.2.3. Las prácticas

El núcleo de XP lo forman **doce prácticas**, que se apoyan unas en otras, potenciando así la acción que cada una de ellas tendría por separado. Estas prácticas se agrupan en cuatro categorías, además la mayoría de estas prácticas no son nuevas, sino que han sido reconocidas por la industria como las mejores prácticas durante años. En la XP, dichas prácticas son llevadas al extremo para que se obtenga algo mucho mejor que la suma de las partes. Las doce prácticas son:

2.2.3.1. La planificación

La planificación en XP es importante porque se necesita asegurar que se trabaja siempre en lo más importante que queda por hacer, además se debe coordinar con otras personas y además se debe saber qué hacer ante un hecho inesperado, es decir, cómo ese hecho afecta a los dos puntos anteriores. La planificación es tan buena como lo sean las estimaciones en que se basa, pero el mundo real siempre desbarata lo planificado. La clave es ajustar el plan tan pronto como tengamos constancia de las consecuencias de cualquier hecho inesperado.

Por tanto la planificación debe ser fácil de realizar y, sobre todo, fácil de mantener actualizada. Además, puesto que una de las razones de planificar es la coordinación con otras personas, la planificación tiene que ser comprensible por todos los participantes en el proyecto. Para realizar la planificación se deben introducir tres conceptos importantes en XP: historias, iteraciones y tareas.

2.2.3.1.1. Historias

Las historias son la unidad mínima de funcionalidad en un proyecto XP. Una historia consiste en una o dos frases escritas por el cliente que describen alguna característica que éste desea ver implementada en el sistema. Las historias se escribirán en fichas, de modo que se puedan manejar fácilmente. Para que una historia sea válida, debe cumplir una serie de requisitos, como son: poder estimar fácilmente su dificultad y poder escribirse pruebas funcionales para ella.

2.2.3.1.2. Iteraciones

XP divide cada versión del producto en una serie de iteraciones de entre una y tres semanas. Estas iteraciones son tan cortar para obtener una medida del progreso, es decir, conocer qué porción del camino llevamos hecha y cuánto nos queda por recorrer. En cada iteración se debe realizar: comprobar que las pruebas funcionales se ejecutan correctamente, planificar la siguiente iteración e informar a los gestores del proyecto.

2.2.3.1.3. Tareas

Una vez que el cliente ha escrito las historias, el equipo de desarrollo divide cada una de ellas en tareas. Una tarea debe ser lo suficientemente concreta como para poder realizarse en un par de días ideales de programación. Después de pasar las historias a tareas, son los propios programadores quienes eligen qué tareas realizará cada uno y se hacen responsables de ellas -autoasignación de trabajo-. El responsable de cada tarea es también quien estimará el número de días ideales de programación que le llevará implementarla.

El tiempo ideal de programación hace alusión al tiempo que estimamos que nos llevaría implementar una determinada característica, completar una historia, etc. suponiendo que todo funcionase correctamente, que no hubiera interrupciones, que no tuviésemos que ayudar a otros miembros del equipo de desarrollo, que no existiesen reuniones… por tanto tiene poco que ver con una estimación de tiempo y se refiere más bien a la dificultad de una determinada tarea o historia.

Al final siempre necesitaremos trasladar la estimación realizada a una unidad temporal. Durante el proyecto, esto resulta tanto más sencillo a medida que avanzamos en él, pues nos basamos en la velocidad del equipo de desarrollo.

2.2.3.2. Versiones cortas

El sistema se pone por primera vez en producción en, a lo sumo, unos pocos meses, antes de estar completamente terminado. Las sucesivas versiones serán aún más frecuentes. El cliente y el equipo de desarrollo se beneficiarán así de la retroalimentación que produce un sistema funcionando, de tal forma que este aprendizaje del sistema y de las necesidades reales del cliente se reflejará en las versiones posteriores. El cliente, basándose en las estimaciones hechas por el equipo de desarrollo, es quien elige las historias a incluir en cada versión.

2.2.3.3. Metáfora

Todos los participantes en el proyecto se deben referir al sistema en términos de una metáfora bien conocida y que se adecue bien al sistema a desarrollar. Esta metáfora, generalmente, provendrá de cualquier ámbito del mundo real que nada tenga que ver con el del proyecto, y viene a representar de una manera sencilla la arquitectura del sistema, al ayudar a comprender los elementos básicos del sistema y sus relaciones.

El objetivo subyacente de esta práctica es promover la comunicación entre los miembros del equipo, cliente incluido. Si a medida que el proyecto avanza descubrimos que la metáfora falla en algunos aspectos, ésta se irá refinando o se sustituirá por otra que represente mejor el sistema.

2.2.3.4. Diseño simple

Por diseño simple se entiende aquel que: pasa todas las pruebas, no tiene lógica duplicada, revela su intención y tiene el menor número posible de clases y métodos.

Una técnica que se adecua a la perfección a XP son las fichas CRC [BC89]. En la Figura 2.4. puede observarse un ejemplo de ficha CRC, precisamente una de las empleadas en el framework para editores de dibujo HotDraw, creado por Kent Beck y Ward Cunningham (1994).

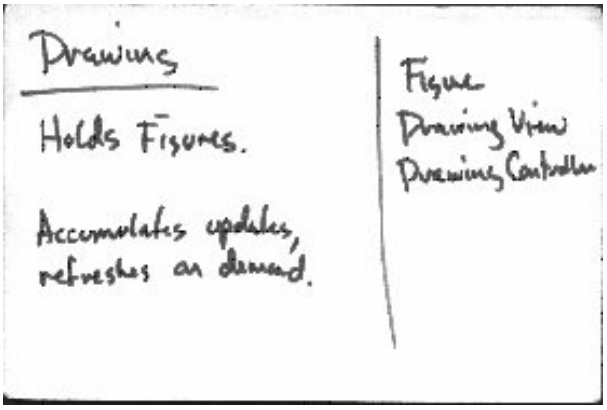


Figura 2.4. Ejemplo de ficha CRC.

2.2.3.5. Pruebas

Las pruebas en los métodos tradicionales constituyen una fase más, situada generalmente al final del proyecto o de cada iteración, en XP éstas se convierten en la parte central del proceso. "Cualquier funcionalidad de un programa para la que no haya una prueba automatizada, simplemente no existe" [bec99b].

En XP no sólo existen pruebas unitarias, sino que tenemos para todas las historias pruebas funcionales escritas por el cliente para poder comprobar que cada historia haya sido implementada correctamente en el sistema. Además, las pruebas de integración se ejecutarán siempre que se añada cualquier nueva clase al sistema, o que se modifique alguna existente.

Es necesario que todas las pruebas se ejecuten de manera automática, con tan sólo pulsar botón. Para ello se debe utilizar un framework de pruebas automatizadas, como JUnit [Jun]. (Figura 2.5.)

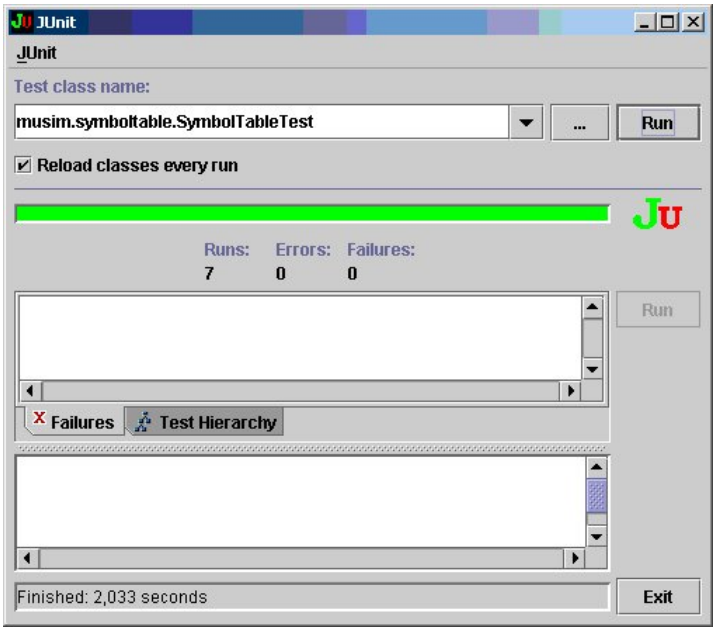


Figura 2.5. JUnit en funcionamiento.

2.2.3.6. Factorización

La factorización [Fow99] del código (en inglés, refactoring) responde al principio de simplicidad, y tiene como objetivo dejar el código en el estado más simple posible. Para ello, se retocará el código existente, que funciona, sin añadirle ninguna funcionalidad, de manera que se sigan ejecutando todas las pruebas, pero con un código que será mucho más claro y, por tanto, más sencillo de mantener.

2.2.3.7. Programación en parejas

Todo el código será desarrollado en parejas, dos personas compartiendo un solo monitor y teclado. El que codifica pensará en el mejor modo de implementar un determinado método, mientras que su compañero lo hará de una manera más estratégica: se sigue el enfoque apropiado, que pruebas podrían seguir sin funcionar, se puede simplificar el sistema,… Los roles son intercambiables y la composición de las parejas puede cambiar siempre que uno de los dos sea requerido por algún otro miembro del equipo para que le ayude con su código.

2.2.3.8. Propiedad colectiva del código

Cualquiera puede modificar cualquier porción del código, en cualquier momento. El uso de estándares de codificación y la confianza que nos dan las pruebas de que todo va a seguir funcionando bien tras una modificación, hacen que esto no sea ningún problema en XP.

2.2.3.9. Integración continua

Cada pocas horas se integra el sistema completo Para ello existirá una máquina de integración, en la que una pareja de programadores cada vez que tengan una clase que haya sido probada unitariamente, la añadirá junto con sus pruebas unitarias, al sistema completo que debe seguir funcionando correctamente. Si es así los programadores darán por finalizada esa tarea. Si no, serán los responsables de dejar el sistema de nuevo con sus pruebas funcionando al 100%.

2.2.3.10. 40 horas semanales

Para ofrecer calidad cada miembro del equipo debe tener un tiempo justo de trabajo y un tiempo de descanso. Las 40 horas no es una regla fija, puede oscilar entre 35 a 45, pero lo que es seguro es que nadie es capaz de trabajar 60 horas a la semana y hacerlo con calidad.

2.2.3.11. Cliente en el sitio

Al menos un cliente real debe estar permanentemente junto al equipo de desarrollo, para responder cualquier consulta que los programadores le planteen, para establecer prioridades…

2.2.3.12. Estándares de codificación

Es decisiva la codificación basada en estándares que haga que todos los miembros del equipo entiendan el código escrito por otro miembro del equipo, por tanto para el éxito de la propiedad colectiva del código.

2.3. UTILIZACIÓN DE MDA ÁGILES

Al utilizar MDA ágiles conseguimos una retroalimentación de los clientes acerca del producto durante el desarrollo. Se escriben pruebas, modelos ejecutables, se compila el modelo utilizando compiladores de modelos, se ejecutan las pruebas, y se entregan fragmentos del sistema al cliente de forma incremental. Por tanto se sigue un enfoque ágil de desarrollo.

Un modelo ejecutable, dado que "es"' ejecutable, puede ser construido, ejecutado, probado y modificado ciclos incrementales [MSU+04]. El procesamiento de los ciclos utilizando modelos no tiene porque ser mas largos que codificando, dado que los modelos se transforman automáticamente dentro del sistema. Como resultado de esto, la producción de modelos ejecutables no tiene por ser más pesado que escribir código, con una infraestructura MDA adecuada. Por tanto, los MDA ágiles son una posibilidad para la construcción de modelos, que pueden ser unidos para generar un sistema.

3. MODEL DRIVEN ARCHITECTURE (MDA)

Model Driven Architecture (MDA) permite la construcción de sistemas de computación a partir de modelos que pueden ser entendidos más fácilmente que el código de un lenguaje de programación. Esta arquitectura fue formulada por el Object Management Group (OMG). MDA parte de la conocida y aceptada idea de separar la especificación de las operaciones de los detalles acerca de cómo el sistema utiliza las posibilidades de la plataforma.

MDA proporciona un enfoque para:

- Especificar un sistema de manera independiente a la plataforma de soporte
- Especificar plataformas
- Elegir una plataforma particular para el sistema
- Transformar la especificación del sistema para una plataforma en concreto.

Los objetivos principales de MDA son la portabilidad, interoperatibilidad y la reusabilidad a través de la separación arquitectónica de conceptos. [PCS+05][PCJ06]

3.1. CONCEPTOS BÁSICOS DE MDA

A continuación se muestran los conceptos que conforman el núcleo de MDA.

3.1.1. Sistema

Los conceptos de MDA serán expresados en términos del sistema existente. Este sistema puede incluir: un programa, un ordenador simple, una combinación de diferentes parte de sistemas, una conjunción de sistemas cada uno con un control diferente (personas, una empresa, federación de empresas,…)

3.1.2. Modelo

El modelo de un sistema es la descripción o especificación de ese sistema y su adaptación a un propósito determinado. Frecuentemente se presenta a un modelo como una combinación de dibujos y texto. El texto puede estar en un lenguaje de modelado o en lenguaje natural.

En el contexto de MDA, los modelos son representaciones formales del funcionamiento, comportamiento o estructura del sistema que se está construyendo [RFW+04]. El término "formal" especifica que el lenguaje utilizado para describir el modelo debe de tener una semántica y sintaxis bien definida.

Un concepto clave a la hora de construir modelos es el de "*abstracción*". Se debe ignorar la información que no es de interés para el dominio del problema a resolver. Los modelos representan elementos físicos o abstractos correspondientes a una realidad simplificada, adecuada al problema que se necesita resolver.

3.1.3. Dirigido por modelos

MDA es un acercamiento al diseño del sistema, para ello se incrementa la importancia de los modelos.

Es dirigido por modelos porque aporta los medios para usar los modelos directamente para el entendimiento, diseño, construcción, despliegue, operación, mantenimiento y modificación.

3.1.4. Arquitectura

La arquitectura del sistema es una especificación de las partes y los conectores del sistema y las reglas para la interacción de las partes usando conectores.

El MDA prescribe algunos tipos de modelos para ser utilizados, cómo tienen que ser preparados esos modelos y las relaciones entre los tipos de modelos.

3.1.5. Punto de vista y Vista

Un punto de vista de un sistema es la técnica para la abstracción utilizando un conjunto seleccionado de conceptos de la arquitectura y reglas estructurales, para centrarse en conceptos particulares dentro del sistema. La palabra "abstracción" es utilizada con su significado de proceso de suprimir los detalles seleccionados para establecer un modelo simplificado.

Los conceptos y reglas pueden ser utilizados para formar un lenguaje de Punto de Vista.

El MDA especifica tres puntos de vista sobre un sistema: un punto de vista de computación independiente, un punto de vista de plataforma independiente y un punto de vista de plataforma específica.

Una Vista o un modelo de punto de vista del sistema es una representación del sistema desde la perspectiva del punto de vista elegido.

3.1.6. Plataforma

Una plataforma es un conjunto de subsistemas y tecnologías que aportan un conjunto coherente de funcionalidad a través de interfaces y patrones de uso específico.

En MDA, el término "plataforma" se utiliza generalmente para hacer referencia a los detalles tecnológicos y de ingeniería que no son relevantes de cara a la funcionalidad esencial del sistema. Esto uso del término se encuadra dentro de la separación fundamental que se realiza en MDA entre la especificación de un sistema y su plataforma de ejecución.

Se define plataforma como la especificación de un entorno de ejecución para un conjunto de modelos [MSU+04]. Estará formada por un conjunto de subsistemas y tecnologías que proporcionan una funcionalidad determinada a través de una serie de interfaces y patrones de uso determinados [MM03].

Una aplicación soportada por la plataforma puede ser utilizada sin preocuparse por los detalles de cómo se suministra la funcionalidad por parte de la implementación de la plataforma.

Ejemplos:

- Tipos genéricos de plataformas: Objetos, batch, dataflow.
- Tipos de plataformas especificas del vendedor: CORBA [COR04], J2EE [Jav05], Microsoft .NET [TT02].

3.1.7. Aplicación

En la definición de MDA se utiliza la palabra aplicación para referirse a la funcionalidad que se está diseñando. Un sistema es descrito en términos de una o más aplicaciones que son soportadas por una o mas plataformas.

3.1.8. Independencia de Plataforma

Esta es una cualidad que el modelo debe tener. Esta característica es la que hace que el modelo sea independiente de las características de un tipo particular de plataforma.

Como la mayoría de las características, la independencia de la plataforma es una cuestión de grado. Por esto, un modelo pudo asumir solamente la disponibilidad de características de un tipo muy general de plataforma, tal como invocación remota, mientras que otro modelo pudo asumir la disponibilidad un sistema particular de las herramientas para la plataforma de CORBA. Asimismo, un modelo pudo asumir la disponibilidad de una característica de un tipo particular de plataforma, mientras que otro modelo se pudo adaptar completamente a ese tipo de plataforma.

3.1.9. Puntos de Vista de los MDA

Punto de vista de computación independiente: se centra sobre la adaptación del sistema, y los requisitos para el sistema; los detalles de la estructura y el proceso del sistema están ocultos o indeterminados.

Punto de vista de independencia de plataforma: se centra en la operatividad del sistema ocultando los detalles sobre la plataforma en particular. Una vista independiente de la plataforma, muestra parte de la especificación completa que no cambia de una plataforma a otra. Esta vista puede utilizar un lenguaje de modelado de propósito general, o un lenguaje especifico al área donde el sistema será utilizado.

Punto de vista de plataforma específica: combina la vista de independencia de plataforma con un enfoque adicional centrado en los detalles del uso por parte del sistema de una plataforma específica.

3.1.10. El modelo de computación independiente (CIM)

En inglés Computation Independent Model (CIM), es una vista del sistema desde el punto de vista de la independencia de computación. Un CIM no enseña la estructura de los sistemas.

Se asume que el principal usuario del CIM, los conocedores del dominio, no conocen los modelos o artificios utilizados para realizar las funciones de los requerimientos que se articulan en el CIM. Además el CIM desempeña un importante rol en tender un puente entre los expertos del dominio y sus requisitos, y los expertos en el diseño y construcción de artefactos que juntos satisfacen los requisitos del dominio.

3.1.11. El modelo de plataforma independiente (PIM)

Se denomina *Platform Independent Model* (PIM) a una vista del sistema centrada en la operación del mismo que esconde los detalles necesarios para una determinada plataforma [MM03]. Un PIM muestra un determinado nivel de independencia de la plataforma de forma que sea posible utilizarlo con un número de plataformas diferentes de forma similar.

Una técnica muy común para conseguir la independencia de plataforma es a través de tecnología neutral de una máquina virtual [How02]. Estas máquinas virtuales son especificaciones de un procesador computacional, que pueden ser realizadas sobre múltiples plataformas computacionales, siendo típicamente su desarrollo lógico [Ort01]. Las máquinas virtuales consiguen de este modo la independencia de de los programas que ejecutan con respecto a la plataforma de ejecución final (habitualmente un sistema operativo funcionando sobre una configuración hardware determinada). Es importante señalar que MDA propone un nivel de abstracción mayor, puesto que se buscará la independencia de la plataforma y dichas máquinas virtuales serán consideradas plataformas de ejecución.

En la Figura 3.1 se recoge un escenario de utilización de MDA que ilustra diferentes niveles de abstracción y los artefactos de ejecución manejados en cada uno de ellos. En este escenario típico, una herramienta MDA recoge los diferentes PIM que especifican el sistema y los transforma en un lenguaje de alto nivel soportado por la plataforma de destino. En el caso de que la plataforma destino utilice una máquina virtual que no sea un intérprete puro \cite{book:Cueva1998}, se requerirá un proceso de compilación que obtenga el código máquina soportado por la máquina virtual. Este es el caso de las plataformas Java y .NET. Es habitual que, por razones de eficiencia, dicha máquina virtual sea compilada en cada sistema operativo como código máquina nativo, que será ejecutado finalmente por el hardware utilizado en cada caso.

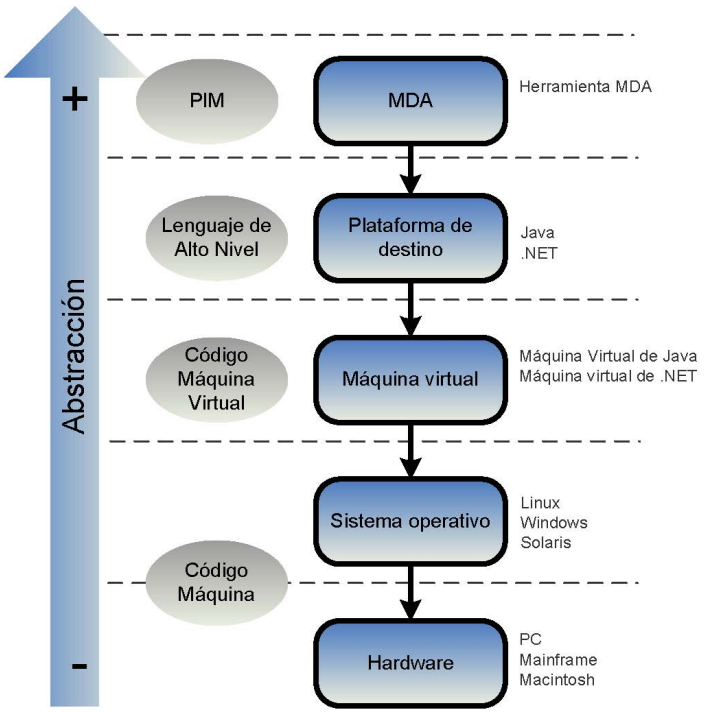


Figura 3.1. Ejemplo de niveles de abstracción e independencia con respecto a la plataforma en MDA.

3.1.12. El modelo de plataforma específica (PSM)

En Inglés Platform Specific Model (PSM) es la vista del sistema desde el punto de vista de la plataforma específica. Un PSM combina las especificaciones del PIM con los detalles que especifica cómo utiliza el sistema un tipo particular de plataforma.

Un PSM es una vista del sistema desde el punto de vista de la plataforma específica de ejecución [MM03]. Combina los diferentes PIM que especifican el sistema con los detalles acerca de cómo utiliza un tipo concreto de plataforma.

La diferencia entre un modelo PSM y uno PIM es el conocimiento que incorpora acerca de la plataforma final de implementación. Un PIM no incorpora ninguna información acerca de la plataforma de destino. Por su parte, un PSM contiene toda la información necesaria para hacer posible su implementación sobre una plataforma concreta. Es decir, contendrá toda la información necesaria para generar el código de la aplicación a partir de él [MB02].

Existen autores que consideran el código fuente de la aplicación un modelo PIM [RFW+04]. No obstante, en la propuesta del OMG el código fuente de la aplicación se considera un resultado de una transformación sobre el PIM [MM03]. Esta implementación del programa en un lenguaje de programación determinado se conoce en ocasiones como *Plattorm Specific Implementation* (PSI).

Aunque en la guía de referencia de MDA del OMG [MM03] se realiza una separación clara entre los dos tipos de modelos manejados en MDA: PIM y PSM, algunos autores consideran que la distinción no siempre es posible [KWB03]. Esto es debido a que los modelos de las aplicaciones suelen contener información relacionada con la plataforma o tipo de plataforma sobre la que se ejecutarán. Por ejemplo, al especificar que un método de una clase es transaccional, puede considerarse que el modelo se convierte en específico a una plataforma tecnológica que soporte transacciones.

3.1.13. El modelo de plataforma

Suministra un conjunto de conceptos técnicos que representan los diferentes tipos de elementos que conforman la plataforma y los servidos ofrecidos por la misma. También especifica, para la utilización de un modelo especifico de plataforma, los conceptos que representan los diferentes tipos de elementos utilizados para especificar el uso de la plataforma por una aplicación.

Un modelo de plataforma especifica además los requisitos de conexión y uso de las partes de la plataforma y la conexión de una aplicación a la plataforma.

Un modelo de plataforma genérica puede establecer una especificación de un estilo particular de arquitectura.

3.1.14. Transformación

La transformación del modelo es el proceso de convertir un modelo en otro del mismo sistema. En MDA se contemplan dos tipos fundamentales de transformaciones (Figura 3.2):

- La transformación de un PIM en un PSM.
- La transformación de un PSM en el código fuente de la aplicación.

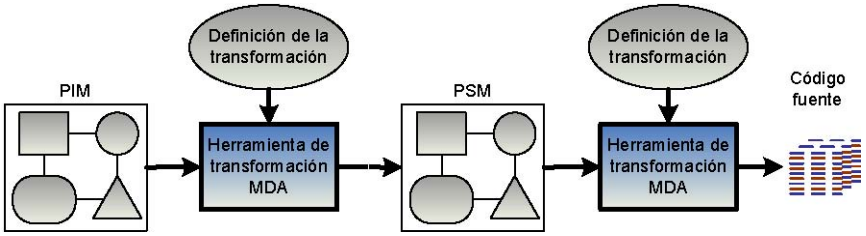


Figura 3.2. Transformaciones y herramientas de transformación.

En la Figura 3.2 se representa esta doble transformación. Una herramienta de transformación MDA recogerá el modelo independiente de la plataforma PIM y lo transformará en un modelo específico a la plataforma PSM. Para realizar esta transformación, hará uso de una metainformación que especifique cómo realizar la transformación. Esta metainformación se denomina "definición de la transformación" [KWB03] o "*mapping rules*" (reglas de mapeo) [MM03]. En general, suele utilizarse el término "*mapping*" (mapeo) para denominar "transformación".

Además de las definiciones de transformación, es habitual etiquetar o marcar los modelos para guiar los procesos de transformación. Por ejemplo, para diferenciar datos persistentes de datos temporales. O para distinguir entre procesos remotos o locales. Esta metainformación se denomina genéricamente como "marcas" (*marks*) [MSU+04]. Las marcas se utilizan tanto en los PIM como en los PSM.

Las dos transformaciones pueden ser realizadas por la misma herramienta de transformación o por herramientas distintas. Es precisamente cómo se aborda la construcción de dichas herramientas de transformación, junto con las definiciones de cómo se realizan las transformaciones, uno de los aspectos claves que diferencian las diferentes líneas de investigación en torno al paradigma MDA. De hecho, en la Guía de Referencia de MDA [MM03], la información que especifica la transformación se representa con una caja blanca vacía (Figura 3.3).

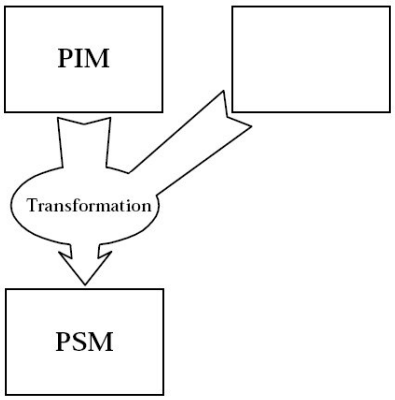


Figura 3.3. Definición de la información de la transformación.

3.1.15. Servicios difundidos

Son los servicios disponibles en una amplia gama de plataformas. MDA suministra habitualmente, un modelo independiente de la plataforma de servicios difundidos. Facilitará el mapeado para la transformación de los modelos, dibujando estos servicios difundido PIMs, a los modelos de plataforma especifica utilizando los servicios suministrados por la plataforma en concreto.

3.1.16. Implementación

Una implementación es una especificación, que suministra toda la información necesaria para la construcción del sistema y ponerlo operativo.

3.2. MODELO DE DESARROLLO

3.2.1. Problemas del Desarrollo Tradicional

En el proceso de desarrollo software de forma tradicional incluye las siguientes fases:

1. Recogida de requisitos
2. Análisis
3. Diseño
4. Codificación
5. Prueba
6. Despliegue

La Figura 3.4 esquematiza el proceso de desarrollo de software tradicional.

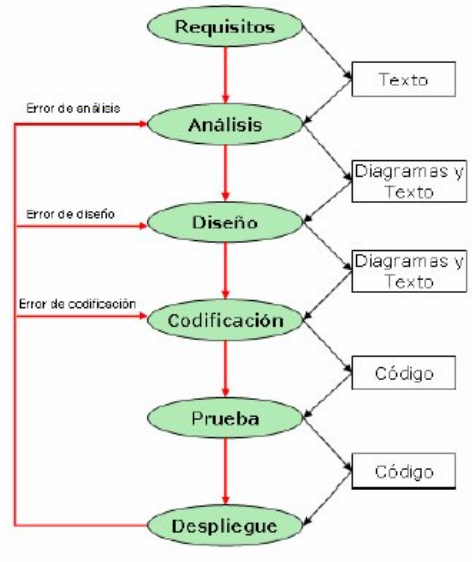


Figura 3.4. Proceso de desarrollo de Software Tradicional.

Durante los últimos años se han hecho muchos progresos en el desarrollo del software, que han permitido construir sistemas más grandes y complejos. Aun así, la construcción de software de la manera tradicional sigue teniendo múltiples problemas:

3.2.1.1. Productividad

El proceso tradicional produce una gran cantidad de documentos y diagramas para especificar requisitos, clases, colaboraciones, etc. La mayoría de este material pierde su valor en cuanto comienza la fase de codificación, y gradualmente se va perdiendo la relación entre los diagramas. Y más aún cuando el sistema cambia a lo largo del tiempo: realizar los cambios en todas las fases (requisitos, análisis, diseño...) se hace inmanejable, así que generalmente se realizan las modificaciones sólo en el código. ¿Entonces para qué perder el tiempo en construir los diagramas y la documentación de alto nivel? Lo cierto es que para sistemas complejos sigue siendo necesario. Lo que necesitamos entonces es un soporte para que un cambio en cualquiera de las fases se traslade fácilmente al resto.

3.2.1.2. Portabilidad

En la industria del software, cada año aparecen nuevas tecnologías y las empresas necesitan adaptarse a ellas, bien porque la demanda de esa tecnología es alta (es ``lo que se lleva''), bien porque realmente resuelve problemas importantes. Como consecuencia, el software existente debe adaptarse o migrar a la nueva tecnología. Esta migración no es ni mucho menos trivial, y obliga a las empresas a realizar un importante desembolso.

3.2.1.3. Interoperabilidad

La mayoría de sistemas necesitan comunicarse con otros, probablemente ya construidos. Incluso si los sistemas que van a interoperar se construyen desde cero, frecuentemente usan tecnologías diferentes. Por ejemplo, un sistema que use Enterprise JavaBeans necesita también bases de datos relacionales como mecanismo de almacenamiento de datos. Necesitamos que la interoperabilidad entre sistemas, nuevos o ya existentes, se consiga de manera sencilla y uniforme.

3.2.1.4. Mantenimiento y documentación

Documentar un proyecto software es una tarea lenta que consume mucho tiempo, y que en realidad no interesa tanto a los que desarrollan el software, sino a aquellos que lo modificarán o lo usarán más adelante. Esto hace que se ponga poco empeño en la documentación y que generalmente no tenga una buena calidad. La solución a este problema a nivel de código es que la documentación se genere directamente del código fuente, asegurándonos que esté siempre actualizada. No obstante, la documentación de alto nivel (diagramas y texto) todavía debe ser mantenida a mano.

3.2.2. Beneficios del MDA

MDA resuelve cada uno de los problemas del desarrollo tradicional expuestos en el apartado anterior:

3.2.2.1. Solución al Problema de la Productividad

En MDA el foco del desarrollo recae sobre el PIM. Los PSMs se generan automáticamente (al menos en gran parte) a partir del PIM. Por supuesto, alguien tiene que definir las transformaciones exactas, lo cual es una tarea especializada y difícil. Pero una vez implementada la transformación, puede usarse en muchos desarrollos. Y lo mismo ocurre con la generación de código a partir de los PSMs. Este enfoque centrado en el PIM aísla los problemas específicos de cada plataforma y encaja mucho mejor con las necesidades de los usuarios finales, puesto que se puede añadir funcionalidad con menos esfuerzo. El trabajo "sucio" recae sobre las herramientas de transformación, no sobre el desarrollador.

3.2.2.2. Solución al Problema de la Portabilidad

En MDA, la portabilidad se logra también enfocando el desarrollo sobre el PIM. Al ser un modelo independiente de cualquier tecnología, todo lo definido en él es totalmente portable. Otra vez el peso recae sobre las herramientas de transformación, que realizarán automáticamente el paso del PIM al PSM de la plataforma deseada.

3.2.2.3. Solución al Problema de la Interoperabilidad

Los PSMs generados a partir de un mismo PIM normalmente tendrán relaciones, que es lo que en MDA se llama puentes. Normalmente los distintos PSMs no podrán comunicarse entre ellos directamente, ya que pueden pertenecer a distintas tecnologías. Este problema lo soluciona MDA generando no solo los PSMs, sino también los puentes entre ellos. Como es lógico, estos puentes serán construidos por las herramientas de transformación, que como vemos son uno de los pilares de MDA.

3.2.2.4. Solución al Problema del Mantenimiento y Documentación

Como ya hemos dicho, a partir del PIM se generan los PSMs, y a partir de los PSMs se genera el código. Básicamente, el PIM desempeña el papel de la documentación de alto nivel que se necesita para cualquier sistema software. Pero la gran diferencia es que el PIM no se abandona tras la codificación. Los cambios realizados en el sistema se reflejarán en todos los niveles, mediante la regeneración de los PSMs y del código. Aun así, seguimos necesitando documentación adicional que no puede expresarse con el PIM, por ejemplo, para justificar las elecciones hechas para construir el PIM.

3.2.3. Modelo de Desarrollo con MDA

Si se emplea MDA para el desarrollo de software, este proceso (Figura 3.5) puede describirse como sigue:

- Paso 1: Construir el PIM
- Paso 2: Utilizar una herramienta de trasformación para generar uno o más PSM a partir de PIM
- Paso 3: Utilizar una herramienta para obtener el código

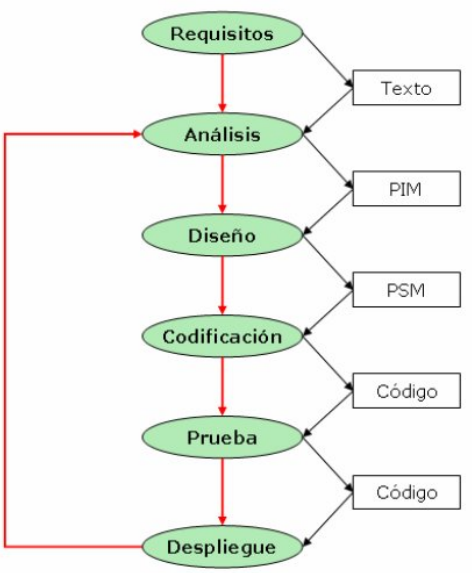


Figura 3.5. Ciclo de Vida.

Las tareas a realizar en este proceso podrían resumirse de la siguiente manera:

- Construcción de modelos: Modelo Independiente de la Plataforma (PIM), Modelo Específico de Plataforma (PSM) y Código fuente.
- Transformaciones: PIM en PSM y PSM en código.
- Especificaciones: Uno o más lenguajes estandar y bien definidos para desarrollar el PIM. En este contexto, se entiende que un lenguaje está bien definido si lo están su sintaxis y su semántica y es adecuado para ser directamente interpretado por el ordenador. Uno de más lenguajes estandar y bien definidos para desarrollar el PIM y Un lenguaje para escribir la definición de las transformaciones entre modelos.

- Herramientas: Implementan las transformaciones que se tienen que llevar a cabo.

3.3. VISIÓN ALTERNATIVA DE MDA

Una visión alternativa y práctica de MDA la encontramos en un manifiesto [BBI+04] elaborado por los principales investigadores de IBM trabajando en IVIDA, en donde se resume MDA como la combinación de tres ideas complementarias. La Figura 3.6 muestra estas tres ideas, que resumiremos a continuación:

Figura 3.6. Visión alternativa de MDA.

3.3.1. Representación directa

Se desplaza el foco del desarrollo del software del dominio de la tecnología hacia las ideas y conceptos del dominio del problema. Reduciendo la distancia semántica entre el dominio del problema y su representación permitimos un mayor acoplamiento de las soluciones a los problemas, logrando diseños más acertados e incrementando la productividad.

3.3.2. Automatización

Usar herramientas para mecanizar aquellas facetas del desarrollo del software que no dependen del ingenio humano. MDA incrementa la velocidad de desarrollo y reduce errores usando herramientas automatizadas para transformar modelos específicos de plataforma en código de implementación. Es lo mismo que hacen los compiladores para los lenguajes de programación tradicionales.

3.3.3. Estándares abiertos

Los estándares han sido uno de los mejores impulsores del progreso en la historia de la tecnología. Los estándares de la industria no solo ayudan a eliminar la diversidad gratuita, sino que también alientan a los vendedores a producir herramientas tanto para propósitos generales como para propósitos especializados, incrementando así las posibilidades del usuario. El desarrollo de código abierto asegura que los estándares se implementan consistentemente y anima la adopción de estándares por parte de los vendedores.

3.4. TRANSFORMACIÓN DE MODELOS

La transformación de modelos juega un papel clave en el desarrollo dirigido por modelos, puesto que serán un conjunto de transformaciones las que, partiendo de un conjunto de modelos que especifican un sistema, permitan conseguir el software ejecutable sobre una plataforma concreta. En este contexto, transformación de modelos hace referencia al proceso de convertir un modelo en otro modelo del mismo sistema [Heb05].

Utilizando la terminología de la iniciativa MDA, un proceso de transformación recibe como entrada un conjunto de modelos independientes de la plataforma (PIM) y un conjunto de reglas de transformación. Como producto de un proceso de transformación se obtiene un modelo específico a la plataforma (PSM). Esta estructura se denomina Patrón MDA [MM03].

Existen varias aproximaciones al problema de la transformación de modelos. Se comenzará analizando el estándar QVT propuesto por el OMG y algunas de sus implementaciones. Paralelamente al desarrollo de QVT han aparecido otras aproximaciones para la transformación de modelos. Entre éstas destacan las propuestas por VIATRA y por ATL, que actualmente presentan diferentes niveles de conformidad con QVT.

En este sentido, gracias a la separación que se realiza en la especificación de QVT de sintaxis abstracta, definida formalmente en base a un metamodelo, y concreta, es posible hacer evolucionar las sintaxis concretas de otros lenguajes de transformación para que se basen en la sintaxis abstracta del metamodelo de QVT.

3.4.1. QVT (Query/View/Transformation)

QVT es la propuesta del OMG para resolver el problema de la transformación de modelos. Se trata de un estándar para la definición de transformaciones sobre modelos MOF. El proceso de definición del estándar culminó a finales de 2005 con la primera versión final de la especificación [MOF05], en la que convergieron las 8 propuestas inicialmente presentadas.

Cuando surgió la iniciativa MDA, los estándares UML, XMI y MOF ya gozaban de un cierto grado de madurez y fueron soportados, en mayor o menor medida, por parte de los fabricantes de herramientas. Sin embargo, las reglas de transformación que implementan dichas herramientas eran definidas en torno a un conjunto de tecnologías propietarias no estándares, tales como el uso de plantillas, la generación de metaclases específicas y el uso de otros sistemas propietarios. La consecuencia es que un elemento tan importante de MDA como es la realización de los sistemas depende totalmente de la herramienta utilizada. Este es el problema que QVT pretende solucionar.

El estándar QVT define tres abstracciones fundamentales, que se corresponden con sus siglas [Heb05]:

- **Consultas** (*Queries*) Una consulta es una expresión que se evalúa sobre un modelo. Los resultados de una consulta son una o varias instancias de los tipos definidos en el modelo transformado, o en el propio lenguaje de consulta. Para la realización de consultas se utilizará un lenguaje de consultas.
- **Vistas** (*Views*) Una vista es un modelo obtenido en su totalidad a partir de otro modelo base. Las consultas son un tipo restringido de vistas.

- **Transformaciones** (*Transformations*) Una transformación genera un modelo a partir de otro modelo de origen. Ambos modelos podrán ser dependientes o independientes, según exista o no una relación que mantenga ambos sincronizados una vez se produzca la transformación. Las vistas son un tipo específico de transformación. Si se modifica una vista, la correspondiente transformación debe ser bidireccional para reflejar los cambios en en el modelo fuente. Las transformaciones se definirán utilizando un lenguaje de transformación. El lenguaje de transformación debe servir para generar vistas de un metamodelo, debe ser capaz de expresar toda la información necesaria para generar automáticamente la transformación de un modelo origen a uno destino, y debe además soportar cambios incrementales en un modelo origen que se ven reflejados en el modelo destino [MOF05].

De estas abstracciones se desprenden por lo tanto dos lenguajes, de consulta y de transformación, a los que se impuso como requisito que estuvieran definidos como metamodelos MOF 2.0. Un último requisito fundamental de la propuesta QVT es relativo a los modelos manipulados: todos los modelos manipulados por los mecanismos de transformación serán además instancias de metamodelos MOF 2.0.

Una vez descrito el planteamiento conceptual de QVT conviene describir la solución propuesta por la especificación finalmente adoptada [MOF05]. Como lenguaje de consulta se definen extensiones al estándar OCL 2. Para la especificación de transformaciones se propone una solución de naturaleza híbrida declarativa e imperativa. La parte declarativa se divide en una arquitectura de 2 capas, que sirve como marco para la semántica de ejecución de la parte imperativa.

Las dos capas de la parte declarativa son:

- **Relations**. Define un lenguaje declarativa para expresar relaciones entre modelos MOF. Este lenguaje soporta reconocimiento de patrones, la creación de plantillas de objetos y la creación implícita de las trazas necesarias para registrar los cambios que se producen cuando se transforman modelos.

- **Core.** Define un lenguaje declarativo de menor nivel de abstracción que *Relations* pero con la misma potencia. La especificación de QVT define las reglas que permiten mapear la semántica de *Relations* a la de *Core*, y dichas reglas las define en base a transformaciones descritas utilizando a su vez *Core*. En este lenguaje, los objetos de traza deben definirse explícitamente y sólo soporta reconocimiento de patrones sobre un conjunto plano de variables, no sobre objetos complejos como en *Relations*.

En cuanto a la parte imperativa de QVT, se definen dos mecanismos para invocar implementaciones imperativas de transformaciones desde los lenguajes *Relations* o *Core*:

- **Operational Mapping.** Se trata de un lenguaje estándar que permite definir procedimientos imperativos de transformación. Dichos procedimientos deben emitir los mismos modelos de traza que el lenguaje *Relations*. Se trata de un lenguaje procedural, definido en torno a un conjunto de extensiones del lenguaje OCL que permiten efectos laterales, e incluye construcciones imperativas tales como bucles y condiciones.
- **Black-box MOF Operation.** No se trata de un lenguaje, sino de un mecanismo que permite enlazar código escrito en cualquier lenguaje. Para ello, se utiliza el lenguaje *Relations* para derivar una operación MOF de una transformación. Esta operación será el punto de enlace con el lenguaje deseado, que deberá definir la operación a ejecutar con la misma signatura, y soportar un enlace con la implementación de MOF que se esté utilizando.

QVT define por lo tanto tres DSL: *Relations*, *Core* y *Operational Mappings*. Para todos ellos define una sintaxis abstracta en base a sus metamodelos MOF 2. La semántica de su ejecución se define mediante descripciones en texto plano. Para cada uno de ellos se proporciona una sintaxis textual concreta definida mediante gramáticas EBNF [Cue01]. Para el lenguaje *Relations* se define además una notación gráfica específica. Finalmente, para los lenguajes *Relations* y *Operational Mappings* se definen diversas extensiones de OCL 2.

3.4.1.1. El Lenguaje Relations

El lenguaje *Relations* ofrece una aproximación declarativa para la especificación de transformaciones. Dado un par de modelos candidatos para la transformación, que deberán ser instancias de metamodelos MOF 2.0, ésta quedará definida como un conjunto de restricciones que los elementos de los modelos deben satisfacer.

Estas restricciones constituyen el concepto de *relación* del lenguaje *Relations*: se definen mediante dos o más dominios y mediante una pareja de predicados *when* (cuándo) y *where* (cómo). Los dominios especifican, mediante un patrón, qué elementos de los modelos candidatos participarán en la transformación. Por su parte, la cláusula *when* especifica la relaciones que determinan cuándo la relación indicada debe sostenerse; y la cláusula *where* especifica la condición que deben satisfacer todos los elementos de modelos que participan en la relación [MOF05].

3.4.1.2. El Lenguaje Operational Mapping

El lenguaje *Operational Mapping* permite definir transformaciones utilizando bien una aproximación imperativa o bien una aproximación híbrida, complementando las transformaciones relacionales declarativas con operaciones imperativas que implementen las relaciones.

Una transformación operacional define una transformación unidireccional expresada de forma imperativa. Se compone de una signatura que indica los modelos involucrados en la transformación y de una definición de la ejecución de la transformación. En la práctica, una transformación operacional se trata de una entidad instanciable con propiedades y operaciones.

3.4.1.3. Implementaciones de QVT

Debido al poco tiempo que ha transcurrido desde que el OMG liberase la primera especificación final de QVT [QVT05] no existe disponible ninguna implementación completa del estándar.

3.4.1.3.1. Borland Together Architect 2006

Dentro de las herramientas comerciales que soportan parcialmente QVT se encuentra Borland Together Archiect [BTA06]. La herramienta implementa un motor QVT pero no soporta el lenguaje declarativo de *Relations*.

Sólo soporta las transformaciones operacionales (imperativas). Soporta además OCL en su versión 2, y lo extiende siguiendo las directrices de QVT. En cuanto a las operaciones de caja negra (*Black-box MOF Operation*), la herramienta permite importar e invocar sentencias Java desde las transformaciones QVT.

Para las transformaciones de modelos a texto (*text-to-model*) la herramienta utiliza el motor de plantillas JET [Pop03], del proyecto Eclipse [Ecl06a] el cuál se basa en EMF

3.4.1.3.2. SmartQVT

SmartQVT [BD06] es una implementación de código abierto de la especificación QVT realizada por France Telecom R&D. Al igual que Together Architect, se basa en metamodelos EMF para los modelos que intervienen en las transformaciones y únicamente considera el lenguaje *Operational Mapping*.

También utiliza EMF para especificar el metamodelo de QVT. Su funcionamiento se basa en una transformación en dos fases:

- En una fase inicial se traduce la sintaxis textual concreta del lenguaje operacional de QVT a instancias del metamodelo de QVT.
- En una segunda fase, se transforma el modelo de QVT en un programa Java basado en EMF que ejecuta la transformación. Para serializar el modelo QVT de entrada se utiliza XMI 2.0.

3.4.1.3.3. MOMENT

MOdel manageMENT [QHB+06] es un prototipo de motor de transformación que implementa el lenguaje *Relations* de la especificación QVT. En su implementación utiliza EMF para la especificación de metamodelos. A partir del código del lenguaje *Relations*, instancia el metamodelo QVT que implementa la herramienta. Como paso previo a obtener el modelo transformado en términos de EMF, se genera una especificación intermedia en el lenguaje Maude [CDE+03], aprovechándose de sus propiedades intrínsecas, como la concordancia de patrones, la parametrización y la reflexión.

3.4.2. VIATRA

VIsual Automated model TRAnsformation (VIATRA 2) es un framework de propósito general para dar soporte a todo el ciclo de vida de los procesos de transformación de modelos. Desarrollado desde 1998 en la Universidad de Tecnología y Económicas de Budapest, ha sido utilizado en diversos proyectos de ámbito europeo en el ámbito de los sistemas empotrados [VIA06].

VIATRA se define en torno a un conjunto de especificaciones propias, al margen de las planteadas por el OMG. Define un lenguaje de metamodelado propio denominado VPM [VP03]. El metamodelo definido es mucho más sencillo que MOF, definiendo dos entidades básicas: `Entity` y `Relation`. Las relaciones pueden ser agregaciones y pueden tener multiplicidades. Además se definen dos relaciones especiales, la generalización y la relación "ser instancia de". La compatibilidad con otros lenguajes de metamodelado, como MOF, se plantea utilizando *plugins* de conversión.

Al igual que MOF, VPM se define en base a un metamodelo formal que constituye su sintaxis abstracta. Se definen dos sintaxis concretas sobre ella, una gráfica similar a UML y una textual denominada *Viatra Textual (Meta)Modeling Language* (VTML).

En cuanto a las transformaciones, además de las transformaciones modelo a modelo, VIATRA considera las transformaciones modelo a texto. Para las transformaciones entre modelos se utilizan patrones de grafos para definir restricciones y condiciones en los modelos. Para especificar las transformaciones se utilizan dos tipos de reglas:

- Reglas de transformación de grafos: para definir manipulaciones elementales de modelos utilizando la técnica de transformación de grafos [EEK+99].
- Máquinas abstractas de estados: Para la descripción de estructuras de control.

Para la especificación de ambos tipos de reglas se ha creado un lenguaje denominado *Virtual Textual Command Language* (VTCL) que es textual y tiene como objeto servir de base para la construcción de editores gráficos que se edifiquen sobre él. VTCL incluye algunas construcciones básicas del lenguaje *Abstract State Machine* (ASM) [BS03] para las construcciones típicas de un lenguaje imperativo.

En cuanto a las transformaciones de modelos a texto se utiliza un lenguaje de plantillas propio denominado *Virtual Textual Template Language* (VTTL), similar al motor Velocity del proyecto Apache Jakarta [Apa06] pero sustituyendo Java por el lenguaje ASM.

3.4.3. ATL

ATLAS Transformation Language (ATL) [INR06] es la respuesta dada por el instituto de investigación *Institut nacional de recherche en informatique et en automatique* (INRIA) a la propuesta de *Request for Proposal* (RFP) para QVT lanzada por el OMG en 2002. Actualmente se encuadra dentro del proyecto *Generative Modeling Technologies* (GMT) de la Fundación Eclipse [Ecl06b], que es la rama de proyectos de investigación abierta en el seno del *proyecto Eclipse Modeling Project*.

ATL define un lenguaje de transformación de modelos especificado tanto por su metamodelo como por una sintaxis concreta textual. Al igual que QVT tiene una naturaleza híbrida entre declarativa e imperativa. Para la manipulación de modelos se basa en MOF.

Como parte del proyecto se ha definido una máquina virtual orientada a la transformación de modelos, con el fin de aumentar la flexibilidad de la arquitectura [INR05]. Las transformaciones especificadas en ATL son transformadas a un conjunto de primitivas básicas de transformación de la máquina virtual. De este modo, las transformaciones ATL son ejecutables porque existe una transformación específica del metamodelo ATL al bytecode de esta máquina virtual. Además, puede extenderse el lenguaje traduciendo las nuevas construcciones al bytecode de la máquina virtual.

La sintaxis concreta de ATL es similar a la definida por QVT, aunque ambos lenguajes no son interoperables por las divergencias en sus metamodelos.

3.4.4. Motor de Transformaciones de BOA 2

El framework BOA 2 es la evolución del framework BOA [PGR+04], una propuesta de framework MDA desarrollado por la empresa Open Canarias. Mientras que en la versión 1 del framework se realizaba una única operación de transformación basada en la traducción de modelos UML serializados mediante XMI utilizando plantillas *eXtensible Stylesheets Language Transformations* (XSLT), en la versión 2 se ha planteado reescribir el motor de transformación implementando la especificación QVT [PGS+05].

El motor de transformación de BOA 2 se centra en la transformación modelo a modelo manipulando modelos EMF. En la presentación del framework [PGS+05] se plantea una implementación de un motor QVT con soporte para características avanzadas como trazabilidad, compilación incremental, transaccionalidad de las transformaciones y bidireccionalidad. Para implementar la bidireccionalidad, propone utilizar observadores [GHJ+95] EMF para registrar la información de cambio de los modelos, puesto que existen transformaciones inherentemente unidireccionales en las que se pierde información tras su ejecución.

Con el fin de independizar el motor de transformación de los lenguajes de transformación que soporte el framework BOA 2 propone un lenguaje de transformaciones de bajo nivel denominado *Atomic Transformation Code* (ATC). Se trata de un lenguaje imperativo neutro de bajo nivel firmemente alineado con EMF. Se prevé una arquitectura basada en máquina virtual que, tras la estabilización del lenguaje, evolucione a una solución donde se transforme el código ATC en código Java por razones de eficiencia. Sobre ATC, se prevé dar soporte a los lenguajes *Relational* y *Operational Mapping* de QVT.

Para la generación final de código de las aplicaciones a partir de los modelos transformados el framework BOA 2 utiliza el motor de plantillas JET [Pop03].

3.4.5. M2M

Model-To-Model Transformation (M2M) es un proyecto Open Source encuadrado dentro del grupo de proyectos de la iniciativa Eclipse Modeling Project de la fundación Eclipse. En su estado actual, es una propuesta de proyecto aceptada en espera de las contribuciones de las compañías Borland, Compuware e INRIA para comenzar su desarrollo.

El planteamiento del proyecto es desarrollar un *framework* de lenguajes de transformación de modelo a modelo. Busca ofrecer una arquitectura modular donde puedan configurarse diferentes motores de transformación. Está previsto implementar 3 motores de transformación: para ATL, para el lenguaje *Operational Mapping* de QVT y para los lenguajes *Core* y *Relational* de QVT.

El proyecto busca por lo tanto salvar las diferencias existentes entre QVT y ATL a la hora de ejecutar transformaciones elevando el nivel de abstracción. Desde el punto de vista de esta tesis, una aportación clave de este proyecto será la de proporcionar una implementación open source del estándar QVT. Esta apuesta responde al propósito perseguido por el *Eclipse Modeling Project* de apostar fuertemente por los estándares existentes, si bien el único motor de transformación implementado hasta la fecha se basa en ATL.

3.4.6. Sistemas de Generación de Código

En los apartados anteriores se han estudiado diferentes propuestas y tecnologías de transformación de modelos. Todos ellos se centran en la transformación que el patrón MDA denomina de PIM a PSM, es decir, transformaciones de modelo a modelo. Para completar el ciclo de vida de un proceso de transformación que obtenga como producto una aplicación ejecutable es necesario ejecutar una transformación de modelo a texto. Es decir, a partir de los modelos transformados es necesario obtener el código fuente final que se ejecutará sobre la plataforma de destino.

La forma más conocida de generador de código son los compiladores tradicionales [Cue98] que reciben como entrada el código fuente creado por el desarrollador. Lo que se plantea en el desarrollo dirigido por modelos es elevar el nivel de abstracción respecto al modelo de desarrollo tradicional, siendo los artefactos de primer nivel que manejarán los desarrolladores modelos. El código fuente, junto con todos los descriptores y ficheros de configuración necesarios para desplegar la aplicación sobre la plataforma de destino, será ahora el producto obtenido a partir de los modelos manejados por los desarrolladores. Se eleva el nivel de abstracción del código generado puesto que éste se edificará sobre la plataforma tecnológica utilizada, cuya existencia responde a su vez a una necesidad de acercar los artefactos manejados al desarrollador humano.

Para enumerar los tipos de generadores de código existentes se utilizará una variante de la clasificación que realiza Jack Herrington en [Her03], prescindiendo de la generación de clases parciales y de la generación de capas en una arquitectura de n-capas, que se considerarán un caso particular de los anteriores:

- Code munging: Este tipo de generadores de código toma determinados aspectos del código de entrada y genera uno o más ficheros de salida. Un ejemplo típico son los generadores de documentación a partir de los comentarios del código fuente de un programa.

- Expansor de código en línea (*inline-code expander*). Este tipo de generadores toma como entrada código fuente que contienen algún tipo especial de marcado que el expansor reemplaza con código de producción cuando crea los ficheros de salida. Un ejemplo son los lenguajes incrustados dentro de un lenguaje anfitrión. Dentro de esta ámbito se encuentran los sistemas que permiten incrustar SQL en un lenguaje de propósito general, como SQLj [CSH+98]. Un tipo particular de generador de código dentro de esta categoría son los motores de plantillas (*template engine*). En este caso, los ficheros de entrada del generador son plantillas que combinan contenido estático que es enviado directamente al fichero de salida con contenido dinámico que es interpretado en un proceso específico. El resultado de esta interpretación puede tener un propósito general, aunque suele orientarse a la generación dinámica de código en función de los metadatos que guían el proceso de generación. Por su importancia, se analizarán en la próxima sección.
- Generación de código mezclado (*mixed-code generation*) Se trata de una variante del anterior modelo en el cual el fichero de código de salida sobreescribe al fichero de origen. En [Her03] se incluye un ejemplo de generador de este tipo que permite generar el código de tests unitarios en programas escritos en C.

Tal y como se ha analizado en los apartados previos, los procesos de transformación son los suficientemente complejos como para desaconsejar su acometida en una única etapa. Es decir, el planteamiento correcto no es tomar los modelos que sirven de entrada para la aplicación y generar todo el código necesario a partir de ellos. Es necesario una etapa media de transformación modelo a modelo. Esta etapa se justifica por la necesidad de realizar transformaciones entre diferentes niveles de abstracción (por ejemplo, entre el ámbito de la programación orientada a objetos y el ámbito de una plataforma tecnológica), así como entre diferentes dominios (por ejemplo, entre el mundo de los objetos y el mundo relacional).

Así pues, la entrada de la herramienta generadora de código no serán los modelos de análisis inicialmente elaborados. Estos deberán ser sometidos a sucesivas transformaciones que desemboquen en modelos detallados para cada dominio y entorno tecnológico asociado en la aplicación final. Así pues, los modelos que finalmente recibirán los generadores de código tendrán el suficiente nivel de detalle como para no exigir una lógica de generación compleja. Esta es la razón por la que el estándar QVT no contempla la traducción modelo a texto. Las implementaciones analizadas de QVT, VIATRA y ATL utilizan diferentes sistemas de generación de código, todos ellos basados en motores de plantillas. La idoneidad de esta tecnología de generación se justifica en su capacidad para definir esqueletos de código parametrizables, sin limitar la lógica de aplicación que puede ser necesaria para decisiones complejas durante la generación.

3.4.6.1. Motores de plantillas

Un motor o procesador de plantillas es un programa que permite combinar una o varias plantillas con un modelo de datos para obtener uno o más documentos de salida.

Un motor de plantillas recibe como entradas:

- Un modelo de datos, que constituirá el contexto de información accesible desde la plantilla.
- Las plantillas, que constituirán la fuente de la traducción. Harán uso de la información contenido modelo de datos para su posterior procesamiento por el motor. Este uso puede consistir simplemente en referenciar parámetros del modelo de datos que se sustituirán por su valor cuando la plantilla se procese; o puede ser un uso más complejo, incluyendo el control del flujo de generación basado en la información del modelo.

Los motores de plantillas han sido usados profusamente para la generación dinámica de código en el ámbito de las aplicaciones Web. Se han desarrollado diversos motores de scripting que se ejecutan en el lado del servidor y que permiten generar dinámicamente el código que se sirve al cliente que accede con un navegador Web. En este área se encuentran *Java Server Pages* (JSP) para J2EE; *Active Server Pages* (ASP) dentro de ASP.NET [TL02] o el motor de PHP [TLM06]. El fundamento de todas estas tecnologías se basa en poder introducir código escrito en los lenguajes soportados por la plataforma correspondiente mediante secuencias de escape dentro de las plantillas que contienen el código estático, típicamente HTML.

Dentro de los motores de plantillas de propósito general destacan Velocity dentro del proyecto Apache Jakarta [Apa06] y JET de la Fundación Eclipse [Pop03]. Éste último ha sido utilizado para completar el ciclo de vida de transformación de algunas herramientas analizadas en los apartados anteriores. Otras herramientas optan por desarrollar un motor de plantillas propio específico para sus necesidades. Este es el caso de VTTL en VIATRA o XPand en la herramienta *openArchitectureWare*.

El OMG publicó en 2004 una RFP para aceptar proposiciones con el objeto de estandarizar un sistema de traducción de modelos a texto bajo la denominación de *MOF Model to Text Transformation Language*. La propuesta se basa en una aproximación basada en plantillas. Las plantillas permiten secuencias de escape que permiten extraer datos de los modelos. Estas secuencias consisten en expresiones especificadas sobre los elementos de metamodelado. Utiliza consultas OCL como el mecanismo principal para seleccionar y extraer información de los modelos.

3.5. IMPLEMENTACIONES DE MODEL DRIVEN ARCHITECTURE (MDA)

El auge de MDA ha impulsado en los últimos años el desarrollo de tecnología tanto de carácter comercial como de propósito general. Algunas de estas herramientas implementan total o parcialmente los conceptos de MDA.

A continuación se hace una enumeración y breve descripción de algunas de estas implementaciones:

- AndroMDA [And]

Se trata de una herramienta de código abierto basada en templates para la generación de código J2EE desde modelos UML/XMI. Utiliza VTL (Velocity Template Engine) como lenguaje scripting y NetBeans MDR como API de los modelos.

- ArcStyler [Arc]

Herramienta comercial de Interactive Objects. Utiliza MOF para soportar estándares como XMI y UML, y además JMI para el acceso al repositorio de modelos.

Integra herramientas de modelado (UML) y desarrollo (ingeniería inversa, explorador de modelos basado en MOF, construcción y despliegue) con la arquitectura CARAT que permite la creación, edición y mantenimiento de cartuchos MDA (MDA-Cartridge) que definen transformaciones.

- ATLAS Transformation Language (ATL) [ATL]

Lenguaje de transformación desarrollado por el equipo de INRIA Atlas.

- Codagen Architect [Cod]

Producto comercial integrado con varias herramientas comerciales UML.

- GMT(Generative Model Transformer) [GMT]

Un proyecto de eclipse que proporciona tecnología para la transformación de modelos bajo la plataforma eclipse.

- MCC (Model Component Compiler) [MCC]

Producto comercial orientado a la generación de código para plataformas J2EE.

- Middlegen [Mid]

Generador de código dirigido por bases de datos basado en JSBC, Velocity, Xdoclet y ANT.

- ModFact [Mod]

Es un repositorio MOF y una máquina QVT para LIP6, esta basado en el lenguaje TRL.

- MTL engine [MTL]

Es una implementación de QVT desarrollada por INRIA TrisKel para NetBeans MDR y Eclipse EMF, basado en el lenguaje MTL

- OMELET [OME]

Es un proyecto Eclipse que trata de proporcionar un framework de carácter general que integre modelos, metamodelos y transformaciones.

- OpenMDX [Opea]

Un entorno MDA de código abierto que genera código para las plataformas J2EE y .NET

- OpenModel [Opeb]

Un framework basado en MOF/JMI para herramientas MDA.

- OptimalJ [Opt]

Esta herramienta de Compuware utiliza notación de patrones para definir las transformaciones PSM y MOF para soportar estándares como UML y XMI. Se trata de un entorno de desarrollo que permite generar aplicaciones J2EE completas a partir de un PIM.

- SosyInc [Sos]

Proporciona una herramienta de modelado y una máquina de transformación que genera GUI y server-side. Está basada en modelos OASIS/UML y reglas sobre la estructura de la aplicación y la lógica del negocio.

- UMT (UML Model Transformation Tool) [UMT]

Herramienta para la trasformación de modelos y la generación de código basada en especificaciones UML/XMI.

- VisualWADE [Vis]

Se trata de una herramienta desarrollada por el grupo de Ingeniería Web de la Universidad de Alicante. VisualWADE permite el diseño y generación automática de aplicaciones web siguiendo una aproximación MDD. Combina diagramas de dominio UML con nuevos modelos para representar la interacción con el usuario sobre un entorno hipermedia. El código intermedio generado es XML. En la versión actual, se proporciona un compilador de modelos que produce entregables en PHP a partir del código intermedio XML.

- XDoclet [XDo]

Herramienta open source basada en atributos para la generación de código J2EE. Aunque realmente no está basada en modelos, puede ser combinada con otras herramientas, como UMT, para lograr resultados basados en modelos.

3.6. ANÁLISIS DE IMPLEMENTACIONES DE MDAS

3.6.1. AndroMDA

AndroMDA [And] es un framework MDA de generación de código. El líder del proyecto es Matthias Bohlen siendo 15 los componentes del núcleo de desarrollo del proyecto en la actualidad. El proyecto se deriva de la herramienta UML2EJB creada por Matthias Bohlen en 2002 [Boh02].

En la actualidad el la arquitectura de la herramienta se encuentra sometida a una profunda revisión que verá la luz en su versión 4. En esta versión se plantean cambios que faciliten la evolución futura de la herramienta, debido a ciertas limitaciones que presenta la herramienta en su versión 3. Dado que la versión actual de la herramienta goza de una gran implantación tanto en la comunidad como en la industria de desarrollo de software, se comenzará analizando sus fundamentos. A continuación se describirán cuáles son los cambios arquitectónicos previstos para la versión 4.

3.6.1.1. AndroMDA versión 3

AndroMDA en su versión 3 es una herrramienta generadora de código a partir de modelos UML. La herramienta es capaz de generar código para las plataformas J2EE y .NET. Dentro de cada una de ellas, permite generar código para múltiples frameworks y tecnologías ampliamente utilizadas en la industria.

Para la manipulación de los modelos introducidos AndroMDA utilizaba la implementación de MOF (Capítulo 7) leyendo los modelos de entrada en formato XMI (Capítulo 8). Esto restringía el abanico de herramientas UML a utilizar pues MDR implementa la versión 1.4 de MOF y las herramientas de modelado actuales soportan MOF 2.0. Esta situación se ha solucionado en las últimas versiones introduciendo una fachada (*facade*) [GHJ+95] que sirve como punto de acceso a diferentes repositorios de metamodelado. De este modo, se da soporte también a metamodelos EMF desde la versión 3.2.

Para la tarea de generar código AndroMDA presenta una arquitectura modular basada en el concepto de cartucho (*cartridge*) [And06a]. Los cartuchos son un tipo especial de plugin que puede configurarse en la herramienta y en el que se delega la tarea de generar código a partir de los elementos los modelos de entrada. Los cartuchos especifican qué elementos del modelo serán procesados y los asocian a determinadas plantillas que determinan qué código será generado a partir de ellos. Para seleccionar dichos elementos se utilizan estereotipos UML así como condiciones que pueden inferirse del modelo en función de las propiedades de sus elementos.

Para el acceso a los elementos del metamodelo AndroMDA propone utilizar el patrón Fachada [GHJ+95]. A las fachadas de los elementos del metamodelo los denomina *metafacades* [And06b]. Estas fachadas encapsulan los detalles de la implementación del metamodelo utilizado y ofrecen una API orientada a objetos que permiten acceder a sus elementos desde las plantillas. Ésto permite centralizar la inteligencia y la responsabilidad de la generación de código en las fachadas, no el lenguaje del motor de plantillas.

En cuanto a los motores de plantillas utilizados AndroMDA Velocity [Apa06], si bien soporta un número arbitrario de motores de plantillas permitiendo configurar cualquiera que implemente un determinado interfaz. En la versión 3.2 se ha añadido soporte para el motor Freemarker [Fre06].

Un problema que surge con la generación de código en AndroMDA es la necesidad de modificar el código una vez este ha sido generado. Los desarrolladores de cartuchos han adoptado diferentes aproximaciones para resolver este problema. En algunos casos se generan automáticamente clases hijas de las generadas en las el desarrollador puede definir métodos para extender su funcionalidad. Otra alternativa es permitir al desarrollador reproducir manualmente la estructura de archivos generados en un directorio específico donde puede modificarlos a su gusto. Los archivos modificados sobreescriben a los generados a la hora de generar la aplicación. Ambas aproximaciones presentan serios inconvenientes. La primera, porque la redefinición de métodos limita mucho las posibilidades de extender la funcionalidad de la aplicación. La segunda, porque recae en el desarrollador la responsabilidad de mantener sincronizados los modelos de entrada y los artefactos de código generados.

Una de las principales virtudes de AndroMDA ha sido el pragmatismo que ha guiado el desarrollo del proyecto. Se han desarrollado cartuchos para generar código basado en los frameworks y tecnologías más extendidos y aceptados en la comunidad. De este modo, se han creado cartuchos para Spring [JHA+05], Hibernate [BK04], Struts [STR] y JSF [Man05].

Otra característica destacada de AndroMDA es que, en la generación de código, los cartuchos hacen uso de las mejores prácticas y patrones de diseño. Es decir, aunque el código se genera automáticamente, se busca que sea un código de calidad, que conforme una aplicación estructurada fácilmente comprensible por un desarrollador humano. Como ejemplo, pueden señalarse el uso de los patrones *Data Access Object* (DAO) y *Value Object* [ACJ01} en el código generado por los cartuchos de Spring e Hibernate.

Desde el punto de vista de una herramienta de desarrollo dirigido por modelos AndroMDA presenta, en su versión 3, ciertas carencias. Por un lado, se trata de una herramienta que contempla exclusivamente transformaciones de modelos a texto. Este enfoque dificulta la acometida de transformaciones complejas, para lo que se requieren múltiples transformaciones modelo a modelo.

Por otro lado, todos los cartuchos desarrollados hasta el momento se basan exclusivamente en modelos UML. En las áreas donde UML permite crear modelos con toda la información necesaria para su transformación a otros dominios es donde AndroMDA se muestra más útil. Un ejemplo son los modelos estáticos de clases que son transformados a tablas en el modelo relacional. A partir de los diagramas estáticos de clases, AndroMDA permite generar todo el código necesario para gestionar la persistencia de las entidades: los ficheros con la información de mapeo objeto-relacional de Hibernate; las clases Java que se corresponden con las entidades persistentes, que serán objetos POJO [Fow00] con métodos de acceso a las propiedades de la clase y la implementación de los métodos `equals()` y `hashCode()` así como las clases DAO que implementan operaciones `create`, `read`, `update and delete` (CRUD) para los objetos persistentes asociados a cada clase.

Sin embargo, existen muchas áreas donde los modelos UML no permiten especificar la información de entrada. Un ejemplo es la especificación de interfaces de usuario. En el cartucho de Struts [And06c] de AndroMDA, a partir de diagramas de actividad UML se puede especificar el comportamiento de la capa de presentación de una aplicación Web. Los diagramas de actividad se muestran adecuados para modelar el flujo de información entre pantallas, así como para indicar qué controladores serán los responsables de gestionar las peticiones realizadas por el usuario. A partir de los diagramas, AndroMDA permite generar toda la información de configuración necesaria para Struts, las vistas JSP y los controladores Java asociados. Se generan además determinadas clases donde el desarrollador puede introducir código manualmente. Sin embargo, si se desea personalizar el aspecto de la aplicación, la única alternativa disponible es copiar el código generado un directorio y modificarlo. De este modo, se está rompiendo el ciclo de vida de la transformación automática, puesto que el código del interfaz deja de ser generado si el desarrollador lo modifica. Además, como esta modificación es manual, es fácil llegar a una situación donde los modelos y los artefactos de código generados dejen de estar sincronizados.

3.6.1.2. AndroMDA versión 4

La arquitectura de AndroMDA se encuentra en la actualidad en un proceso de profunda revisión. El objetivo es acometer una serie de cambios que verán la luz con la versión 4 de la herramienta [Boh06]. Los principales objetivos que motivan esta revisión son:

- Soportar un número arbitrario de metamodelos, para que los lenguajes utilizados para construir los modelos de entrada no estén obligados a basarse en el metamodelo de UML.
- Soporte para procesos de transformación modelo a modelo. Como se vio en la sección 3.6.1.1 actualmente la herramienta sólo soporta un enfoque modelo a texto. AndroMDA hará uso de ATL [ATL] para implementar el motor de transformación, aunque permitirá configurar un número arbitrario de motores de transformación.
- Permitir utilizar cartuchos en cadena, sirviendo las salidas de un cartucho como entradas del cartucho siguiente. Se trata de orquestar el funcionamiento de los cartuchos utilizando el patrón Cadena de Responsabilidad [GHJ+95].

En la Figura 3.7 se muestra la propuesta de arquitectura para la versión 4 de AndroMDA. Los componentes de la herramienta se organizan según el patrón arquitectónico que propone dividir la arquitectura en capas [BMR+96]. Va a utilizarse esta estructura para analizar los principales componentes de la herramienta que son novedad en la versión 4 y que presentan interés desde el punto de vista de esta investigación.

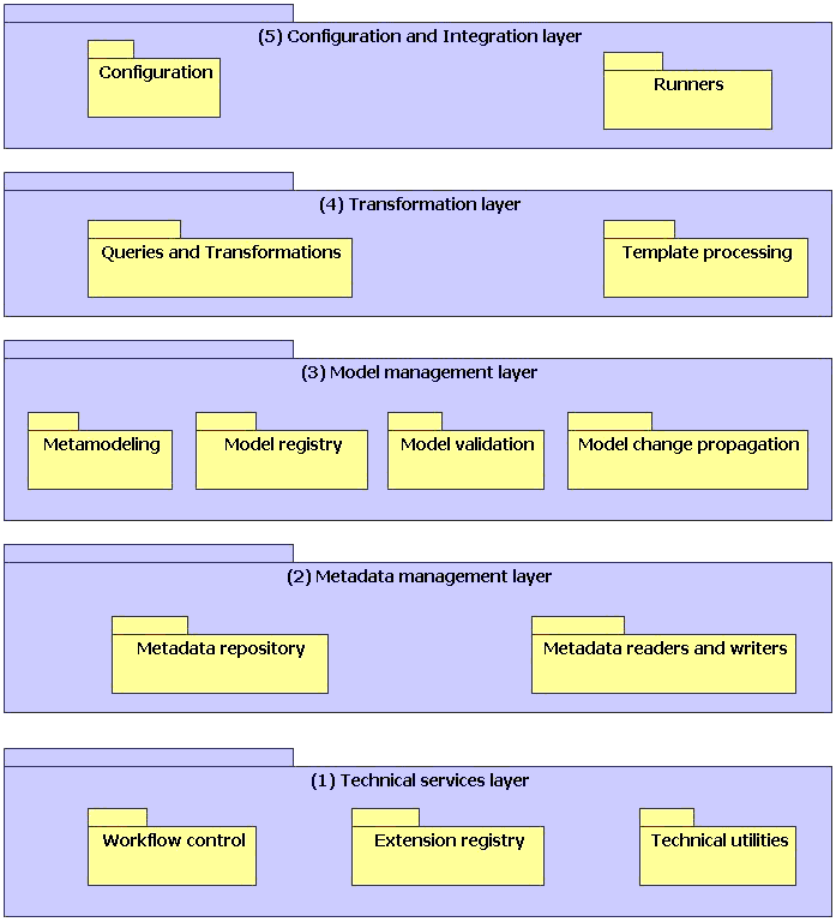


Figura 3.7. Arquitectura de AndroMDA versión 4.

En la capa de servicios técnicos se introduce un motor de workflow. Su función es orquestar las etapas de transformación. Su justificación radica en la complejidad de los flujos de tareas que aparece al someter múltiples modelos de entradas a sucesivas acciones de transformación, recibiendo unas etapas la salida que producen otras. Para la definición de procesos el motor de workflow utiliza un lenguaje XML específico de AndroMDA.

Dentro de la capa de gestión de metadatos se incluyen interfaces para abstracciones básicas comunes para importar y manejar metadatos. Se definen tres abstracciones básicas: un repositorio de metadatos (`IMetadataRepository`) del que se pueden obtener modelos (`IModel`) los cuales están compuestos de elementos de modelado (`IModelElement`). Para utilizar sistemas concretos de gestión de metamodelos, como EMF o MDR, deben crearse adaptadores [GHJ+95] para los interfaces (`IMetadataRepository`) e (`IModel`). Los metamodelos indican la sintaxis abstracta de los modelos, que deberá ser única. Para permitir procesar las diferentes sintaxis concretas definidas sobre ellos, se deben crear objetos que sean capaces de leer dicha sintaxis y transformarla a la sintaxis abstracta correspondiente.

En la capa de gestión de modelos se introduce sistema que permite validar modelos durante las transformaciones verificando las restricciones OCL que se definan sobre ellos. También se introduce un sistema encargado de propagar los cambios en los modelos al código. Este sistema tiene como fin atenuar el problema de la modificación manual del código generado que fue descrita en sección 3.6.1.1. Se propone una etapa posterior a la transformación de modelos en las que se detecte los cambios realizados y se generen comandos de *refactoring* [Fow99]. Estos comandos serán posteriormente por un motor que podrá configurarse con alguno de los motores de *refactoring* existentes. De esta manera los cambios introducidos en el modelo pueden ser propagados al código introducido manualmente.

Dentro de la capa de transformación se encuentran los componentes que implementan las transformaciones modelo a modelo y modelo a texto. Para las transformaciones modelo a modelo se usará ATL [ATL] aunque se permite configurar otros motores de transformación que deberán implementar un interfaz `IRunnable` que define tareas ejecutables que el motor de workflow puede orquestar. Este será también el interfaz que deberán implementar los motores de plantillas que se usarán para las transformaciones modelo a texto. Al igual que en la versión 3 se ofrecerán implementaciones para FreeMarker [Fre06] y Velocity [Apa06].

3.6.2. OpenArchitectureWare (oAW)

Es un framework [OAW06] generador de aplicaciones modular que sigue una filosofía MDD. Soporta un número arbitrario de modelos de entrada y ofrece una familia de lenguajes que permiten validar y transformar modelos, así como generar código a partir de ellos. Como parte del proyecto se ofrece un importante soporte para la plataforma Eclipse [Ecl06a] que incluye editores para todos los lenguajes que componen el framework. En la actualidad el framework se encuadra dentro del proyecto GMT de Eclipse [Ecli06b].

Un aspecto novedoso de cara radica en la propuesta del framework para la gestión de modelos y metamodelos, que difiere al planteamiento estudiado en otros sistemas. Para la especificación de metamodelos oAW implementa un sistema de tipos muy sencillo. Sobre dicho sistema de tipos, se define un lenguaje, muy similar a Java, de que permite definir y ejecutar expresiones. Este lenguaje sirve a su vez como base para el resto de lenguajes del framework. Una primera consecuencia de este diseño es que se presenta una sintaxis unificada para los diferentes lenguajes propuestos.

El sistema de tipos [Eff06b] define un conjunto de tipos primitivos. Se define además las entidades básicas del sistema de especificación de metamodelos, que es extremadamente simple: en el sistema de tipos se define la metaclase Tipo (`Type`), que puede tener Propiedades (`Property`) y Operaciones (`Operation`). Además un tipo puede heredar de uno o varios tipos (se permite herencia múltiple).

La principal novedad de oAW radica en que se permite configurar el sistema de tipos con un número arbitrario de implementaciones de metamodelos. De este modo, cuando, desde alguno de los lenguajes definidos por el framework, se haga referencia a un tipo, esta petición será traducida a la implementación concreta utilizada. Los tipos del metamodelo definidos en oAW sirven como enlace con los equivalentes de la implementación utilizada. Por ejemplo, una `Operation` de oAW se corresponde con una `MOperation` de MOF. Las implementaciones de metamodelos que pueden integrarse en el sistema de tipos por defecto son: EMF, la implementación de UML sobre EMF realizada en el proyecto Eclipse, y una implementación basada en Java que utiliza el sistema de introspección de Java [JCR97] para implementar el acceso a los metatipos y a sus elementos.

Sobre el sistema de tipos descrito se construye un lenguaje de expresiones denominado *Expressions* [Eff06a]. El lenguaje es una mezcla de Java y OCL. Este lenguaje sirve como base para definir el resto de lenguajes del framework.

El framework incluye un lenguaje para definir restricciones sobre modelos denominado *Check* [Eff06a]. Se trata de un lenguaje declarativo, similar a OCL, que permite asociar condiciones a los modelos que deben verificarlas.

Para el problema de la transformación de modelos el framework define un lenguaje específico denominado *Extend* [Eff06b]. Se trata de un lenguaje de corte funcional. El lenguaje fue en principio diseñado para permitir extender de los modelos de una forma no intrusiva. Para realizar esta extensión, permite definir librerías de operaciones basadas en Java o en el lenguaje *Extensions*, que son añadidas a los modelos.

A partir de la versión 4.1 de la herramienta, se incluyó en el lenguaje *Extend* la capacidad de definir transformaciones de modelos. Se introdujo una construcción especial denominada *create extension*. Se trata de operaciones que reciben como parámetro el modelo a transformar y devuelven como resultado el objeto transformado. El cuerpo de estas extensiones puede especificarse, como cualquier otra extensión, utilizando el lenguaje Java o utilizando el lenguaje *Expressions*.

El lenguaje *Extend* permite realizar transformaciones modelo a modelo. Par las transformaciones modelo a texto se define un lenguaje basado en plantillas denominado *Xpand2* [Eff06d]. El lenguaje soporta características avanzadas como el polimorfismo paramétrico. Si hay dos plantillas con el mismo nombre definidas para dos metaclases que heredan de la misma superclase, *Xpand2* usa la plantilla de la subclase correspondiente en el caso que ésta sea referenciada como la superclase. Además, permite extender las plantillas de forma no intrusiva utilizando programación orientada a aspectos [KLM+97]. Incluye construcciones para introducir código externo en determinados puntos de una plantilla. El motor se encarga de tejer la plantilla final.

Para orquestar todo el proceso de transformación, la herramienta incluye un motor de workflow [EV06a]. Para la especificación de los procesos se utiliza un lenguaje XML específico de oAW. El motor se basa en el patrón de inyección de dependencias [Fow04] para configurar los componentes que participan en el flujo de procesos. El motor contiene toda la información necesaria para guiar el proceso de generación. En su entrada se indican cuáles son los modelos y metamodelos de entrada, los ficheros de restricciones sobre dichos modelos, así como las transformaciones a las que éstos son sometidos.

Utilizando el framework oAW se ha construido otro framework, denominado *Xtext*, para desarrollar lenguajes DSL textuales [EV06b]. Este framework permite definir un DSL a partir de su gramática EBNF [Cue98]. A partir de esta entrada, *Xtext* genera automáticamente

- El meta modelo que representa la sintaxis abstracta del lenguaje definido.
- Un analizador sintáctico (*parser*) que permite procesar la sintaxis concreta de un modelo especificado mediante el DSL e instanciar el AST correspondiente. Para la generación del parser se utiliza Antlr [Ant].
- Un editor para Eclipse específico para el DSL creado.
- Los ficheros de restricciones en lenguaje *Check* que permiten definir restricciones semánticas al lenguaje.

3.6.3. Software Factories

Software factories representa la apuesta de Microsoft para el desarrollo dirigido por modelos. El origen de esta iniciativa se sitúa en una reacción a UML como lenguaje para modelar todos los componentes de un sistema. En su lugar, se apuesta por utilizar intensivamente lenguajes específicos de dominio (DSL) para especificar los diferentes componentes comprendidos en una aplicación.

El término *factory* es una referencia al objetivo que, en última instancia, persigue esta propuesta: industrializar el desarrollo de software [GSC+04]. Esto significa conseguir un proceso de desarrollo que presente características ya presentes en otras industrias con muchos más años de madurez que el software. Entre estas características se encuentran la capacidad de personalizar y ensamblar componentes estándares para producir variantes de un mismo producto; estandarizar, integrar y automatizar los procesos de producción; desarrollar herramientas extensibles y configurables que permitan automatizar las tareas repetitivas. Lo que se busca es conseguir establecer líneas de producción que automaticen la creación de productos.

Al igual que otras iniciativas en el campo del desarrollo dirigido por modelos, *software factories* utiliza modelos formales para la definición de los programas. Los modelos podrán especificarse con un número arbitrario de sintaxis concretas que se edificarán sobre una sintaxis abstracta. La principal diferencia radica en que, en primer lugar, se renuncia a disponer de un lenguaje de metamodelado común para la especificación de los metamodelos de los lenguajes utilizados.

En [GSC+04] se discuten las características de las gramáticas libres de contexto [Cue01] y metamodelos para especificar la sintaxis abstracta de los lenguajes. Se concluye que ambas aproximaciones deben ser combinadas. Por otra parte, se hace énfasis en la necesidad de utilizar lenguajes específicos de dominio (DSL), gráficos o textuales, desarrollados expresamente para la parte del sistema que se desea modelar.

Aunque en los documentos de presentación de la iniciativa se critica continuamente el intento de utilizar UML para modelar todos los aspectos de un sistema [Coo04][GSC+04], la necesidad de crear lenguajes específicos de modelado ya había sido resaltada en la iniciativa MDA [Fra03]. Ésta fue la principal razón del desarrollo de MOF, que es el estándar alrededor del cual giran el resto de especificación del OMG que comprenden la iniciativa MDA. Sin embargo, la propuesta de Microsoft no incluye un lenguaje de especificación de metamodelos único, aunque reconoce la importancia de los metamodelos.

Alrededor de los lenguajes específicos de dominio, la propuesta de *software factories* comprende un conjunto de conceptos que conviene analizar. Para dicho análisis se partirá de la definición de *software factory* dada en [GSC+04] por los creadores de la propuesta:

> *"A software factory is a software **product line** that configures extensible tools, processes, and content using a software factory **template** based on a software factory **schema** to automate the development and maintenance of variants of an archetypical product by adapting, assembling, and configuring **framework-based components**"*

En la definición se han resaltado los principales componentes de la iniciativa. Se identifican los frameworks como la construcción que liga los modelos al código. Los frameworks comprenden el código que implementa los aspectos que son comunes a un dominio, y exponen los elementos del dominio que varían como puntos de extensión del framework [Coo04]. En *software factories*, el papel de los modelos es definir cómo se extienden dichos puntos.

Otro componente destacado en la iniciativa son los patrones. Los patrones se entiende que son modelos parametrizables que comprenden un conjunto de reglas acerca de cómo pueden combinarse, total o parcialmente, esos con otros [Coo04]. Con este planteamiento, lo que se desea desarrollar es una tecnología que permita ensamblar modelos específicos, frameworks y patrones para construir aplicaciones. De este enfoque surge el concepto de línea de productos.

Al desarrollar una línea de productos se busca capturar sistemáticamente el conocimiento acerca de cómo producir los componentes de la aplicación, exponiéndolo a través de activos de producción. Los activos de producción comprenden activos de implementación, tales como lenguajes, patrones y frameworks; y también comprenden activos de proceso, que serán microprocesos que indican cómo utilizar diversos activos específicos de implementación y herramientas para automatizar partes del proceso [GSC+04].

El concepto de *factory schema* es de vital importancia en la iniciativa. Una *factory schema* se define como un grafo dirigido. Sus nodos representan perspectivas o puntos de vista del sistema desde los cuales se desea desarrollar algún aspecto del sistema [GSC+04]. Por ejemplo, un punto de vista de un sistema puede ser su arquitectura de requisitos. Una vez identificados los puntos de vista del sistema, se utiliza para identificar y agrupar los artefactos que deben desarrollarse para crear el producto software. Las conexiones del grafo representan relaciones entre los diferentes puntos de vista y se denominan *mappings*.

Las relaciones entre los puntos de vista realmente definen las relaciones entre los artefactos en ellos agrupados. Un *mapping* encapsula el conocimiento acerca de cómo implementar los artefactos descritos por un punto de vista en términos de los artefactos descrito por otro. Los *mappings* pueden ser directos o inversos, según cuál sea la dirección de la derivación. La derivación puede además ser parcial o completa.

Desde el punto de vista de esta tesis, los *mappings* que resultan interesantes son los computables. Con su ejecución se pueden generar, total o parcialmente, los artefactos descritos en el punto de vista de destino a partir de los artefactos contenidos en el punto de vista de origen. Para definir un *mapping* computable, deben utilizarse definiciones formales de los puntos de vista de origen y de destino, o lo que es lo mismo, deben utilizarse lenguajes formales para expresarlos y permitir su computación. El concepto de transformación en *software factories* surge precisamente de la computación de los *mappings*.

Del concepto de *factory schema* se deriva el de *factory template* que es una instancia de una *factory schema*. Comprende la implementación de los DSL, los patrones, los frameworks y las herramientas.

En [GSC+04] se discuten en profundidad las etapas que comprende un proceso de creación de software factories. El proceso se descompone en dos macroetapas. En la primera se construye el *factory schema*, lo que implica realizar un análisis y un diseño de la línea de productos. En la segunda etapa se construye la *factory template* que instancia la línea de productos software.

La propuesta de software factories ha sido objeto de confrontación entre miembros del OMG y miembros de Microsoft. En *The MDA Journal* se produjo un caluroso debate entre Steve Cook, antiguo miembro del OMG ahora en Microsoft, y Michael Guttman. Steve Cook, además de presentar la propuesta de *software factories*, criticaba al OMG por su propuesta de utilizar UML para modelar todos los aspectos del software. Por su parte, Michael Guttman acusaba a Microsoft de tener como única motivación en su iniciativa el utilizar formatos propietarios, ignorando los estándares existentes del OMG.

3.6.4. Borland Together Architect Edition

Borland Together Architect Edition [BTA06] es un entorno de desarrollo que se integra con los entornos Eclipse y Visual Studio .NET. El entorno cuenta, desde su versión 2006, con características de MDA.

Desde el punto de vista de esta tesis, este entorno resulta interesante porque introduce, en un producto comercial listo para producción, un conjunto de herramientas que comprenden una solución MDA integral. Comprende un diseñador de diagramas UML 2 para construir los modelos, un motor de transformación modelo a modelo basado en la especificación QVT en su versión previa al estándar finalmente adoptado e integra el motor de plantillas JET [Pop03] para la generación de código (transformación modelo a texto).

En su implementación, la herramienta realiza una implementación propia de MOF que, en su versión para Eclipse, integra con EMF. En cuanto a la implementación de QVT, aunque actualmente es cerrada, se prevé que sea liberada como consecuencia de la colaboración de Borland en el proyecto MTOM de Eclipse.

4. PATRONES DE DISEÑO

Los patrones de diseño hicieron su aparición con la publicación del libro Design Patterns [GHJ+95] en el que se recogían 23 patrones diseño comunes.

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.

En palabras de Christopher Alexander, "cada patrón describe un problema que se repite una y otra vez en nuestro entorno, a la vez que describe una solución general a dicho problema, de modo que ésta pueda ser aplicada más de un millón de veces sin tener que volver a hacer lo mismo dos veces" [Ale77]. Tomando como fuente de inspiración esa definición de patrón que Alexander aplicó en la forma de soluciones arquitectónicas al diseño de edificios y ciudades, los patrones de diseño software describen soluciones simples y elegantes a una serie de problemas concretos que tienen lugar en el diseño orientado a objetos [GHJ+95].

4.1. ELEMENTOS BÁSICOS

En términos generales, un patrón tiene cuatro elementos esenciales:

1. El nombre del patrón permite describir, en una o dos palabras, un problema de diseño, así como sus soluciones y consecuencias. Al dar nombre a un patrón estamos incrementado nuestro vocabulario de diseño, lo que nos permite diseñar a un mayor nivel de abstracción. De esta manera, resulta más fácil pensar en nuestros diseños y transmitirlos a otros, junto con sus ventajas e inconvenientes.

2. El problema describe cuándo aplicar el patrón. Explica el problema y su contexto. Puede describir problemas concretos de diseño (por ejemplo, cómo representar algoritmos como objetos), así como las estructuras de clases y objetos que son sintomáticas de un diseño inflexible. A veces, el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patrón.

3. La solución describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o una implementación en concreto, sino que un patrón es más bien como una plantilla que puede aplicarse en muchas situaciones diferentes. El patrón proporciona una descripción abstracta de un problema de diseño y cómo lo resuelve una estructura general de clases y objetos.

4. Las consecuencias son los resultados de aplicar el patrón, así como sus ventajas e inconvenientes. Aunque cuando se describen decisiones de diseño muchas veces no se reflejan las consecuencias de éstas, lo cierto es que son de vital importancia para evaluar las distintas alternativas. En el caso concreto de los patrones de diseño, las consecuencias incluyen su impacto sobre la flexibilidad, extensibilidad y portabilidad de un sistema.

4.2. CLASIFICACIÓN DE LOS PATRONES DE DISEÑO

Los patrones de diseño varían en su granularidad y nivel de abstracción. Dado que existen muchos patrones de diseño, es necesario un modo de organizarlos.

En su libro Erich Gamma [GHJ+95] realiza una clasificación general de los 23 patrones de diseño agrupados en tres categorías: patrones de creación, estructurales y de comportamiento. Esta clasificación sigue el criterio del propósito, es decir lo que realiza cada patrón.

4.2.1. Patrones de Creación

Este tipo de patrones son los que tienen un propósito de creación de objetos. Dependiendo del ámbito, si se aplican principalmente a Clases o a Objetos, podemos decir que los patrones de creación de clases delegan alguna parte del proceso de creación de objetos en las subclases, mientras que los patrones de creación de objetos lo hacen en otro objeto.

En las siguientes secciones se describe cada uno de los patrones de esta categoría.

4.2.1.1. Abstract Factory

El propósito del patrón Abstract Factory (Fábrica Abstracta) es crear objetos cuya clase concreta sólo se conoce en tiempo de ejecución. Para ello, se declara una interfaz para la creación de los distintos tipos de objetos (llamados productos en el contexto de este patrón) y se deja a las clases concretas que la implementan la tarea de crear realmente los objetos. La forma más frecuente de hacerlo es definiendo un método de fabricación (patrón Factory Method) para cada producto.

La estructura general del patrón se representa en la Figura 4.1.

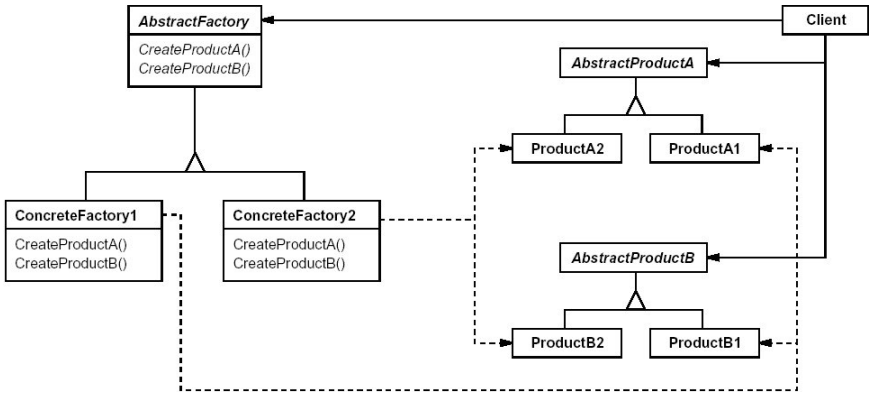


Figura 4.1. Estructura del patrón Abstract Factory.

4.2.1.2. Builder

Permite separar la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda dar lugar a diferentes representaciones.

La estructura general del patrón se representa en la Figura 4.2.

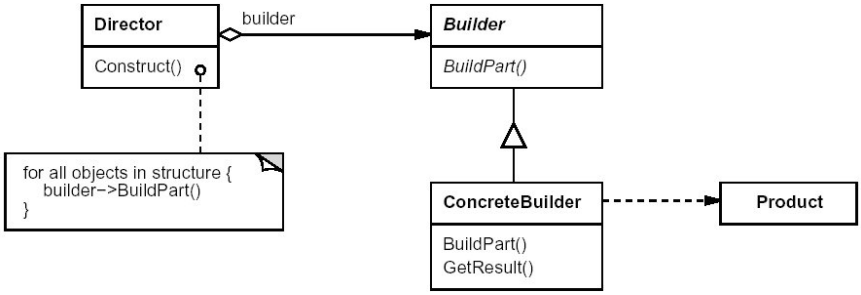


Figura 4.2. Estructura del patrón Builder.

4.2.1.3. Factory Method

Define una interfaz para crear un objeto, pero delega en las subclases la creación del mismo, dejando que sean éstas quienes decidan de qué clase concreta es el objeto a crear.

La estructura general del patrón se representa en la Figura 4.3.

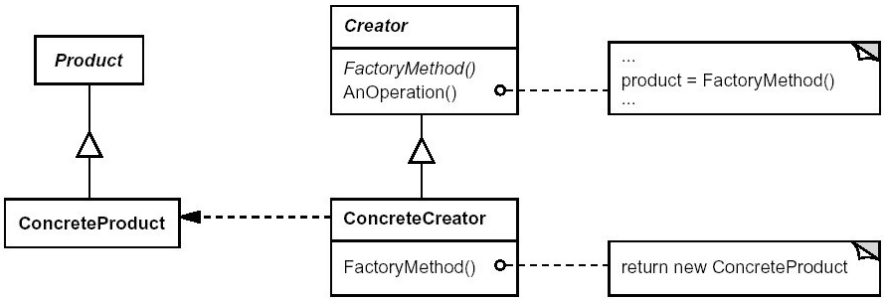


Figura 4.3.Estructura del patrón Factory Method.

4.2.1.4. Prototype

Especifica los tipos de objetos a crear por medio de una instancia prototípica, de modo que la creación de nuevos objetos consiste únicamente en clonar dicho prototipo.

La estructura general del patrón se representa en la Figura 4.4.

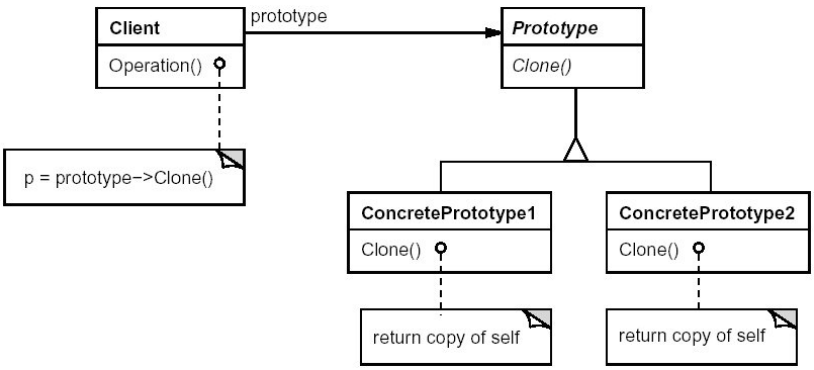


Figura 4.4. Estructura del patrón Prototype.

4.2.1.5. Singleton

El propósito de este patrón de diseño es garantizar que una clase sólo tiene una única instancia, proporcionando así mismo un punto de acceso a dicha instancia.

Realmente, el patrón de diseño Singleton se utiliza siempre que se quiere controlar el número de objetos creados de una clase dada, y que no tiene por qué ser sólo uno.

La estructura general del patrón se representa en la

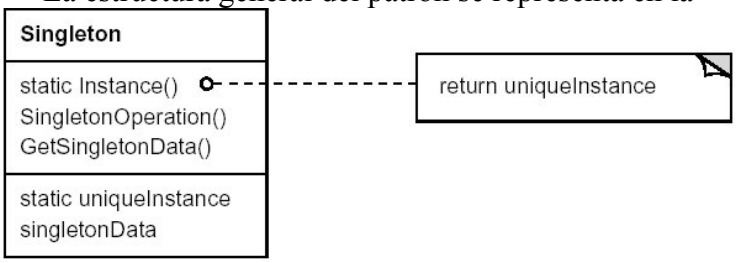


Figura 4.5.

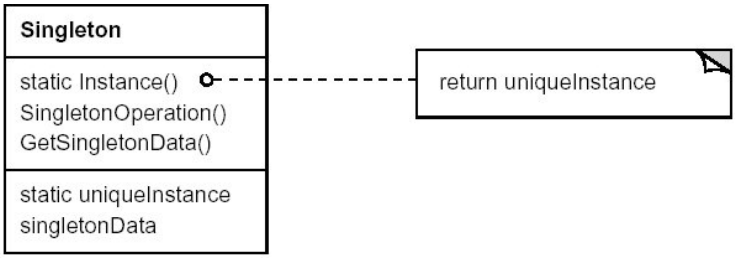


Figura 4.5.Estructura del patrón Singleton.

4.2.2. Patrones Estructurales

Este tipo de patrones tratan con la composición de clases u objetos. Dependiendo del ámbito, si se aplican principalmente a Clases o a Objetos, podemos decir que los patrones estructurales de clases usan la herencia para componer las clases, mientras que los patrones estructurales describen formas de ensamblar objetos.

En las siguientes secciones se describe cada uno de los patrones de esta categoría.

4.2.2.1. Adapter

Adapta la interfaz de una clase a la de otra que es la que espera el cliente, permitiendo así que trabajen juntas clases que de otro modo no podrían por tener interfaces incompatibles.

Este patrón de diseño, también llamado Wrapper (Envoltorio), es muy utilizado para poder reutilizar clases procedentes de una biblioteca en un dominio específico. Lo habitual será que las clases del dominio propio de la aplicación, aunque sean similares en funcionalidad, requieran interfaces distintas a las de las clases que se pretenden reutilizar.

La estructura general del patrón Adapter para clases se representa en la Figura 4.6.

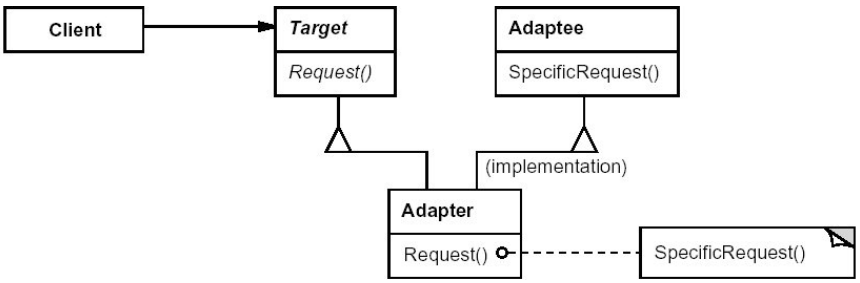


Figura 4.6. Estructura del patrón Adapter de clases.

4.2.2.2. Bridge

Separa una abstracción de su implementación, de modo que ambas puedan variar de forma independiente.

Cuando una abstracción puede tener varias implementaciones posibles, la forma más habitual de darles cabida es mediante la herencia. Una clase abstracta define la interfaz de la abstracción, y las subclases concretas la implementan de distintas formas. Pero este enfoque no siempre es lo bastante flexible. La herencia liga una implementación a la abstracción de forma permanente, lo que dificulta modificar, extender y reutilizar abstracciones e implementaciones de forma independiente.

La estructura general del patrón se representa en la Figura 4.7.

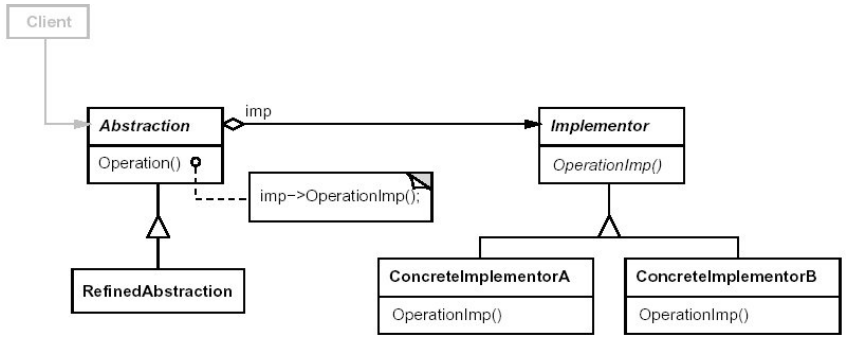


Figura 4.7. Estructura del patrón Bridge.

4.2.2.3. Composite

Compone objetos formando estructuras arbóreas para representar jerarquías de parte-todo, permitiendo que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

La estructura general del patrón se representa en la Figura 4.8.

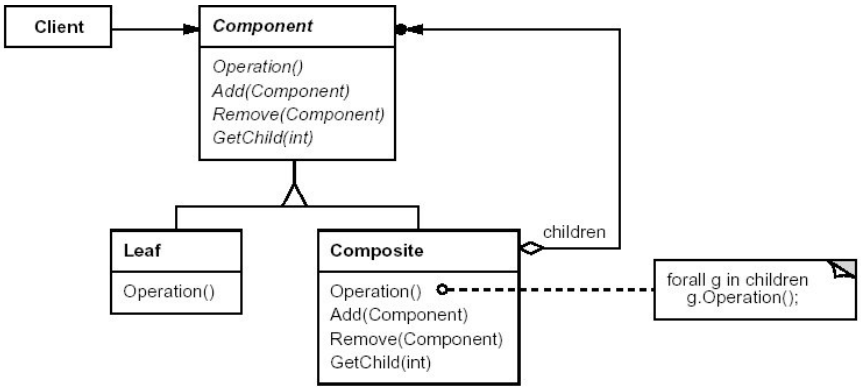


Figura 4.8.Estructura del patrón Composite.

4.2.2.4. Decorator

Permite añadir dinámicamente nuevas responsabilidades a un objeto, proporcionando así una alternativa flexible a la herencia para extender la funcionalidad.

A veces queremos añadir responsabilidades a objetos individuales en vez de a toda una clase. Por ejemplo, una biblioteca de clases para interfaces de usuario debería permitir añadir propiedades, como un borde, o comportamientos, como capacidad de desplazamiento, a cualquier componente de la interfaz de usuario.

Un modo de añadir responsabilidades es a través de la herencia. Heredar de una clase para obtener un borde, por ejemplo, dibujaría un borde alrededor de todas las instancias de la subclase. Sin embargo, esto es inflexible, ya que la elección del borde se hace estáticamente. Un cliente no puede controlar cómo y cuándo decorar el componente con un borde.

Un enfoque más flexible sería encerrar el componente en otro objeto que añada el borde. Al objeto confinante se le denomina decorador. El decorador se ajusta a la interfaz del componente que decora de manera que la presencia de aquél es transparente para los clientes del componente. El decorador redirige las peticiones al componente y puede realizar acciones adicionales (tales como dibujar un borde) antes o después. Dicha transparencia permite anidar decoradores de forma recursiva, permitiendo así añadir un número ilimitado de responsabilidades dinámicamente.

La principal característica de este patrón de diseño es que el objeto decorado puede aparecer en cualquier lugar en que lo haría el objeto al que decora, de manera que los clientes normalmente no tienen que distinguir entre uno y otro, siendo la decoración totalmente transparente para ellos.

La estructura general del patrón se representa en la Figura 4.9.

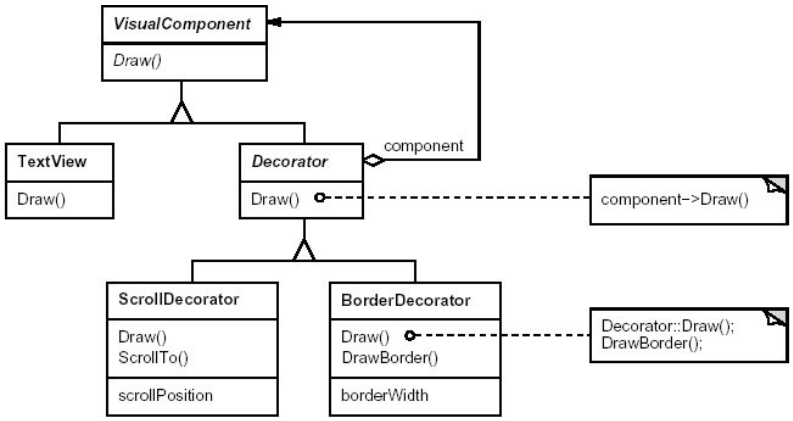


Figura 4.9. Estructura del patrón Decorator.

4.2.2.5. Facade

El propósito del patrón Facade (Fachada) es proporcionar una interfaz unificada para un conjunto de interfaces de un subsistema. Es decir, define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.

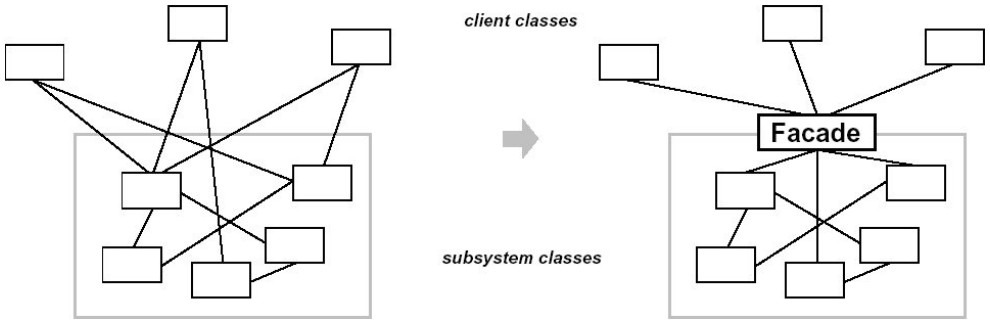


Figura 4.10.Aplicación del patrón Facade.

Estructurar un sistema en subsistemas ayuda a reducir la complejidad. Así, un objetivo de diseño habitual es minimizar la comunicación y dependencias entre subsistemas. Un modo de lograr esto es introduciendo un objeto fachada que proporcione una interfaz única y simplificada para los servicios más generales del subsistema. La Figura 4.10 muestra esquemáticamente el resultado de aplicar este patrón sobre un subsistema dado.

Las clases clientes del subsistema sólo se comunican con el objeto Facade, facilitando así la labor a los programadores en la mayoría de las clases. No obstante, este patrón no oculta el resto de las clases del subsistema, de modo que todavía es posible acceder a ellas en los pocos casos en los que se necesite mayor control que el que proporciona la interfaz de la fachada.

La estructura general del patrón se representa en la Figura 4.11.

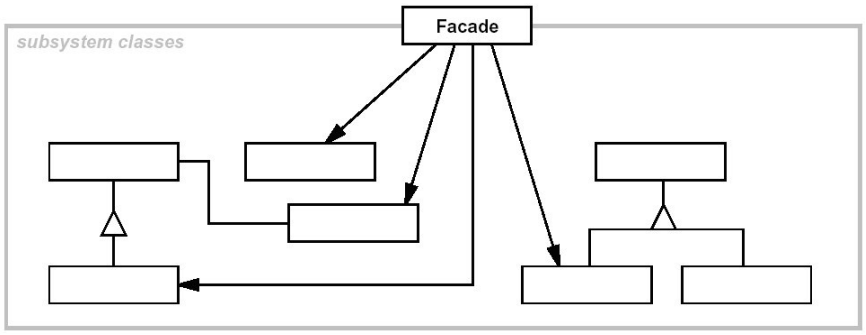


Figura 4.11.Estructura del patrón Facade.

4.2.2.6. Flyweight

Este patrón de diseño, que en español se traduciría literalmente como "peso mosca", especifica cómo se pueden soportar un gran número de objetos pequeños de manera eficiente, compartiéndolos.

Esto permite resolver el dilema que muchas veces se da en la orientación a objetos y que consiste en que, si bien la mayor flexibilidad la obtenemos cuando los objetos se usan de forma intensiva en todo el diseño, llevar este enfoque al extremo -que todo sea un objeto- haría que determinadas implementaciones fuesen prohibitivamente ineficientes.

4.2.2.7. Proxy

El patrón Proxy (Apoderado o Representante) proporciona un representante o sustituto de un objeto para controlar el acceso a aquél.

Las razones para usar este patrón son varias. En general, resulta de utilidad cada vez que se necesita una referencia a un objeto que vaya más allá de un simple puntero. Éstas son varias situaciones comunes en las que es aplicable este patrón:

- Un proxy remoto proporciona un representante local de un objeto situado en otro espacio de direcciones.

- Un proxy virtual permite crear objetos costosos por encargo. Pensemos en un editor de texto que permita insertar objetos gráficos en un documento. Algunas de estas imágenes, si son muy grandes, puede llevar algún tiempo crearlas, ralentizando con ello la apertura del documento. Sin embargo, ésta debería ser una operación que se efectuase rápidamente, por lo que habría que evitar crear todas estas imágenes a la vez en cuanto se abra el documento. Por otro lado, tampoco hace falta crear todos esos objetos, ya que no todas las imágenes van a estar visibles en el documento al mismo tiempo. La solución sería utilizar otro objeto, un proxy de la imagen, que actúe como un sustituto de la imagen real. El proxy se comporta igual que la imagen y se encarga de crearla cuando sea necesario.
- Un proxy de protección controla el acceso al objeto original. Este tipo de proxy es útil cuando los objetos pueden tener diferentes permisos de acceso. Por ejemplo, los KernelProxy del sistema operativo Choices proporcionan un acceso protegido a los objetos del sistema operativo.
- Una referencia inteligente es un sustituto de un simple puntero que lleva a cabo operaciones adicionales cuando se accede a un objeto. Algunos ejemplos de usos típicos son:
    - Contar el número de referencias al objeto real, de manera que éste pueda liberarse automáticamente cuando no haya ninguna referencia apuntándole (también se conocen con el nombre de punteros inteligentes).
    - Cargar un objeto persistente en memoria cuando es referenciado por primera vez.
    - Comprobar que se bloquea el objeto real antes de acceder a él para garantizar que no pueda ser modificado por ningún otro objeto

La estructura general del patrón se representa en la Figura 4.12.

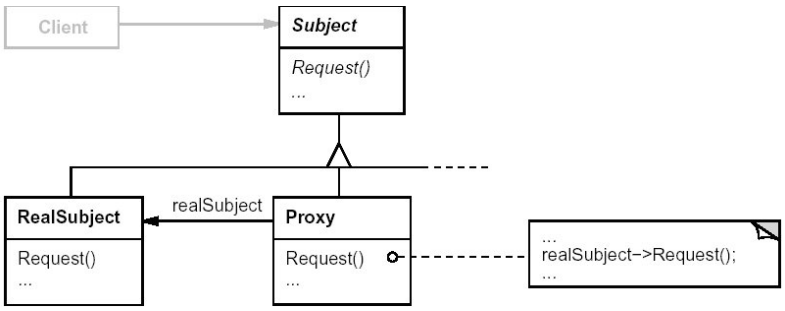


Figura 4.12. Estructura del patrón Proxy.

4.2.3. Patrones de Comportamiento

Este tipo de patrones caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad. Dependiendo del ámbito, si se aplican principalmente a Clases o a Objetos, podemos decir que los patrones de comportamiento de clases usan la herencia para describir algoritmos y flujos de datos, mientras que los patrones de comportamiento de objetos describen cómo cooperan un grupo de objetos para realizar una tarea que ningún objeto puede llevar a cabo por sí mismo.

En las siguientes secciones se describe cada uno de los patrones de esta categoría.

4.2.3.1. Chain of Responsibility

El propósito de este patrón (Cadena de Responsabilidad) es evitar el acoplamiento entre el emisor de una petición y su receptor, dando a más de un objeto la posibilidad de responder a dicha petición.

La idea es tener una cadena de objetos que son capaces de tratar estas peticiones, redirigiendo la petición a su sucesor en la cadena cuando ellos, por la razón que sea, no la procesan (bien sea por que no pueden, porque la petición no era para ellos, etcétera). Con este patrón de diseño es posible añadir o eliminar participantes en la cadena y, por tanto, cambiar dinámicamente las responsabilidades en el tratamiento de las peticiones.

El problema es que el objeto que en última instancia proporciona la ayuda no conoce explícitamente al objeto que inicia la petición de ayuda. Necesitamos un modo de desacoplar el objeto que da lugar a la petición de ayuda de los objetos que podrían proporcionar dicha información. El patrón Chain of Responsibility define cómo hacer esto.

El primer objeto de la cadena recibe la petición y o bien la procesa o bien la redirige al siguiente candidato en la cadena, el cual hace lo mismo. El objeto que hizo la petición no tiene un conocimiento explícito de quién la tratará -decimos que la petición tiene un receptor implícito-.

La estructura general del patrón se representa en la Figura 4.13.

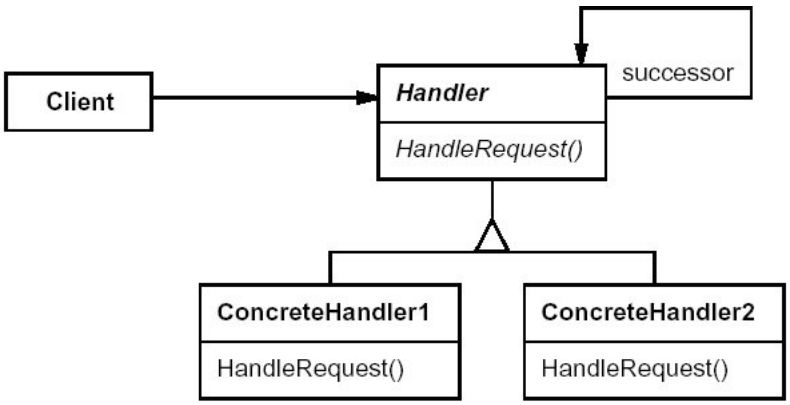


Figura 4.13. Estructura del patrón Chain of Responsibility.

4.2.3.2. Command

Este patrón de diseño (Orden) encapsula una petición en un objeto. Esto permitiría, por ejemplo, parametrizar clientes con distintas peticiones, guardar éstas en una cola o revertir los efectos de las operaciones ya realizadas (mediante las típicas opciones deshacer/repetir).

A veces es necesario enviar peticiones a objetos sin saber nada acerca de la operación solicitada o de quién es el receptor de la petición. Lo que hace el patrón Command es convertir la propia petición en un objeto, el cual se puede guardar y enviar exactamente igual que cualquier otro objeto.

La estructura general del patrón se representa en la Figura 4.14.

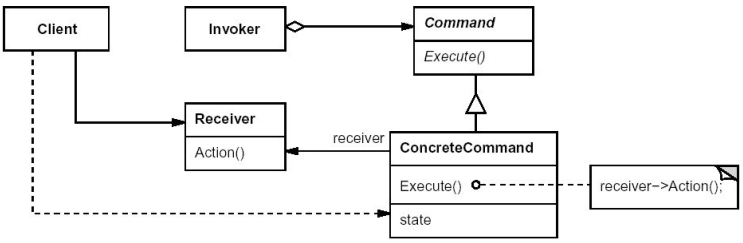


Figura 4.14.Estructura del patrón Command.

4.2.3.3. Interpreter

Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar instrucciones de ese lenguaje.

La motivación general que subyace en el patrón Interpreter (Intérprete) es resolver problemas simples y repetitivos formulándolos como sentencias de un lenguaje de programación. Este lenguaje debe estar definido de tal forma que un programa se pueda representar como un árbol sintáctico abstracto (AST), el cual se recorrerá posteriormente para ejecutar el programa.

La estructura general del patrón se representa en la Figura 4.15.

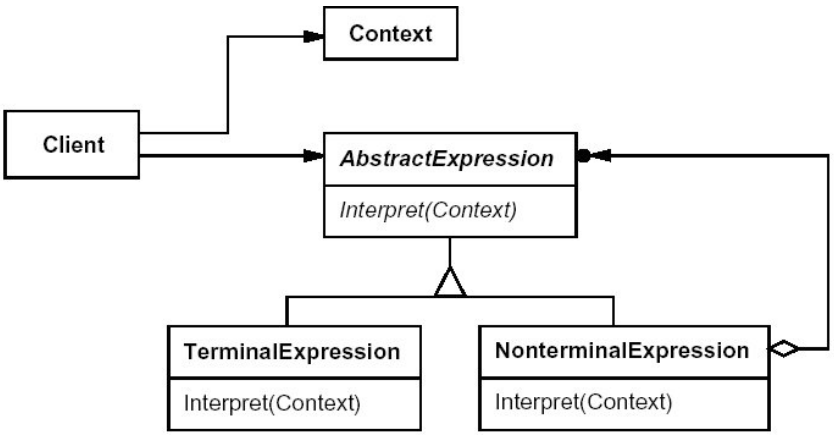


Figura 4.15.Estructura del patrón Interpreter.

4.2.3.4. Iterator

Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado, como una lista, sin exponer su representación interna. También permite realizar distintos tipos de recorrido sin tener que plagar la interfaz de la lista, en este caso, con las operaciones necesarias para llevar a cabo dichos recorridos. O incluso sería posible realizar varios recorridos a la vez sobre la misma lista.

La idea es separar la responsabilidad del recorrido de la colección y sacarla fuera a un objeto iterador. La clase Iterator define una interfaz para acceder a los elementos de la colección y cada objeto de esa clase es el responsable de saber cuál es el elemento actual, es decir, sabe qué elementos ya han sido recorridos.

La estructura general del patrón se representa en la Figura 4.16.

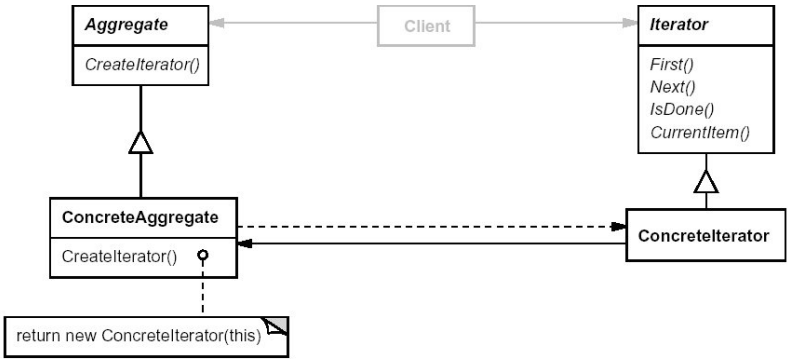


Figura 4.16.Estructura del patrón Iterator.

4.2.3.5. Mediator

Cuando hay un gran número de objetos fuertemente interrelacionados se hace enormemente dificultoso realizar cambios sobre ellos, puesto que cada uno de ellos depende en gran medida de los otros. El patrón de diseño Mediator (Mediador) desacopla dichos objetos al encapsular el modo en que interactúan entre sí, evitando que los objetos se refieran explícitamente unos a otros, lo que permite variar la interacción entre ellos de forma independiente.

Este patrón es importante porque, si bien uno de los principios de la orientación objetos promueve la distribución de comportamiento entre distintos objetos que cooperan entre sí, dicha distribución puede dar lugar a una estructura de objetos con muchas conexiones entre ellos. En el caso peor, cada objeto acaba por conocer a todos los demás.

Así, aunque dividir un sistema en muchos objetos suele mejorar la reutilización y aumenta la flexibilidad del diseño, la proliferación de interconexiones entre ellos tiende a reducir ésta de nuevo, ya que es menos probable que un objeto pueda funcionar sin la ayuda de otros (el sistema en su conjunto se comporta como si fuera un bloque monolítico). Más aún, puede ser difícil cambiar el comportamiento del sistema de manera significativa, ya que éste se encuentra distribuido en muchos objetos. Como resultado, podemos vernos forzados a definir muchas subclases para personalizar el comportamiento del sistema.

Básicamente, lo que promueve el patrón de diseño Mediator es que haya un objeto mediador que sirva como intermediario entre los distintos objetos relacionados. De esta forma, los objetos sólo conocen al mediador, reduciendo así el número de interconexiones entre ellos.

La estructura general del patrón se representa en la Figura 4.17.

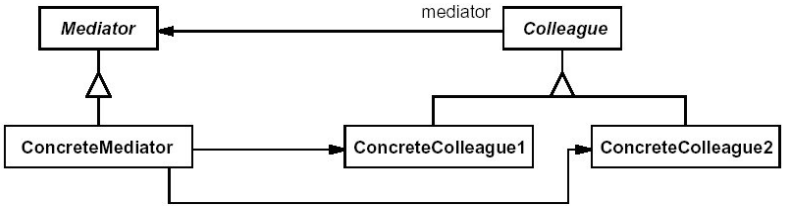


Figura 4.17. Estructura del patrón Mediator.

4.2.3.6. Memento

El propósito de este patrón es guardar el estado de un objeto de forma que éste pueda ser restaurado más tarde.

Un memento es un objeto que almacena una instantánea del estado interno de otro objeto (el creador del memento). Esto es útil por ejemplo a la hora de implementar la habitual operación de deshacer o mecanismos de recuperación de errores. Por ejemplo, en el primer caso la operación de deshacer solicitará un memento al objeto correspondiente cuando necesite guardar el estado de éste, el cual creará e inicializará el memento con información que representa su estado actual. En aras de preservar la ocultación de datos propia de la orientación a objetos, sólo el creador puede almacenar y recuperar información del memento, siendo éste "opaco" a otros objetos.

La estructura general del patrón se representa en la Figura 4.18.

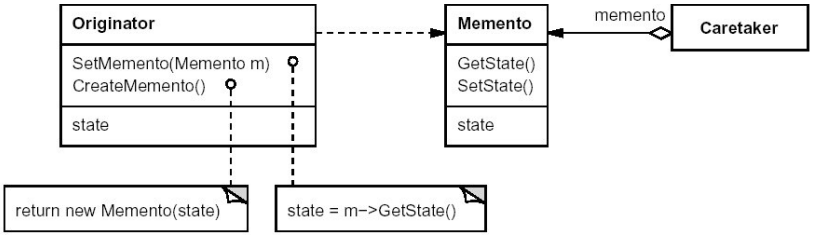


Figura 4.18.Estructura del patrón Memento.

4.2.3.7. Observer

El patrón Observer (Observador), que también se conoce con el significativo nombre de Publicar-Suscribir (Publish-Subscribe), permite definir una dependencia uno a muchos entre objetos, de manera que cuando el estado de un objeto cambia se avisa automáticamente a todos los que dependen de él para que se actualicen.

Este patrón de diseño es, por ejemplo, la base sobre la que se sustenta el patrón arquitectónico Modelo-Vista-Controlador (MVC) [BMR+96], cuya primera implementación conocida es en Smalltalk 80 [KP88]. La idea que subyace tras MVC es que las clases que definen los datos de una aplicación y sus representaciones puedan reutilizarse de forma independiente.

Los principales objetos de este patrón son el sujeto y el observador. Un sujeto puede tener cualquier número de observadores dependientes de él. Cada vez que cambia el estado del sujeto éste avisa a todos sus observadores. En respuesta, cada observador consultará al sujeto su nuevo estado para actualizarse de forma apropiada.

La estructura general del patrón se representa en la Figura 4.19.

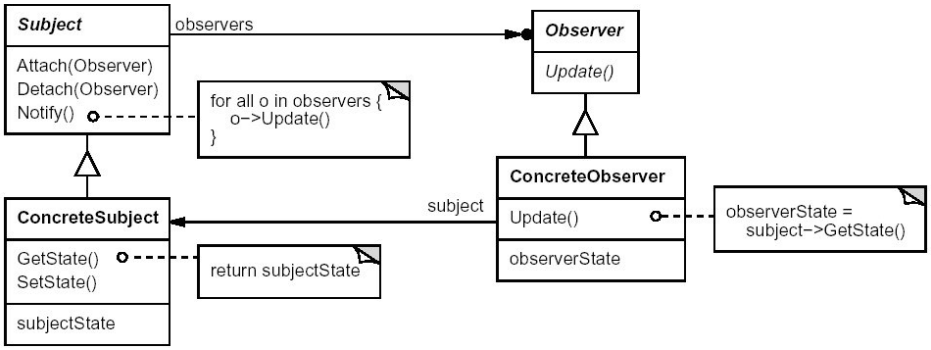


Figura 4.19.Estructura del patrón Observer.

4.2.3.8. State

Permite que un objeto modifique su comportamiento cuando cambia su estado interno, como si hubiera cambiado la clase del objeto.

Muchas veces, el comportamiento de un objeto puede ser dividido en diferentes "modos", dependiendo de su estado actual. El modo habitual de implementar esto era incluyendo alguna variable de estado y consultando el valor de la misma en cada método que depende del estado. Naturalmente, esto oscurece el código al mezclar la lógica propia de cada operación con un montón de sentencias case o de if anidados, más propios de la programación estructurada que del buen código orientado a objetos.

La propuesta que hace este patrón para evitar eso es crear una clase abstracta para el estado junto con una serie de clases concretas que heredan de ella para representar cada uno de los estados concretos.

La estructura general del patrón se representa en la Figura 4.20.

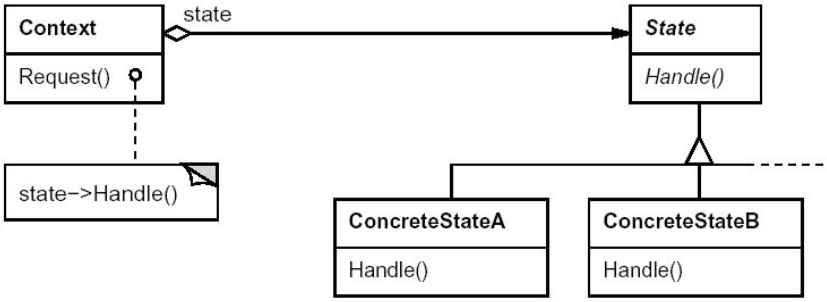


Figura 4.20. Estructura del patrón State.

4.2.3.9. Strategy

El patrón Strategy (Estrategia) define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables, permitiendo así que pueda cambiarse el algoritmo independientemente de los clientes que lo usan.

La estructura general del patrón se representa en la Figura 4.21.

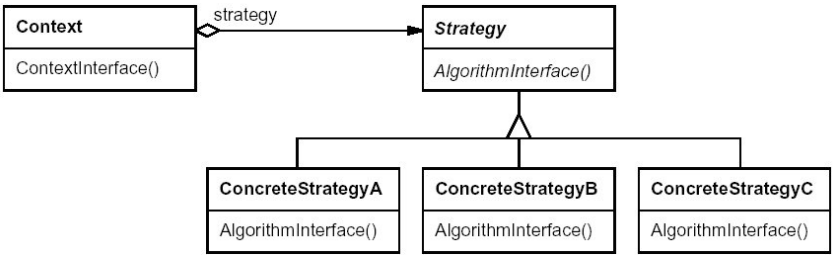


Figura 4.21. Estructura del patrón Strategy.

4.2.3.10. Template Method

El patrón Template Method (Método de Plantilla) permite que una operación defina únicamente el esqueleto de un algoritmo, postergando algunos pasos concretos a las subclases. De esta manera, posibilita que las subclases redefinan ciertos pasos de un algoritmo sin modificar la estructura del algoritmo en sí.

Este patrón resulta especialmente útil en los frameworks, en los que se definen métodos que posteriormente deberán que ser especializados por las aplicaciones concretas.

La estructura general del patrón se representa en la Figura 4.22.

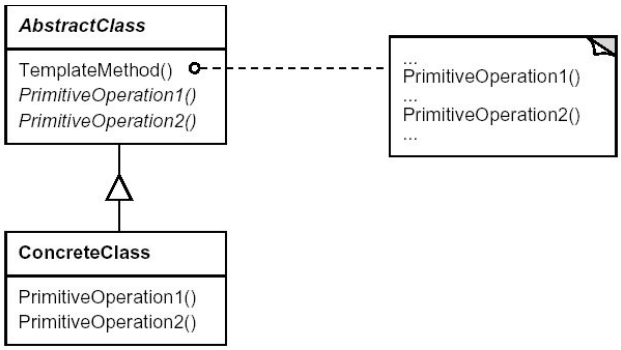


Figura 4.22. Estructura del patrón Template Method.

4.2.3.11. Visitor

El patrón Visitor (Visitante) define una operación sobre los elementos de una estructura de objetos sin cambiar las clases de los elementos sobre los que opera.

La estructura general del patrón se representa en la Figura 4.23.

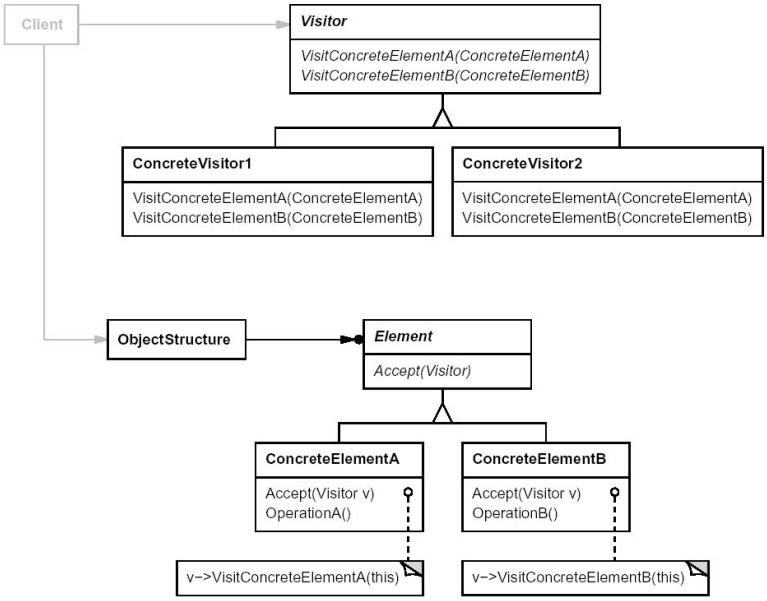


Figura 4.23.Estructura del patrón Visitor.

5. UNIFIED MODELING LANGUAGE: UML 2.1

El lenguaje unificado de modelado (UML) es una especificación del OMG (Object Management Group (OMG)). Además esta especificación ha sido aceptada por ISO (International Organization for Standardization).

UML es un lenguaje estándar para escribir planos de software. UML puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

UML es apropiado para modelar desde sistemas de información en empresas hasta aplicaciones distribuidas basadas en la Web, e incluso para sistemas empotrados de tiempo real muy exigentes. Es un lenguaje muy expresivo, que cubre todas las vistas necesarias para desarrollar y luego desplegar tales sistemas.

UML es sólo un lenguaje y por tanto es tan sólo una parte de un método de desarrollo de software.

5.1. CARACTERÍSTICAS

5.1.1. UML es un lenguaje

Un lenguaje proporciona un vocabulario y las reglas para combinar palabras de ese vocabulario con el objetivo de posibilitar la comunicación. Un lenguaje de modelado es un lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema. Un lenguaje de modelado como UML es por tanto un lenguaje estándar para los planos del software.

El vocabulario y las reglas de un lenguaje como UML indican cómo crear y leer modelos bien formados, pero no dicen qué modelos se deben crear ni cuándo se deberían crear. Esta es la tarea del proceso de desarrollo de software.

5.1.2. UML es un lenguaje para visualizar

UML es algo más que un simple montón de símbolos gráficos. Detrás de cada símbolo en la notación UML hay una semántica bien definida. De esta manera, un desarrollador puede escribir un modelo en UML, y otro desarrollador, o incluso otra herramienta, puede interpretar ese modelo sin ambigüedad.

5.1.3. UML es un lenguaje para especificar

En este contexto, especificar significa construir modelos precisos, no ambiguos y completos. En particular, UML cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar un sistema con gran cantidad de software.

5.1.4. UML es un lenguaje para construir

UML no es un lenguaje de programación visual, pero sus modelos pueden conectarse de forma directa a una gran variedad de lenguajes de programación.

Esta correspondencia permite ingeniería directa: la generación de código a partir de un modelo UML en un lenguaje de programación. Lo contrario también es posible: se puede reconstruir un modelo en UML a partir de una implementación.

UML es lo suficientemente expresivo y no ambiguo como para permitir la ejecución directa de modelos, la simulación de sistemas y la instrumentación de sistemas en ejecución.

5.1.5. UML es un lenguaje para documentar

Una organización de software que trabaje bien produce toda clase de artefactos además de código ejecutable. Estos artefactos incluyen:

- Requisitos.
- Arquitectura.
- Diseño.
- Código fuente.
- Planificación de proyectos.
- Pruebas.
- Prototipos.
- Versiones.

Dependiendo de la cultura de desarrollo, algunos de estos artefactos se tratan más o menos formalmente que otros. Tales artefactos no son sólo los entregables de un proyecto, también son críticos en el control, la medición y comunicación que requiere un sistema durante su desarrollo y después de su despliegue.

UML cubre la documentación de la arquitectura de un sistema y todos sus detalles. UML también proporciona un lenguaje para expresar requisitos y pruebas. Finalmente; UML proporciona un lenguaje para modelar las actividades de planificación de proyectos y gestión de versiones.

5.2. LA ESPECIFICACIÓN DE UML

El lenguaje UML viene definido por un conjunto de documentos publicados por el OMG. La versión actual de UML está compuesta por cuatro partes.

- UML Superstructure. Especifica el lenguaje desde el punto de vista de sus usuarios finales. Define 6 diagramas estructurales, 3 diagramas de comportamiento, 4 diagramas de interacción así como los elementos que todos ellos comprenden.
- UML Infrastructure. Especifica las construcciones que conforman los cimientos de UML. Para ello, define el núcleo de un metalenguaje que podrá ser reutilizado para definir los metamodelos de otros lenguajes: alinea el metamodelo de UML con MOF.
- UML Object Constraint Language. OCL es un lenguaje que permite ampliar la semántica de los modelos UML mediante la definición de precondiciones, postcondiciones, invariantes y otras condiciones.
- UML Diagram Interchange. Extiende el metamodelo de UML con un paquete adicional que modela información de carácter gráfico asociada a los diagramas, permitiendo el intercambio de modelos conservando su representación original.

5.3. MODELO CONCEPTUAL DE UML

Para comprender UML, es necesario adquirir un modelo conceptual del lenguaje, y esto requiere comprender tres elementos principales: los bloques básicos de construcción de UML, las reglas que dictan cómo se pueden combinar estos bloques físicos y algunos mecanismos comunes que se aplican a través de UML.

5.3.1. Bloques de construcción de UML

El vocabulario de UML incluye tres clases de bloques de construcción:

- Elementos.
- Relaciones.
- Diagramas.

Los elementos son abstracciones que son ciudadanos de primera clase en un modelo, las relaciones ligan estos elementos entre sí y los diagramas agrupan colecciones interesantes de elementos.

5.3.1.1. Elementos en UML

Hay cuatro tipos de elementos:

5.3.1.1.1. Elementos estructurales

Son los nombres de los modelos de UML. En su mayoría son las partes estáticas de un modelo, y representan cosas que son conceptuales o materiales. Son siete tipos:

- Clase: es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase implementa una o más interfaces. Gráficamente, una clase se representa como un rectángulo, generalmente incluyendo su nombre, atributos y operaciones.(Figura 5.1)

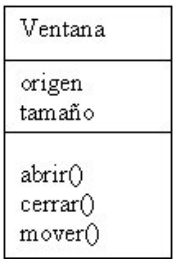


Figura 5.1.Representación de una clase.

- Interfaz: es una colección de operaciones que especifican un servicio de una clase o componente. Una interfaz, por lo tanto, describe el comportamiento externo visible del elemento. Una interfaz puede representar el comportamiento completo de la clase o componente o únicamente una parte de ese comportamiento. Una interfaz define una conjunto de especificaciones de operaciones pero nunca un conjunto de operaciones de implementación. Gráficamente, una interfaz se representa como un círculo junto con su nombre. Una interfaz raramente se encuentra sola. Mejor dicho, esta unida típicamente a la clase o componente que realiza la interfaz. (Figura 5.2)

Figura 5.2.Ejemplo de una interfaz.

- Colaboración: define una interacción y es una asociación de papeles y otros elementos que trabajan juntos para suministrar algún comportamiento cooperativo mayor que la suma del comportamiento de sus elementos. Por tanto, las colaboraciones tienen dimensión tanto estructural como de comportamiento. Una clase dada puede participar en varias colaboraciones. Estas colaboraciones representan, pues, la implementación de patrones que forman un sistema. Gráficamente, una colaboración se representa como una elipse de borde discontinuo, incluyendo normalmente sólo su nombre. (Figura 5.3)

Figura 5.3.Ejemplo de colaboracion.

- Caso de uso: es una descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular. Un caso de uso se utiliza para estructurar los aspectos de comportamiento en un modelo. Un caso de uso es realizado por una colaboración. Gráficamente, un caso de uso se representa como una elipse de borde continuo, incluyendo normalmente sólo su nombre. (Figura 5.3)

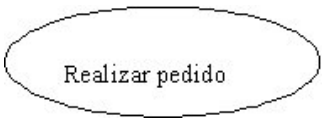


Figura 5.4.Ejemplo de caso de uso.

- Clase activa: es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución y por lo tanto pueden dar origen a actividades de control Unas clase activa es igual que una clase, excepto en que sus objetos representan elementos cuyo comportamiento es concurrente con otros elementos. Gráficamente, una clase activa se representa como una clase, pero con líneas más gruesas, incluyendo normalmente su nombre, atributos y operaciones. (Figura 5.5)

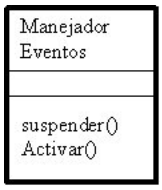


Figura 5.5.Ejemplo de clase Activa.

- Componente: es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la implementación de dicho conjunto. En un sistema, se podrán encontrar diferentes tipos de componentes de despliegue, así como componentes que sean artefactos del proceso de desarrollo, tales como archivos de código fuente. Un componente representa típicamente el empaquetamiento físico de diferentes elementos lógicos, como clases, interfaces y colaboraciones. Gráficamente, un componente se representa como un rectángulo con pestaña, incluyendo normalmente sólo su nombre. (Figura 5.6)

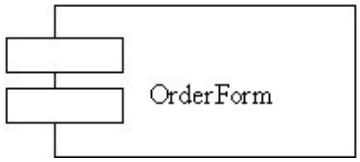


Figura 5.6.Ejemplo de componente.

- Nodo: es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que por lo general dispone de algo de memoria y, con frecuencia, capacidad de procesamiento. Un conjunto de componentes puede residir en un nodo y puede también migrar de un nodo a otro. Gráficamente, un nodo se representa como un cubo, incluyendo normalmente sólo su nombre. (Figura 5.7)

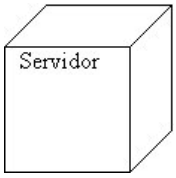


Figura 5.7.Ejemplo de nodo.

5.3.1.1.2. Elementos de comportamiento

Son las partes dinámicas de los modelos UML. Estos son los verbos de un modelo, y representan comportamiento en el tiempo y el espacio. Hay dos tipos principales:

- Interacción: es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para alcanzar un propósito específico. El comportamiento de una sociedad de objetos o una operación individual puede especificarse con una interacción. Una interacción involucra muchos otros elementos, incluyendo mensajes, secuencias de acción (el comportamiento invocado por un mensaje) y enlaces (conexiones entre objetos). Gráficamente, un mensaje se muestra como una línea dirigida, incluyendo casi siempre el nombre de la operación. (Figura 5.8)

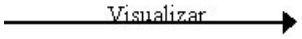


Figura 5.8.Ejemplo de interaccion.

- Máquina de estados: es un comportamiento que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, juntos con sus reacciones a estos eventos. El comportamiento de una clase individual o una colaboración de clases puede especificarse con una máquina de estados. Una máquina de estados involucra a tres elementos, incluyendo estados, transiciones (el flujo de un estado a otro), eventos (que disparan una transición) y actividades (la respuesta a una transición). Gráficamente, un estado se representa como un rectángulo de esquinas redondeadas, incluyendo normalmente su nombre y sus subestados, si los tiene. (Figura 5.9)

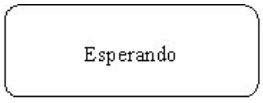


Figura 5.9.Ejemplo de estado.

5.3.1.1.3. Elementos de agrupación

Son las partes organizativas de los modelos UML. Estos son las cajas en las que puede descomponerse un modelo. Hay un único elemento principal de este tipo que son los paquetes.

- Paquete: es un mecanismo de propósito general para organizar elementos en grupos. Los elementos estructurales, los elementos de comportamiento, e incluso otros elementos de agrupación pueden incluirse en un paquete. Al contrario que los componentes (que existen en tiempo de ejecución), un paquete es puramente conceptual (sólo existe en tiempo de desarrollo). Gráficamente, un paquete se visualiza como una carpeta, incluyendo normalmente sólo su nombre y, a veces, su contenido. (Figura 5.10)

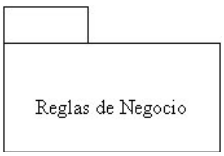


Figura 5.10.Ejemplo de paquete.

5.3.1.1.4. Elementos de anotación

Son las partes explicativas de los modelos UML. Son comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento de un modelo. El tipo principal de este elemento es la nota.

- Nota: es simplemente un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos. Gráficamente, una nota se representa como un rectángulo con una esquina doblada, junto con un comentario textual o gráfico. (Figura 5.11)

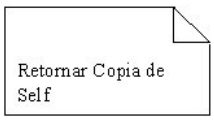


Figura 5.11.Ejemplo de nota.

5.3.1.2. Relaciones en UML

Hay cuatro tipos de relaciones:

5.3.1.2.1. Dependencia

Es una relación semántica entre dos elementos, en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (elemento dependiente). Gráficamente, una dependencia se representa como una línea discontinua, posiblemente dirigida, ocasionalmente incluye una etiqueta. (Figura 5.12)

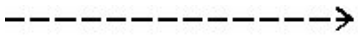


Figura 5.12.Representación de la relación de dependencia.

5.3.1.2.2. Asociación

Es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. La agregación es un tipo especial de asociación, que representa una relación estructural entre un todo y sus partes. Gráficamente, una asociación se representa como una línea continua, posiblemente dirigida, que a veces incluye una etiqueta y a menudo incluye otros adornos, como multiplicidad y los nombre de rol. (Figura 5.13)

Figura 5.13.Ejemplo de la relación de asociación.

5.3.1.2.3. Generalización

Es una relación de especialización/generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los objetos del elemento general (el padre). De esta forma, el hijo comparte la estructura y el comportamiento del padre. Gráficamente, una relación de generalización se representa como una línea continua con una punta flecha vacía que apunta al padre. (Figura 5.14)

Figura 5.14.Representación de la relación de generalización.

5.3.1.2.4. Realización

Es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Gráficamente, una relación de realización se representa como una mezcla entre una generalización y una relación de dependencia. (Figura 5.15)

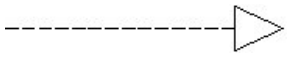


Figura 5.15.Representación de la relación de realización.

5.3.1.3. Diagramas en UML

Un diagrama es la representación gráfica de un conjunto de elementos. Los diagramas se dibujan para visualizar un sistema desde diferentes perspectivas, de forma que un diagrama es una proyección de un sistema.

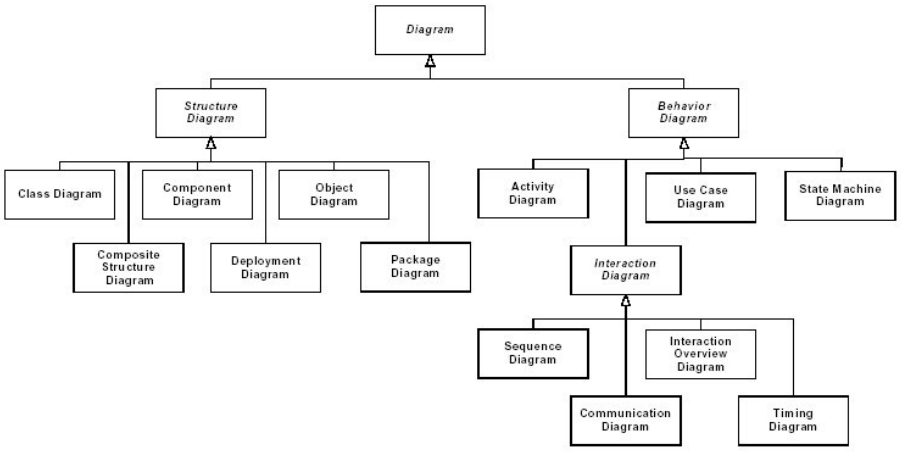


Figura 5.16.Jerarquía de los diagramas UML 2.1.

En teoría, un diagrama puede contener cualquier combinación de elementos y relaciones. En la práctica sólo surge un pequeño número de combinaciones, las cuales son consistentes con las cinco vistas más útiles que comprenden la arquitectura de un sistema con gran cantidad de software. Por ello, UML incluye los siguientes diagramas: (Figura 5.16)

5.3.1.3.1. Diagrama de clases

Muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Estos diagramas son los diagramas más comunes en el modelado de sistemas orientados a objetos. Los diagramas de clases cubren la vista de diseño estática de un sistema. Los diagramas de clases que incluyen clases activas cubren la vista de procesos estática de un sistema. (Figura 5.17)

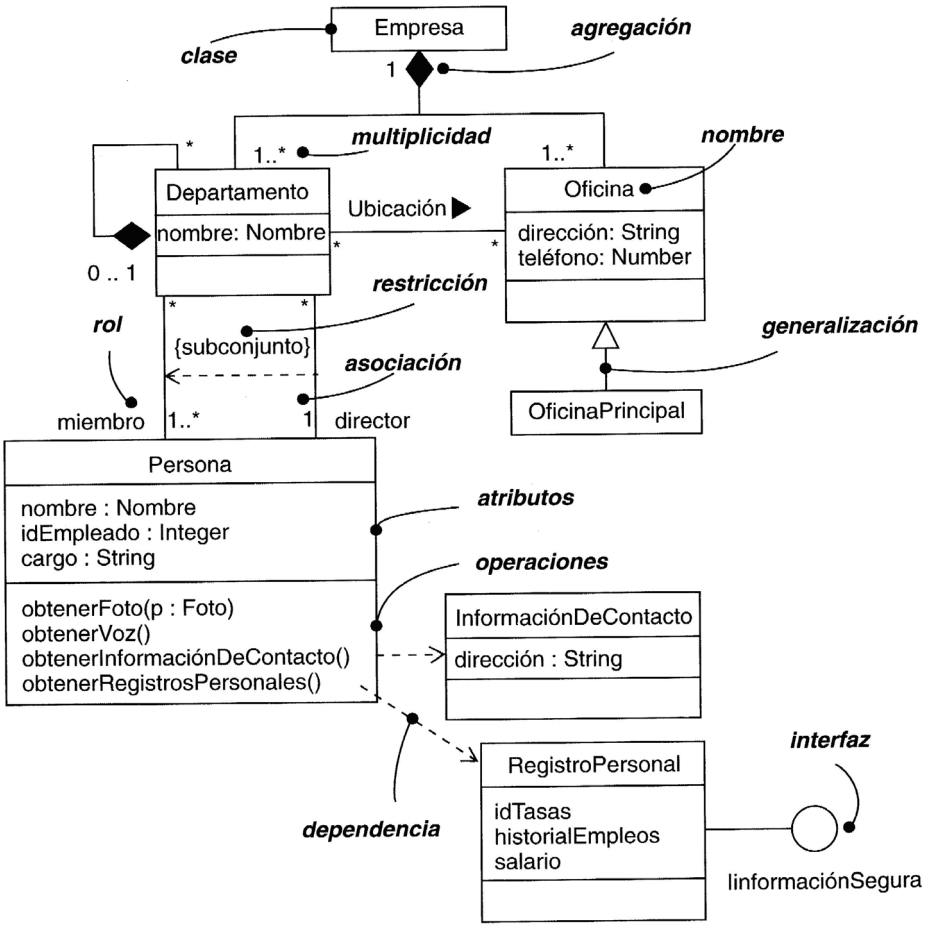


Figura 5.17.Diagrama de Clases.

5.3.1.3.2. Diagrama de objetos

Muestra un conjunto de objetos y sus relaciones. Los diagramas de objetos representan instantáneas de instancias de los elementos encontrados en los diagramas de clases. Estos diagramas cubren la vista de diseño estática o la vista de procesos estática de un sistema como lo hacen los diagramas de clases, pero desde la perspectiva de casos reales o prototípicos. (Figura 5.18)

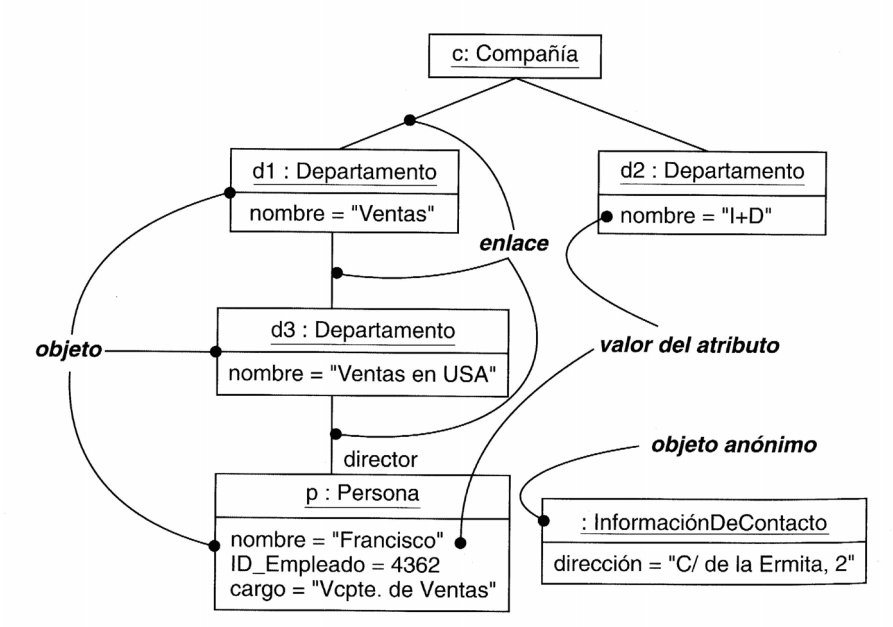


Figura 5.18.Diagrama de Objetos.

5.3.1.3.3. Diagrama de casos de uso

Muestra el conjunto de casos de uso y actores (un tipo especial de clases) y sus relaciones. Los diagramas de casos de uso cubren la vista de casos de uso estática de un sistema. Estos diagramas son especialmente importantes en el modelado y organización del comportamiento de un sistema. (Figura 5.19)

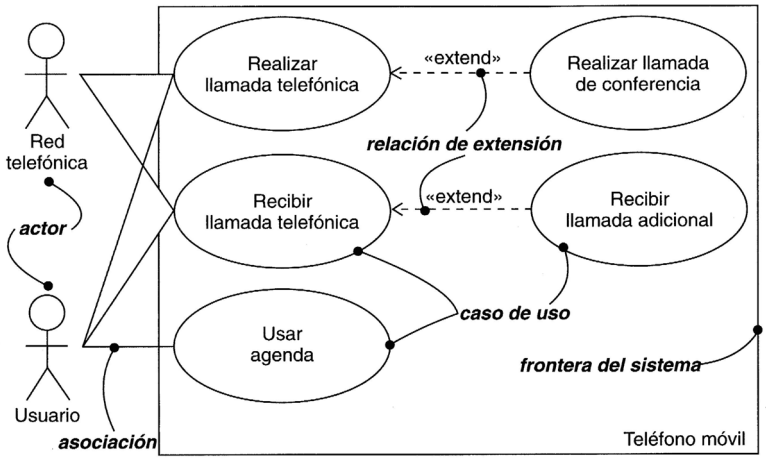


Figura 5.19.Diagrama de Casos de Uso.

5.3.1.3.4. Diagrama de interacción

Muestra una interacción, que consta de un conjunto de objetos y sus relaciones, incluyendo los mensajes que pueden ser enviados entre ellos. Estos diagramas cubren la vista dinámica de un sistema. Los diagramas de secuencia y de colaboración son de este tipo. Además estos diagramas son isomorfos, es decir, que se puede tomar uno y transformarlo en el otro.

- Diagrama de secuencia: es un diagrama de interacción que resalta la ordenación temporal de los mensajes. (Figura 5.20)

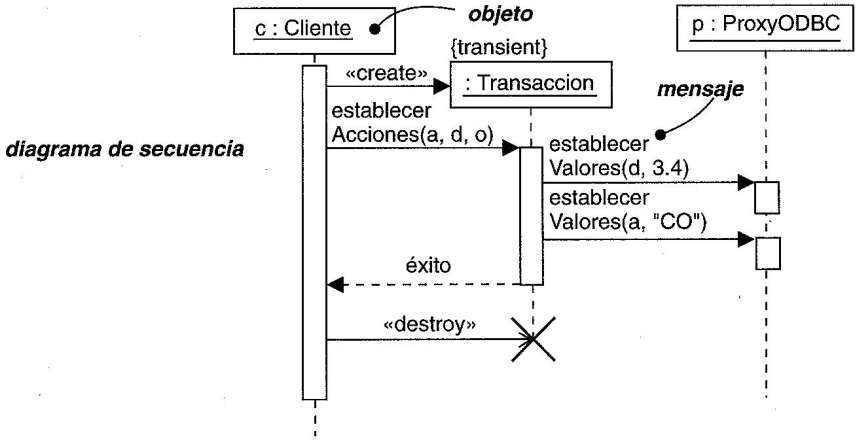


Figura 5.20.Diagrama de Secuencia.

- Diagrama de colaboración: es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes. (Figura 5.21)

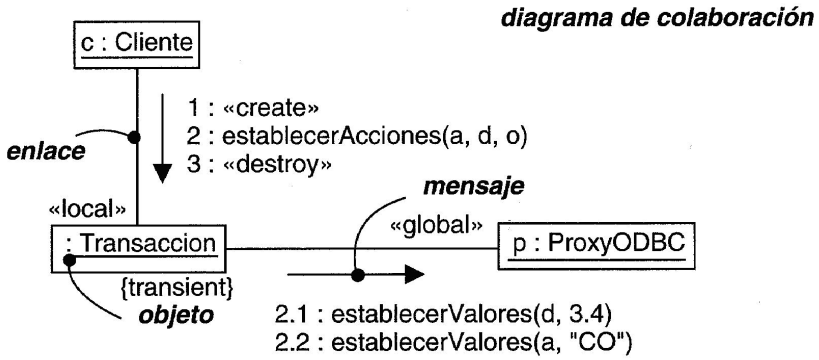


Figura 5.21.Diagrama de Colaboración.

5.3.1.3.5. Diagrama de estados

Muestra una máquina de estados, que consta de estados, transiciones, eventos y actividades. Los diagramas de estados cubren la vista dinámica de un sistema. Son especialmente importantes en el modelado del comportamiento de una interfaz, una clase o una colaboración y resaltan el comportamiento dirigido por eventos de un objeto. (Figura 5.22)

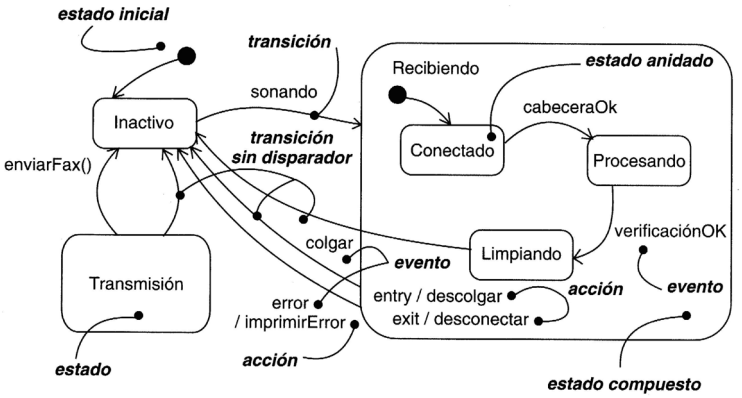


Figura 5.22.Diagrama de Estados.

5.3.1.3.6. Diagrama de actividades

Es un tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema. Los diagramas de actividades cubren la vista dinámica de un sistema. Son especialmente importantes al modelar el funcionamiento de un sistema y resaltan el flujo de control entre objetos. (Figura 5.23)

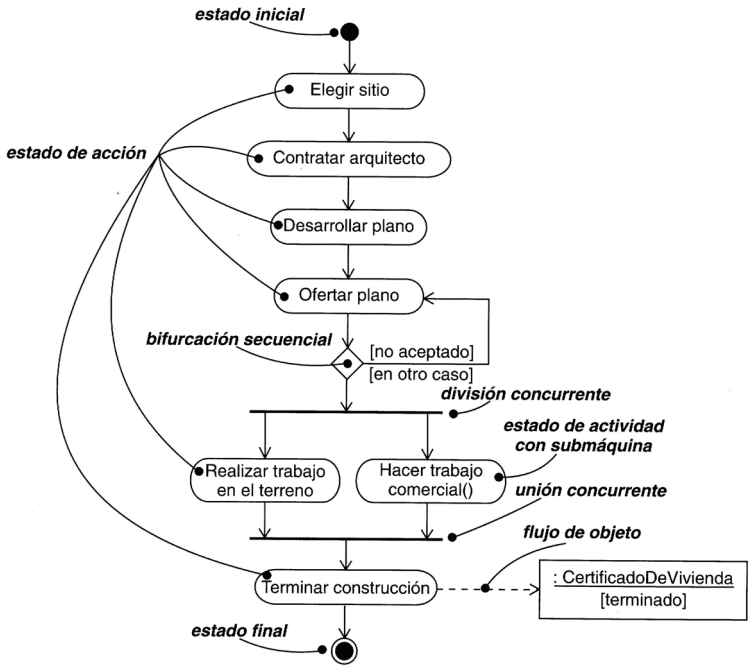


Figura 5.23.Diagrama de Actividades.

5.3.1.3.7. Diagrama de componentes

Muestra la organización y las dependencias entre un conjunto de componentes. Los diagramas de componentes cubren la vista de implementación estática de un sistema. Se relacionan con los diagramas de clases en que un componente se corresponde, por lo común, con una o más clases, interfaces o colaboraciones. (Figura 5.24)

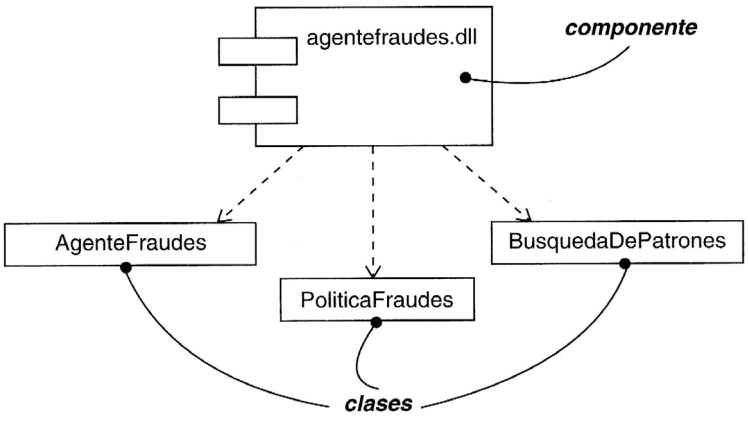


Figura 5.24.Diagrama de Componentes.

5.3.1.3.8. Diagrama de despliegue

Muestra la configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. Los diagramas de despliegue cubren la vista de despliegue estática de una arquitectura. Se relacionan con los diagramas de componentes en que un nodo incluye, por lo común, uno o más componentes. (Figura 5.25)

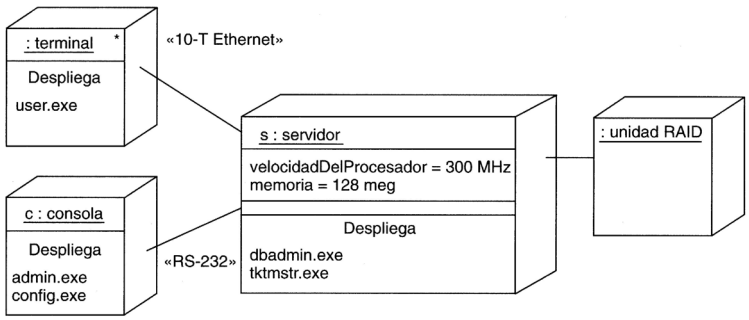


Figura 5.25.Diagrama de Despliegue.

5.3.2. Reglas de UML

Los bloques de construcción de UML no pueden simplemente combinarse de cualquier manera. Como cualquier lenguaje, UML tiene un número de reglas que especifican a qué debe parecerse un modelo bien formado. Un modelo bien formado es aquél que es semánticamente autoconsistente y está en armonía con todos sus modelos relacionados.

Los modelos que no llegan a ser bien formados son inevitables conforme los detalles de un sistema van apareciendo y mezclándose durante el proceso de desarrollo de software. Las reglas de UML estimulan (pero no obligan) a considerar las cuestiones más importantes de análisis, diseño e implementación que llevan a tales sistemas a convertirse en bien formados con el paso del tiempo.

5.3.3. Mecanismos comunes en UML

UML se simplifica mediante la presencia de cuatro mecanismos comunes que se aplican de forma consistente a través de todo el lenguaje:

5.3.3.1. Especificaciones

Detrás de cada elemento de la notación gráfica de UML hay una especificación que proporciona una explicación textual de la sintaxis y semántica de ese bloque de construcción. La notación gráfica de UML se utiliza para visualizar un sistema; la especificación de UML se utiliza para enunciar detalles del sistema.

5.3.3.2. Adornos

Todos los elementos en la notación UML comienzan con un símbolo básico, al cual puede añadirse una variedad de adornos específicos de ese símbolo. (Figura 5.26)

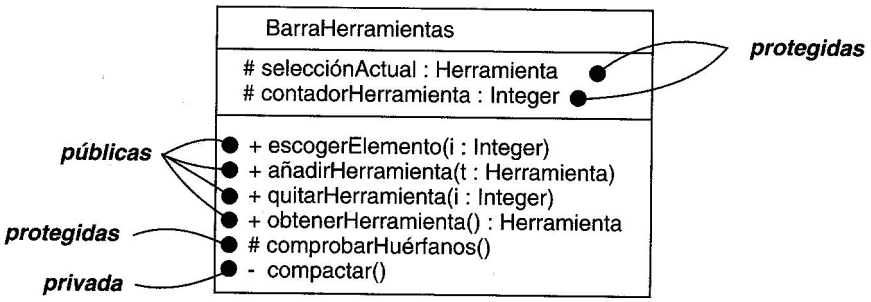


Figura 5.26.Adornos en la definición de una clase.

5.3.3.3. Divisiones comunes

Al modelar sistemas orientados a objetos, el mundo puede dividirse, al menos en un par de formas.

- División entre clases y objetos. (Figura 5.27)

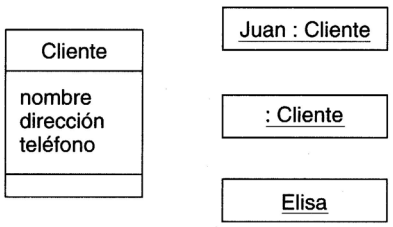


Figura 5.27.División entre clases y objetos.

- División entre interfaz e implementación. (Figura 5.28)

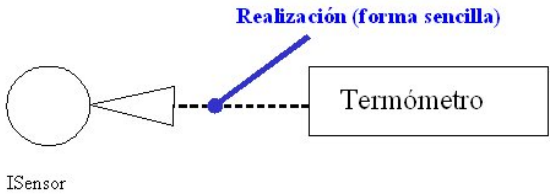


Figura 5.28.División entre interfaz e implementación.

5.3.3.4. Mecanismos de extensibilidad

UML proporciona un lenguaje estándar para escribir planos software, pero no es posible que un lenguaje cerrado sea siempre suficiente para expresar todos los matices posibles de todos los modelos en todos los dominios y en todos los momentos. Por esto, UML es abierto-cerrado, siendo posible extender el lenguaje de manera controlada. Los mecanismos de extensión son:

- Estereotipos
- Valores etiquetados
- Restricciones

5.4. ARQUITECTURA

La visualización, especificación, construcción y documentación de un sistema con gran cantidad de software requiere que el sistema sea visto desde varias perspectivas. Diferentes usuarios siguen diferentes agendas en relación con el proyecto, y cada uno mira a ese sistema de formas diferentes en diversos momentos a lo largo de la vida del proyecto. La arquitectura de un sistema es quizás el artefacto más importante que puede emplearse para manejar estos diferentes puntos de vista y controlar el desarrollo iterativo e incremental de un sistema a lo largo de su ciclo de vida.

La arquitectura es el conjunto de decisiones significativas sobre:

- La organización de un sistema software.
- La selección de elementos estructurales y sus interfaces a través de los cuales se constituye el sistema.
- Su comportamiento, como se especifica en las colaboraciones entre esos elementos.
- La composición de esos elementos estructurales y de comportamiento en subsistemas progresivamente más grandes.

- El estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos y sus interfaces, sus colaboraciones y su composición.

La arquitectura no tiene que ver solamente con la estructura y el comportamiento, sino también con el uso, la funcionalidad, el rendimiento, la capacidad de adaptación, la reutilización, la capacidad de ser comprendido, las restricciones económicas de tecnología y los compromisos entre alternativas, así como los aspectos estéticos.

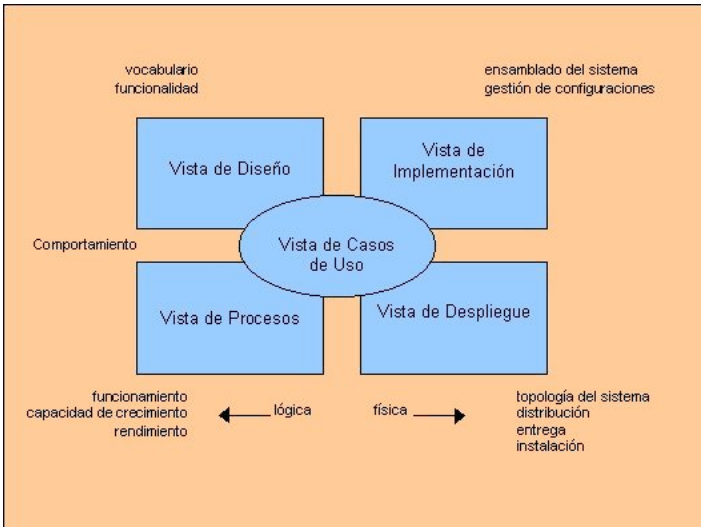


Figura 5.29.Modelado de la arquitectura de un sistema.

La arquitectura de un sistema con gran cantidad de software puede describirse mejor a través de cinco vistas interrelacionadas, como se muestra en la Figura 5.29. Cada vista es una proyección de la organización y la estructura del sistema, centrada en un aspecto particular de ese sistema.

Cada una de las cinco vistas puede existir por sí misma, de forma que diferentes usuarios puedan centrarse en las cuestiones de la arquitectura del sistema que más les interesen. Estas cinco vistas también pueden interactuar entre sí.

5.4.1. Vista de Casos de Uso

Comprende los casos de uso que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de las pruebas. Esta vista no especifica realmente la organización de un sistema software. Más bien, existe para especificar las fuerzas que configuran la arquitectura del sistema. Con UML, los aspectos estáticos de esta viste se capturan en los diagramas de casos de uso; los aspectos dinámicos de esta vista se capturan en los diagramas de interacción, diagramas de estados y diagramas de actividades.

5.4.2. Vista de Diseño

Comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución. Esta vista soporta principalmente los requisitos funcionales del sistema, entendiendo por ello los servicios que el sistema debería proporcionar a sus usuarios finales. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de clases y objetos; los aspectos dinámicos se capturan en los diagramas de interacción, diagramas de estados y diagramas de actividades.

5.4.3. Vista de Procesos

Comprende los hilos y procesos que forman los mecanismos de sincronización y concurrencia del sistema. Esta vista cubre principalmente el funcionamiento, capacidad de crecimiento y rendimiento del sistema. Con UML, los aspectos estáticos y dinámicos de esta vista se capturan en el mismo tipo de diagramas que la vista de diseño, pero con énfasis en las clases activas que representan estos hilos y procesos.

5.4.4. Vista de Implementación

Comprende los componentes y archivos que se utilizan para ensamblar y hacer disponible el sistema físico. Esta vista se preocupa principalmente de la gestión de configuraciones de las distintas versiones de un sistema, a partir de componentes y archivos un tanto independientes y que pueden ensamblarse de varias formas para producir un sistema en ejecución. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de componentes; los aspectos dinámicos de esta vista se capturan en los diagramas de interacción, diagramas de estados y diagramas de actividades.

5.4.5. Vista de Despliegue

Contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema. Esta vista se preocupa principalmente de la distribución, entrega e instalación de las partes que constituyen el sistema físico. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de despliegue; los aspectos dinámicos de esta vista se capturan en los diagramas de interacción, diagramas de estados y diagramas de actividades.

5.5. UML EJECUTABLE

Los orígenes de lo que hoy se conoce como UML ejecutable son anteriores al propio lenguaje. Se remontan a los trabajos realizados por Shlaer y Mellor y su propuesta para construir modelos de análisis orientados a objetos especificados formalmente [SM88] [SM91]. Una característica de sus modelos es que tenían una semántica bien definida. En teoría estos modelos eran potencialmente ejecutables, aunque no existían herramientas que permitiesen ejecutarse.

En 1993 cuando IBM definió *Process Specification Language* (PSL) y construyó un prototipo Smalltalk que permitía ejecutar modelos Shlaer-Mellor. Influenciados por este prototipo, en la empresa Kennedy Carter comenzaron a desarrollar la versión inicial de lo que posteriormente se convertiría uno de los primeros lenguajes de acción UML: *Action Specification Language* (ASL).

Con la adopción de UML por parte del OMG en 1997 como lenguaje estándar de modelado Software, Steve Mellor junto con miembros de Kennedy Carter desarrollaron una propuesta para adaptar el proceso Shlaer-Mellor y formalizar su uso con UML [WM99]. Se detectó que era necesario aplicar varias restricciones y añadir determinadas características a los modelos UML para hacerlos ejecutables. Esto llevó a formar un consorcio en el seno del OMG denominado *Action Semantics* que trabajó desde 1998 a 2001 para desarrollar la especificación del mismo nombre: *Action Semantics*. Esta especificación fue incluida dentro del estándar UML en su versión 1.5.

UML Ejecutable puede entenderse como una manera de utilizar UML, como un perfil del lenguaje o como la manera formal en la cual pueden especificarse un conjunto de reglas que indican cómo los elementos de UML encajan unos con otros con un propósito concreto: definir la semántica de los modelos construidos con precisión [MB02]. Para ello, no sólo utiliza *Action Semantics* para especificar con precisión la semántica de acción de los modelos, sino que utiliza un subconjunto de UML eliminando las construcciones que restan precisión al lenguaje. Desde el punto de vista de MDA, UML ejecutable es la única aproximación en la actualidad que hace uso de *Action Semantics* para modelar el comportamiento en los PIM.

5.5.1. La Propuesta de UML Ejecutable

UML Ejecutable se define como un perfil de UML que define una semántica de ejecución para un subconjunto de UML [Mel04]. Para ello lo que hace, es tomar un subconjunto del lenguaje UML que elimina las construcciones semánticamente ambiguas y añadirle la semántica de ejecución a través de *Action Semantics*. La propuesta resultante es *computacionalmente completa*. Esto signfica que es lo suficientemente expresiva para la definición de modelos que pueden ser ejecutados en un ordenador, bien a través de su interpretación o como programas equivalentes generados a partir de los modelos a través de algún tipo de transformaciones automatizadas.

Dado que no todos los elementos de UML tienen semántica de ejecución no todos son utilizados en UML ejecutable. Esto no significa que dichos elementos y los diagramas que los representan no tengan valor de cara a construir modelos en UML ejecutable, su valor será el mismo que con UML convencional: artefactos que ayudan a la comprensión de los sistemas, que lo documentan y que facilitan la comunicación. Sin embargo, al no ser adecuados para la especificación de modelos ejecutables, no resultan de interés en el estudio de la propuesta de UML Ejecutable.

Este es el caso de los diagramas de componentes y de despliegue. Los únicos diagramas estructurales UML utilizados son los diagramas de clases para especificar las características estructurales estáticas de los conceptos que se capturan en los dominios. Los diagramas de casos de uso sí son utilizados en UML ejecutable. Aunque los casos de uso se modelan con una notación informal sirven para enlazar los modelos ejecutables que realizan cada escenario con los requisitos funcionales que representan. Son por lo tanto valiosos en UML ejecutable para trazar los requisitos a través de los modelos ejecutables [RFW+04].

Los diagramas de actividad tampoco son considerados en las propuestas de UML Ejecutable de Steve Mellor [MB02] y de Kennedy Carter [RFW+04]. La razón es que antes de UML 2.0 los diagramas de actividad eran considerados como una vista extendida de los diagramas de estados y en UML ejecutable se prefieren éstos últimos para representar el comportamiento de los objetos. No obstante, en UML 2 el modelo de actividades sufrió importantes modificaciones. En lugar de ser un caso especial de máquinas de estados, las actividades UML 2 presentan un modelo de ejecución propio y son el mecanismo elegido para coordinar acciones. En versiones anteriores de *Action Semantics*, las acciones eran agregadas por una metaclase `Procedure` que representaba el cuerpo de las operaciones. En UML2 las acciones son contenidas en comportamientos `Behavior`. El principal mecanismo para agrupar acciones son las actividades `Activity`, que son un tipo de comportamiento. Para representar el cuerpo de las operaciones en UML 2 se asocian comportamientos a las mismas.

Los diagramas UML utilizados en UML Ejecutable para modelar comportamiento son los diagramas de estados. El enfoque utilizado es: para modelar el comportamiento de los objetos que cambia con el tiempo, se utilizan diagramas de estados combinados con un lenguaje de acción. Para el comportamiento que no es dependiente del estado de los objetos, se utiliza únicamente el lenguaje de acción [RFW+04].

Los diagramas de interacción no son utilizados en UML Ejecutable para modelar comportamiento de una manera formal. No obstante, los diagramas de interacción sí son considerados importantes artefactos de modelado para visualizar la dinámica de los dominios modelados [MB02]. Se utilizan diagramas de comunicación para describir las comunicaciones existentes entre instancias de máquinas de estados y los diagramas de secuencia para describir las secuencia temporal de interacciones entre objetos.

6. OBJECT CONSTRAINT LANGUAGE: OCL 2.0

OCL es un lenguaje para la descripción textual precisa de restricciones que se aplican a los modelos gráficos UML. Fue desarrollado por IBM y adoptado en octubre de 2003 por el grupo OMG como parte de UML 2.1 (Capítulo 5).

El lenguaje OCL puede ser utilizado para diferentes propósitos:

- Como lenguaje de consulta.
- Dentro del modelo de clase para expresar invariantes sobre clases y tipos.
- Para especificar tipos invariantes para Estereotipos.
- Para describir pre y postcondiciones sobre Operaciones y Métodos.
- Para describir controles.
- Para especificar objetivos para mensajes y acciones.
- Para especificar restricciones sobre operaciones.
- Para especificar reglas de derivación de atributos para una expresión sobre el modelo UML.

6.1. FUNDAMENTOS DE OCL

6.1.1. Expresiones, tipos y valores en OCL

En OCL [WK03] cada valor, ya sea un objeto una instancia de un componente o un valor de datos, tiene un tipo que define que operaciones pueden ser aplicadas al objeto. Los tipos se dividen dentro de dos grupos: tipos predefinidos en la biblioteca estándar y tipos definidos por el usuario.

Dentro de los tipos predefinidos podemos distinguir entre tipos básicos y de colleccion. Los tipos básicos son Integer, Real, String y Boolean, cuyas definiciones son similares a otros lenguajes. Dentro de las colecciones están Collection, Set, Bag, OrdenedSet y Sequence. Las colecciones se utilizan para especificar exactamente los resultados de navegación a través de las asociaciones en el diagrama de clases.

Los tipos definidos por el usuario se definen por el usuario en los diagramas UML. Cada elemento instanciable del modelo, es decir, cada clase, interfaz, componente o tipo de datos, en un diagrama UML es automáticamente un tipo de OCL.

Las expresiones OCL representan un valor, por tanto tiene un tipo ya sea un tipo definido por el usuario o un tipo predefinido. Además cada expresión tiene un resultado que será el valor que resulta de evaluar la expresión. El tipo del resultado es igual al tipo de la expresión.

OCL distingue entre tipos de valores y tipos de objetos. Ambos son tipos, ambos especifican instancias, pero hay una diferencia importante: los tipos de valor definen instancias que no cambian. Por ejemplo si tenemos un Integer con valor 1 siempre tendrá ese valor. En cambio los tipos de objetos o "Classifiers" representan tipos que definen instancias que pueden cambiar su valor o valores. Por ejemplo: una instancia de la clase Persona puede cambiar el valor del atributo Edad pero seguirá siendo la misma instancia. Esta diferencia se puede explicar en otros términos, por ejemplo Martin Fowler [Fow99] llama a los tipos de objetos, referencia de objetos y a los tipos de valor, objetos valor.

Otra característica es que los tipos de valor suponen un identidad. Para un tipo de valor, el valor identifica la instancia, de ahí el nombre. Dos ocurrencias de un tipo valor que tenga el mismo valor son por definición la misma instancia. Dos ocurrencias de un tipo objeto son solamente la misma instancia si tienen la misma identidad. Por tanto, los tipos valor tienen identidad basada en valor, y los tipos de objeto tienen identidad basara en la referencia.

Cada uno de los tipos básicos tiene una serie de operaciones. Además dentro de las operaciones existen unas reglas de precedencia. Dentro de los operadores también se puede utilizar los operadores infijos.

6.1.2. Tipos definidos por el usuario

Cuando se define un tipo en un diagrama UML, un tipo de usuario, se le otorgan una serie de características. Cada una de estas características puede ser utilizada en una expresión OCL. Las características de un tipo definido por el usuario incluye:

- Atributos.
- Operaciones.
- Atributos de Clase.
- Operaciones de Clase.
- Extremos de asociaciones que derivan de las asociaciones y las agregaciones.

Los atributos de un tipo definido por el usuario puede ser utilizado en expresiones escribiendo un punto seguido del nombre del atributo. De forma similar, las operaciones de este tipo también se pueden utilizar en las expresiones. Existe una restricción, al ser OCL un lenguaje libre de efectos laterales, no está permitido el cambio de un objeto. Solamente se pueden realizar operaciones de consulta, que retornan un valor pero no cambian nada. De acuerdo con la especificación UML, cada operación tiene una etiqueta denominada “isQuery”, cuyo valor verdadero, indica que la operación no tiene efectos laterales y puede ser utilizada en expresiones.

La notación del punto utilizada para hacer referencia a los atributos se utiliza también para hacer referencia a las operaciones. No obstante, el nombre de la operación, va seguida siempre de paréntesis, que encierran los argumentos opcionales de la operación, pero los paréntesis son obligatorios tanto si hay o no hay argumentos. Esto es debido a que es necesario distinguir entre atributos y operaciones, dado que UML permite atributos y operaciones con nombres idénticos.

La visibilidad de los atributos y las operaciones se ignora por defecto en OCL. Opcionalmente OCL puede utilizar reglas dadas en la especificación UML. En este caso un atributo privado no puede utilizarse en una expresión OCL, dado que no es visible en la instancia contextual.

Las operaciones de clase y los atributos de clase también pueden utilizarse en expresiones. La sintaxis para referirse a los atributos de clase o operaciones será el nombre de la clase seguido por dos puntos y el nombre del atributo u operación, y sus parámetros.

Otra de las características de los tipos definidos por el usuario se deriva de las asociaciones del modelo de clases. Cada asociación tiene un número de extremos. Cada extremo tiene una multiplicidad, un tipo al que está conectado y un orden opcional, además también tiene un nombre, se llama rol (rolename). Conceptualmente un extremo de una asociación define una característica de la clase conectada a otros extremos de asociación.

Los extremos de asociación se pueden utilizar para navegar de una objeto a otro del sistema, por esto se denominan navegadores (navigations). Si no existe el nombre del extremo, se utiliza el tipo de conexión, pero entonces es obligatoria la definición de un rol. Los navegadores son tratados igual que los atributos, utilizando la notación del punto para referirse a ellos.

Existe un tipo definido por el usuario especial, el tipo enumeración (enumeration). Este tipo se utiliza habitualmente como un tipo para los atributos. Se define dentro de un diagrama de clase de UML usando el estereotipo de enumeración. El valor definido en la enumeración puede ser utilizado dentro de una expresión OCL.

6.1.3. Tipos Collection

En los sistemas orientados a objetos, la manipulación de colecciones de objetos es bastante común, dado que las relaciones uno-a-uno son raras, muchas de las asociaciones se definen como una relación entre un objeto y una colección de otros objetos. Para manipular estas colecciones, OCL ha predefinido una serie de tipos para tratar con colecciones, conjuntos o ambos.

Tenemos cinco tipos de colecciones, que son:

- Set: es una colección que contiene instancias de un tipo válido OCL. Un conjunto no puede contener elementos duplicados y además los elementos no están ordenados.
- OrderedSet: igual que Set pero sus elementos están clasificados.
- Bag: es una colección que puede contener elementos duplicados, es decir, una misma instancia puede aparecer mas de una vez. Habitualmente es el resultado de combinar navegadores. Los elementos no están ordenados.
- Sequence: igual que Bag pero clasificado.
- Collection: supertipo abstracto de los cuatro anteriores.

Hay que recalcar que los elementos de las colecciones OrderedSet y Sequence están clasificados pero no ordenados, es decir, los elementos tienen un número de secuencia como los elementos de un array en los lenguajes de programación.

6.2. RELACIÓN CON EL METAMODELO UML

En la especificación de OCL tanto los conceptos como las relaciones entre los mismos se han expresado en forma de un metamodelo del MOF (Capítulo 7). Esta presentación en forma de metamodelo del MOF, al ser más formal, permite la correspondencia (mapping) con un dominio semántico con mucha más facilidad. A continuación se describen los elementos de dicha correspondencia.

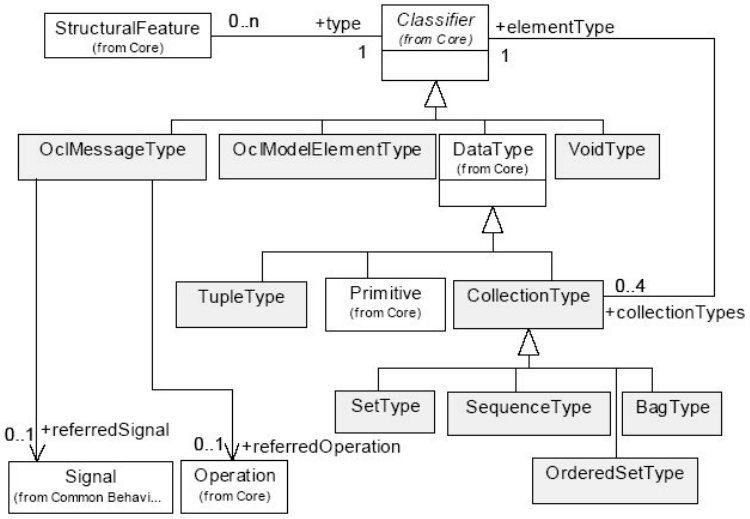


Figura 6.1.Metamodelo de los tipos de OCL.

- El paquete de Tipos: OCL es un lenguaje tipado, cada expresión tiene un tipo que se declara explícitamente o puede ser derivado estáticamente. Al realizar la evaluación de una expresión se devuelve un valor de este tipo. El metamodelo para los tipos OCL se muestra en la Figura 6.1.
- El paquete de expresiones: Este paquete define la estructura que las expresiones OCL pueden tener. Las relaciones de herencia entre todas las clases de este paquete se muestra en la Figura 6.2.

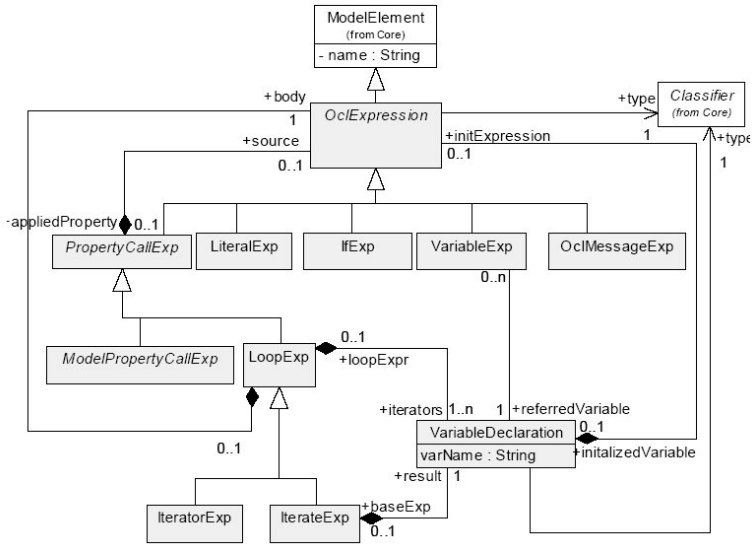


Figura 6.2.Metamodelo de las expresiones de OCL.

7. META-OBJECT FACILITY (MOF)

El Meta-Object Facility (MOF) es un estándar del Object Management Group. Originalmente estaba en la especificación de UML (capítulo 5).

MOF juega un papel fundamental en el ámbito del MDA (capítulo 3) dado que a través de MOF se define UML, es decir que MOF se utiliza como Metamodelo para definir UML.

7.1. DEFINICIÓN DE METAMODELO

Un metamodelo es un modelo de un modelo. Por tanto un modelo es una abstracción y un metamodelo es una abstracción de mayor nivel, cuyo propósito es definir las propiedades del modelo en sí mismo. Algunos usos de los metamodelos son:

- Como un esquema de datos semánticos que necesitan ser intercambiados o almacenados.
- Como un lenguaje que soporta un método particular o proceso
- Como un lenguaje para expresar semántica adicional de información existente

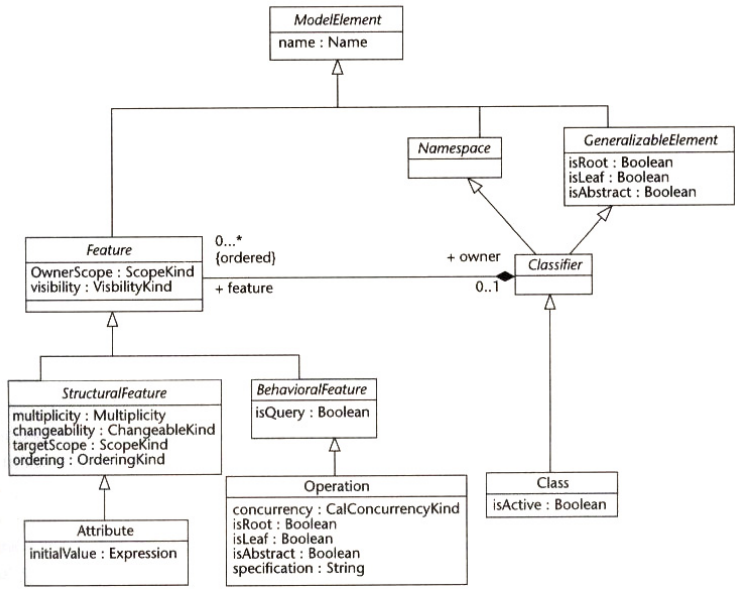


Figura 7.1.Fragmento del metamodelo de UML.

Por ejemplo, si nos centramos en UML (Figura 7.1), un modelo de un banco incluye elementos como Cuenta, Cliente, Fondo, y otros elementos similares que aparecen en un banco. Un modelo de modelos, es decir un metamodelo, incluye elementos como Clase, Atributo, Operación, Parámetro, Asociación y otros elementos similares que aparecen en un modelo.

7.2. DEFINICIÓN DE META-OBJECT FACILITY (MOF)

Como se ha comentado en la introducción MOF es un metamodelo para definir UML. Por ejemplo en la Figura 7.1, se muestra un fragmento del metamodelo de UML que define el modelo para la construcción de clases. El metamodelo de UML define la sintaxis abstracta de UML, y utiliza MOF para definir formalmente la sintaxis y las construcciones del modelo. En otras palabras, con MOF se define formalmente las construcciones. Además un metamodelo especifica algunos elementos semánticos de forma informal utilizando el lenguaje natural. Esta combinación de definiciones formales e informales es a lo que se denomina MOF metamodelo, en algunos casos también se denomina "MOF model".

MOF parte de la premisa de que puede existir más de un buen modelo y por tanto debe ser más que un lenguaje de modelado [Fra03]. Se necesitan diferentes conjuntos de construcciones de modelado para diferentes funciones. Un conjunto de construcciones de modelado, por ejemplo, para unos datos relacionales debe incluir tabla, columna, clase y otros. Un conjunto de construcciones de modelado para el modelo de clases de UML debe incluir clase, atributo, operación, asociación y otros. Por tanto, para describir un tipo de modelo particular tenemos que describir el conjunto de construcciones de modelado que compone dicho tipo de modelo. MOF define una forma global para describir las construcciones de modelado, sin tener en cuenta el ámbito de dichas construcciones.

MOF toma la idea de construcciones de clases del modelo de objeto de UML y los muestra como un medio común para describir la sintaxis de las construcciones de modelo. Por tanto, los metamodelos MOF son como modelos de clase de UML, y se pueden utilizar herramientas de este para crearlos. Con MOF, se modela una construcción de modelado como una clase y las propiedades de la construcción como los atributos de la clase. Las relaciones entre las construcciones se definen como relaciones. Además se utiliza OCL (capítulo 6) para aumentar la precisión del modelo.

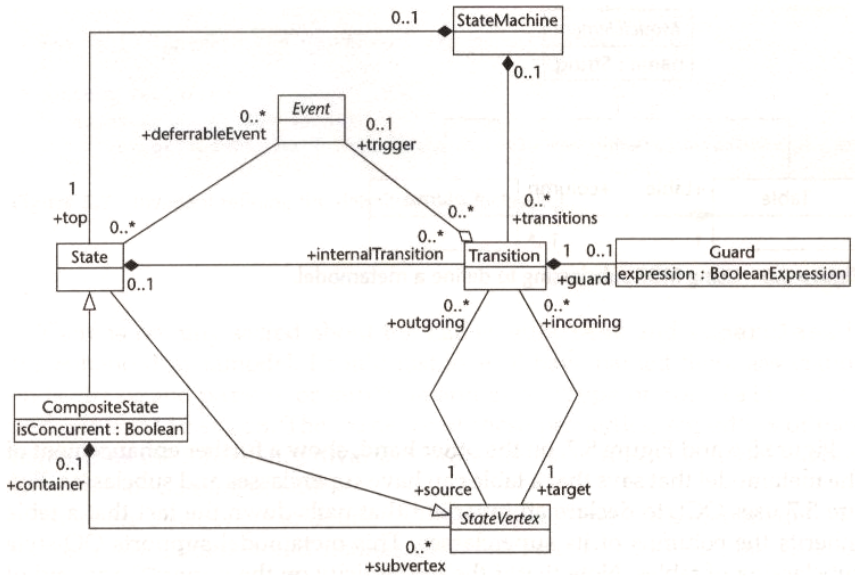


Figura 7.2. Metamodelo para los diagramas de estados de UML.

En la Figura 7.2, muestra un fragmento de la sintaxis abstracta del metamodelo para el diagrama de estados, que es una parte del metamodelo de UML. El diagrama de estado es diferente del modelo de Clase, y por tanto el metamodelo de UML es diferente del que se mostraba en la Figura 7.2. Estos metamodelos son diferentes porque describen construcciones que componen diferentes tipo de modelos.

7.2.1. Metanivel MOF

MOF esta diseñado dentro de una arquitectura de cuatro metaniveles denominados M3, M2, M1 y MO, en el ámbito de MDA. (Tabla 7.1)

Metanivel	Descripción
M3	MOF
M2	Metamodelos
M1	Modelos
M0	Objetos y Datos

Tabla 7.1. Metaniveles de MOF.

- **Metanivel M3**. Este metanivel es el MOF, por tanto es el conjunto de construcciones utilizadas para definir metamodelos. Por ejemplo: Clase MOF, Atributo MOF, Asociaciones MOF, etc.
- **Metanivel M2**: Este metanivel son los Metamodelos que consisten en instancias de las construcciones MOF. Por ejemplo: Clase UML, Atributo UML, Estado UML, Actividad UML, etc.
- **Metanivel M1**: Este nivel son los Modelos, que consisten en instancias de las construcciones del metamodelo M2. Por ejemplo: Clase Cuenta en un banco, clase Cliente, tabla Empleados, etc.
- **Metanivel M0**: Este nivel son los Objetos y Datos, es decir instancias de las construcciones del modelo M1. Por ejemplos: Cliente Cris Pelayo, Cuenta 111118, Empleado Bego García-Bustelo, etc.

7.3. IMPLEMENTACIONES DE MOF

A continuación se presentan las dos implementaciones de MOF libres más conocidas. Debe destacarse que detrás de cada una de ellas existe una gran empresa del sector: Sun en un caso e IBM en el otro. Ambas implementaciones forman parte de la arquitectura de sendos entornos de desarrollo.

7.3.1. MDR

Metadata Repository (MDR) es una implementación del estándar MOF [MDR] desarrollada dentro de la plataforma de herramientas de desarrollo NetBeans [Net]. En su versión actual, implementa la versión 1.4 de MOF y permite importar y exportar modelos utilizando XMI versión 1.1 y 1.2. Además, puede accederse a los metadatos del repositorio programáticamente utilizando la API JMI, tanto en su versión específica al metamodelo como en su versión reflectiva genérica.

Existe un catálogo de implementaciones de metamodelos basados en MOF para utilizar con MDR. Estos metamodelos son cargados por MDR en su forma XMI complementada con etiquetas específicas JMI para guiar el proceso de

7.3.2. EMF

Eclipse Modeling Framework (EMF) es un framework de modelado y una utilidad de generación de código desarrollada como parte del proyecto Eclipse [Ecl06a]. El *framework* se edifica en torno la implementación de un metamodelo denominado *ECore*. Dicho metamodelo es similar a MOF aunque no son compatibles [BSM+03]. Ofrece servicios para importar y exportar modelos utilizando XMI. Además, soporta importar modelos utilizando interfaces Java anotados. Ofrece mecanismos que permiten generar el código Java que implementa los modelos especificados. Si bien este código está generado en términos de interfaces e implementaciones, no es compatible con JMI.

8. XML METADATA INTERCHANGE (XMI)

XMI es un estándar del Object Management Group (OMG) para el intercambio de información vía XML [XML04]. Puede ser utilizado para todos los metadatos de los metamodelos que estén expresados en MOF. El uso más común de XMI [XMI05] es como formato de intercambio de modelos UML, es decir, se puede utilizar para la serialización de metamodelos.

8.1. DEFINICIÓN DE XML METADATA INTERCHANGE (XMI)

Desde la perspectiva de modelado de OMG, los datos se dividieron en modelos abstractos y modelos concretos. Los modelos abstractos representan la información semántica, mientras que los modelos concretos representan diagramas visuales. Los modelos abstractos son instancias de lenguaje de modelado basado en MOF como puede ser UML. Para los diagramas se utiliza el XMI.

El propósito de XMI es facilitar el intercambio de los metadatos entre las herramientas de modelado UML y los repositorios de metadatos MOF en entornos heterogéneos.

XMI integra cuatro estándar: XML, UML , MOF y la correspondencia entre MOF y XMI

La integración de estos cuatro estándares dentro de XMI permite herramientas de desarrollo para sistemas distribuidos de modelos de objetos y otros metadatos.

8.1.1. XMI y DTD

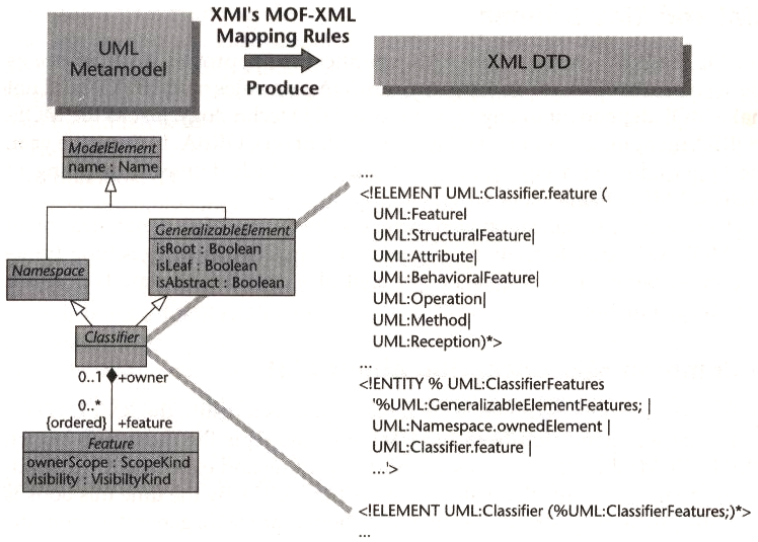


Figura 8.1.Aplicación de las reglas de la transformación XMI sobre un metamodelo UML.

Cuando OMG aplicó XMI al metamodelo de UML crearon un DTD para el intercambio de modelos UML. En la Figura 8.1 se muestra como se aplica XMI a un elemento "Classifer" de un metamodelo UML, se muestra sólo un fragmento del elemento DTD generado para representar un "Classifier".

XMI define un conjunto de reglas para producir documentos XML que se validarán contra los DTDs generados. Estas reglas son necesarias debido a que las reglas de producción de DTDs especifican solamente la sintaxis de XML para los modelos de representación. Por tanto las reglas de producción de documentos especifican la semántica de la información XML que será introducida dentro de los huecos definidos por un XMI DTD, siendo este un DTD de XML generado a través de las reglas de transformación de XMI a MOF.

Las reglas XMI para generar DTDs y documentos no sólo hacen posible generar DTDs desde un metamodelo automáticamente, sino que permiten generar código de implementación de los modelos y desde un repositorio de metadatos.

8.1.2. XMI y XML Schema

En 2001 el World Wide Web Consortium (W3C), aprobó XML Schema como sucesor de XML DTDs, además adopto la especificación XMI que define una transformación de MOF a XML Schema.

9. SISTEMAS DE PERSISTENCIA

En este capítulo se definen los conceptos de persistencia y otros relacionados. Además se estudia las forma de realizar el desarrollo de Sistemas Persistentes [OMA+99], por último se estudian los frameworks de persistencia con mayor presencia en el mercado [Lop06].

9.1. DEFINICIÓN DE PERSISTENCIA

El concepto de persistencia de datos hace referencia al tiempo en el que los datos existen y son utilizables [AMP86]. Por tanto podemos determinar que el término persistencia se refiere a mantener los valores de los datos durante todo su tiempo de vida.

El concepto de persistencia se encuentra recogido como uno de los tres principios secundarios del modelo de objetos [Boo94]. En este contexto se entiende persistencia como la cualidad de un objeto de mantener su identidad y relaciones con otros objetos con independencia del sistema o proceso que lo creó [Mar01].

Dentro de la persistencia existe el concepto de *Persistencia ortogonal*, que según las conclusiones expuestas en [AM95] podemos determinar que la persistencia de un sistema es ortogonal si cumple:

- Todos los objetos pueden persistir, independientemente de su tipo
- No hay diferencias para el programador a la hora de manejar datos persistentes y no persistentes
- El mecanismo para identificar los objetos persistentes no está relacionado con el sistema de tipos, es decir, es ortogonal al universo del discurso del sistema.

Se debe definir tres conceptos muy relacionados con los Sistemas de Persistencia que son: base de datos, sistemas de gestión de bases de datos y sistemas de indexación, es decir, almacenamiento de datos y objetos y formas de acceso a los mismos.

9.2. ALMACENAMIENTO Y ACCESO A LOS DATOS Y OBJETOS

9.2.1. Base de Datos

Una base de datos puede definirse como una colección de registros almacenados de una manera sistemática en un ordenador, de tal modo que una aplicación informática pueda consultarla para utilizar los datos almacenados. Para optimizar la recuperación y ordenación, cada registro se organiza habitualmente como un conjunto de elementos de datos (o hechos). La aplicación informática utilizada para gestionar y consultar la base de datos se conoce como Sistema Gestor de Base de Datos (SGBD).

Muchos autores consideran que una colección de datos constituye una base de datos únicamente si presenta ciertas propiedades, por ejemplo, si se gestiona la integridad y calidad de los datos, si se permite el acceso a los mismos por varios usuarios, si presentan un esquema, o si se dispone de un lenguaje de consulta. Sin embargo, no existe un consenso a la hora de definir estas propiedades.

Existe normalmente una descripción de la estructura de los tipos de los datos almacenados en la base de datos esta descripción se conoce como esquema. El esquema describe los objetos que se representan en la base de datos y las relaciones entre ellos. Existen diversas formas de organizar un esquema, o lo que es lo mismo, de modelar la estructura de la base de datos. Esto da lugar a la aparición de los modelos de bases de datos (o modelos de datos) entre los que se encuentran el jerárquico, el modelo en red, el relacional, el multidimensional o el orientado a objetos.

El modelo de datos que domina el mercado actualmente es el relacional. Este modelo fue introducido en un artículo académico por E. F. Codd en 1970 [Cod70] y su principal objetivo era conseguir que los sistemas de gestión de bases de datos fuesen independientes de las aplicaciones que los usasen. Para ello plantea abstraer la descripción de la estructura de la información separándola de la descripción de los mecanismos físicos de acceso. Para describir el modelo abstracto de la información propone un modelo matemático definido en términos de lógica de predicados y teoría de conjuntos.

El modelo de bases de datos orientadas a objetos es el resultado de la aplicación del paradigma de la orientación a objetos a la tecnología de base de datos. Su motivación es acercar el mundo de las bases de datos a los lenguajes de programación, donde el paradigma de objetos domina el mercado en la actualidad, permitiendo evitar la sobrecarga de convertir la información entre su representación en la base de datos como filas en tablas y su representación en el lenguaje de programación como objetos, es decir, conseguir la desadaptación de impedancias [Mai89].

9.2.2. Sistema Gestor de Base de Datos

Un sistema gestor de bases de datos (SGBD) es una aplicación informática (o un conjunto de ellas) diseñada para gestionar una base de datos y para ejecutar sobre ella las operaciones que soliciten diversos clientes.

Las funcionalidades básicas de cualquier sistema gestor de bases de datos incluyen:

- Un lenguaje de modelado para describir el esquema de cada base de datos gestionada con el SGBD (lenguaje de definición de datos).

- Un lenguaje que permita realizar consultas sobre la base de datos (lenguaje de consulta de datos).
- Un lenguaje que permita actualizar los datos contenidos en la base de datos (lenguaje de manipulación de datos).
- Un mecanismo de transacciones que idealmente garantice las propiedades ACID (atomicidad, consistencia, integridad y disponibilidad) de la información, con el objeto que asegurar la integridad de los datos independientemente de los accesos concurrentes de los usuarios (control de concurrencia) y de los fallos (tolerancia a fallos).

Los SGBD se clasifican según el modelo de datos que utilicen. El mercado está dominado actualmente por los SGBD relacionales, que utilizan como lenguaje de definición, consulta y manipulación de datos SQL [Ms93] o una variante propietaria del mismo. Son ejemplos de estos sistemas Ingres, Oracle, DB2 o SQL Server.

Los SGBD orientados a objetos, si bien han influido con sus ideas en los SGBD relacionales, han tenido una penetración muy baja en el mercado debido, entre otras razones, al rendimiento notablemente inferior con respecto a los relacionales, a la ausencia de estándares en un primer momento y al fracaso en la adopción de la normalización propuesta después.

9.2.3. Sistema de Indexación

Un índice, en terminología de base de datos, es una estructura de almacenamiento físico empleada para acelerar la velocidad de acceso a los datos. Los índices juegan un papel fundamental en las bases de datos, tanto relacionales como orientadas a objetos, siendo un soporte básico e indispensable para acelerar el procesamiento de las consultas [BO99].

En el ámbito de las consultas en un sistema de persistencia orientado a objetos, conviene resaltar tres características que vienen directamente impuestas por el modelo de objetos [Mar01]:

- El mecanismo de la herencia (Jerarquías de Herencia). La herencia provoca que una instancia de una clase sea también una instancia de su superclase. Esto implica que, el ámbito de acceso de una consulta sobre una clase en general incluye a todas sus subclases, a menos que se especifique lo contrario.
- Predicados con atributos complejos anidados (Jerarquías de Agregación). Mientras que en el modelo relacional los valores de los atributos se restringen a tipos primitivos simples, en el modelo orientado a objetos el valor del atributo de un objeto puede ser un objeto o conjunto de objetos. Esto provoca que las condiciones de búsqueda en una consulta sobre una clase, se puedan seguir expresando de la forma <atributo operador valor>, al igual que en el modelo relacional, pero con la diferencia básica de que el atributo puede ser un atributo anidado de la clase.
- Predicados con invocación de métodos. En el modelo de objetos los métodos definen el comportamiento de los objetos, y al igual que en el predicado de una consulta puede aparecer un atributo, también puede aparecer la invocación de un método.

Las características anteriores exigen técnicas de indexación que permitan un procesamiento eficiente de las consultas bajo estas condiciones [MC99]. Así, son muchas las técnicas de indexación en orientación a objetos que se han propuesto y que clásicamente [BF05] se pueden clasificar en:

- Estructurales. Se basan en los atributos de los objetos. Estas técnicas son muy importantes porque la mayoría de los lenguajes de consulta orientados a objetos permiten consultar mediante predicados basados en atributos de objetos. A su vez se pueden clasificar en:
    - Técnicas que proporcionan soporte para consultas basadas en la Jerarquía de Herencia.
    - Técnicas que proporcionan soporte para predicados anidados, es decir, que soportan la Jerarquía de Agregación.
    - Técnicas que soportan tanto la Jerarquía de Agregación como la Jerarquía de Herencia.

- De Comportamiento. Proporcionan una ejecución eficiente para consultas que contienen invocación de métodos.

El mecanismo de indexación utilizado por un sistema de persistencia debería ser adaptable, permitiendo al usuario del sistema de persistencia cambiar los parámetros relativos al sistema de indexación y adaptar de este modo su comportamiento. En concreto, un mecanismo de indexación extensible debería permitir:

- Añadir al sistema nuevas técnicas de indexación [Sto86]. Dos escenarios en los que sería deseable añadir nuevas técnicas de indexación serían la utilización de un nuevo tipo de dato que sugiera una técnica determinada y la utilización de una técnica específica para el tipo de consulta más frecuentemente realizada en el sistema. Existen diversos sistemas que implementan esta característica, como Starburst [LMP87], POSTGRES [Aok91] u Oracle [ORA], y por algunos generadores de bases de datos como EXODUS [CDF+86].
- Añadir nuevos tipos al sistema permitiendo utilizar sobre ellos las técnicas de indexación existentes.

En [Mar01] se propone un diseño orientado a objetos basado en patrones de diseño [GHJ+95] para evitar el impacto sobre el sistema a la hora de implementar las parametrización descrita. En concreto se propone utilizar el patrón *Strategy* para soportar nuevas técnicas de indexación, y el patrón *Command* para que cada técnica de indexación pueda funcionar con diferentes operadores según el tipo de dato que se indexe.

Por otra parte, existen circunstancias en las que el sistema está capacitado para seleccionar la técnica de indexación idónea en unas circunstancias determinadas, sin necesidad de actuación por parte del usuario. Por ejemplo, el sistema podría seleccionar una técnica de indexación concreta en función de las características de una consulta realizada y del modelo de objetos. Otro ejemplo sería que el sistema detectase cuál es la técnica de indexación que mejor se comporta para un tipo de datos dado. Estos ejemplos sugieren un mecanismo de indexación adaptativo, es decir, capaz de adaptarse de una manera transparente a las necesidades del problema.

9.3. DESARROLLO DE APLICACIONES PERSISTENTES

En este apartado se analizará una muestra representativa de los sistemas más utilizados a la hora de desarrollar aplicaciones persistentes.

El modelo de datos dominante en la actualidad es el modelo relacional, representado a nivel práctico por el lenguaje SQL [MS93]. Tomando el lenguaje Java como ejemplo, el programador suele utilizar SQL, ya sea de un modo directo, utilizando JDBC [FEB03] o SQLJ [CSH+98], o indirecto, empleando algún sistema de traducción objeto-relacional como JDO [JDO03] o Hibernate [BK04] o un framework de persistencia específico.

Otro mecanismo para dar persistencia a una aplicación es utilizar un sistema de almacenamiento basado en ficheros. Centrándonos en Java como ejemplo, esta plataforma incluye una tecnología de serialización de objetos [JOS03]. Adicionalmente, XML [XML04] se está empleando como formato popular de formato de ficheros. En la plataforma Java se ha introducido un sistema de serialización de objetos Java Beans con XML [JOS03].

9.3.1. SQL: Structured Query Language

SQL [MS93] es el lenguaje más utilizado para crear, modificar y recuperar los datos de los sistemas de gestión de bases de datos relacionales.

Sus orígenes se encuentran en la década de los 70, en el trabajo realizado en el centro de investigación San Jose de IBM [IBM], donde se buscaba desarrollar un sistema de bases de datos denominado "System R", basado en el modelo relacional de Codd [Cod70]. El lenguaje *Structured English Query Language* se diseñó con el objeto de manipular y recuperar los datos almacenados en System R. El acrónimo *SEQUEL* fue más tarde reducido a SQL por ser el primero una marca registrada. Aunque el trabajo de Codd tuvo mucha influencia sobre el diseño de SQL, el lenguaje fue diseñado en IBM por Donald D. ChamBerlin y Raymond F. Boyce [CB74], publicándose en 1974 con el objeto de incrementar el interés en SQL.

SQL fue estandarizado por ANSI (American Nacional Standards Institute) en 1986 y por ISO (International Organization for Standardization) en 1987.

Aunque SQL está normalizado tanto por ANSI como por la ISO, existen numerosas extensiones y variaciones de los estándares. La mayor parte de éstas son de una naturaleza propietaria, como por ejemplo PL/SQL de Oracle o Sybase, SQL PL/SQL (SQL Procedural Language) de IBM y Transact.-SQL de Microsoft. Tampoco es extraño que las implementaciones comerciales no soporten características básicas del estándar, como por ejemplo los tipos DATE o TIME, y utilicen en su lugar alguna variación propietaria. En consecuencia, el código SQL rara vez puede ser portado entre sistemas de bases de datos sin tener que realizar modificaciones sustanciales. Existen varias razones que justifican esta falta de portabilidad:

- La complejidad y tamaño del estándar hace que la mayor parte de las bases de datos no lo implementen en su totalidad.
- Los estándares no especifican el comportamiento de la base de datos en diversas áreas fundamentales, por ejemplo en los índices, dejando esta decisión en manos de las diferentes implementaciones.
- Si bien el estándar especifica con precisión la sintaxis que un sistema de bases de datos debe implementar para ser conforme con la norma, la especificación de la semántica de las construcciones del lenguaje está mucho menos definida, existiendo áreas de ambigüedad.
- Muchas empresas desarrolladoras de sistemas de gestión de bases de datos habían desarrollado sus sistemas antes de la aparición del estándar. Por ello, se han visto a llegar a un acuerdo entre romper la compatibilidad hacia atrás con sus sistemas o implementar correctamente el estándar.

SQL fue diseñado con un propósito específico: recuperar los datos contenidos en una base de datos relacional. Se trata de un lenguaje declarativo basado en conjuntos, al contrario que otros lenguajes imperativos como C++ [Str98] o Java [GJS04], diseñados para poder resolver un conjunto mucho más amplio de problemas. Para integrar SQL con éstos lenguajes de programación existen varias alternativas:

- Existen extensiones propietarias al estándar, como PL/SQL, que convierten SQL en un lenguaje de programación completo.

- Incrustar el código SQL dentro del lenguaje de programación utilizado para desarrollar la aplicación, denominado en este caso ``lenguaje anfitrión''.
- SQL puede ser utilizado indirectamente desde el lenguaje de programación, haciendo uso de un mecanismo de persistencia en el que se delegan los detalles de implementación de la misma. Dentro de los productos utilizados habitualmente en el desarrollo empresarial, esta es la opción más avanzada y donde más alternativas existen.

9.3.2. JDBC: Java Database Connectivity

Sun Microsystems introdujo con la versión 1.1 de la plataforma Java la API (Application Programming Interface) Java Database Connectivity (JDBC) [FEB03] como la manera estándar de comunicarse con una base de datos relacional utilizando el lenguaje SQL. Actualmente, esta API es considerada, con diferencia, la más exitosa dentro de la plataforma Java [Jor04].

Durante el diseño de la API JDBC, Sun consideró tres objetivos como fundamentales [Ree00]:

- JDBC debería ser una API a nivel de SQL.
- JDBC debería tener en cuenta las características de las APIs de bases de datos existentes.
- JDBC debería ser simple.

Una API a nivel de SQL significa que JDBC permite al desarrollador construir sentencias SQL e incrustarlas dentro de llamadas Java a la API. Es decir, se utiliza básicamente SQL pero JDBC suaviza la transición entre el mundo de las bases de datos y de las aplicaciones Java. Por ejemplo, los resultados de la base de datos son devueltos como objetos Java y los problemas de acceso que sucedan son notificados mediante excepciones.

Por otra parte, la idea de Sun de proporcionar una API de acceso a bases de datos universal no es nueva, puesto que mucho antes de su aparición ya existía una gran confusión provocada por la proliferación de APIs de acceso a bases de datos propietarias. De hecho, Sun recogió a la hora de diseñar JDBC las mejores características de otra API de este tipo: Open DataBase Connectivity (ODBC) [San98]. ODBC fue desarrollada para crear un único estándar para el acceso a bases de datos en entornos Windows. Aunque la industria aceptó ODBC como el medio principal de acceso a bases de datos en Windows, este estándar no encaja bien en el mundo de Java, principalmente por tratarse de una API C que requiere APIs intermedias para otros lenguajes, además de ser una API con una notable complejidad y tamaño.

Además de ODBC, JDBC está enormemente influido por otra API existente, la X/OPEN SQL Call Level Interface (CLI). Sun quiso reutilizar las principales abstracciones de estas APIs con el fin de facilitar la aceptación de JDBC por parte de los vendedores de bases de datos, además de facilitar la migración de los desarrolladores de ODBC y SQL CLI.

Sun se dio cuenta además de que derivar una API de las APIs existentes podía favorecer un desarrollo rápido de las soluciones que utilizasen motores de bases de datos que soportasen los antiguos protocolos. En concreto, se desarrolló un puente ODBC que asocia llamadas JDBC a llamadas ODBC, dando de este modo a las aplicaciones Java acceso a cualquier sistema gestor de bases de datos que soportase ODBC.

Finalmente JDBC fue diseñado con el fin de que fuese tan sencilla como fuera posible ofreciendo a los desarrolladores la máxima flexibilidad. Un criterio clave [Ree00] utilizado fue el de ofrecer un interfaz sencillo para las tareas más comunes, mientras que las tareas menos habituales se ofrecían mediante interfaces especializados. Por ejemplo, tres interfaces manejan la gran mayoría de accesos a una base de datos, mientras que para el resto de tareas, menos comunes y más complejas, JDBC ofrece diversos interfaces.

Con respecto a la arquitectura de la API, JDBC se estructura en torno a un conjunto de interfaces que los vendedores de sistemas de bases de datos deben implementar (Figura 9.1). El conjunto de clases que implementan los interfaces JDBC para un SGBD concreto se denomina driver JDBC.

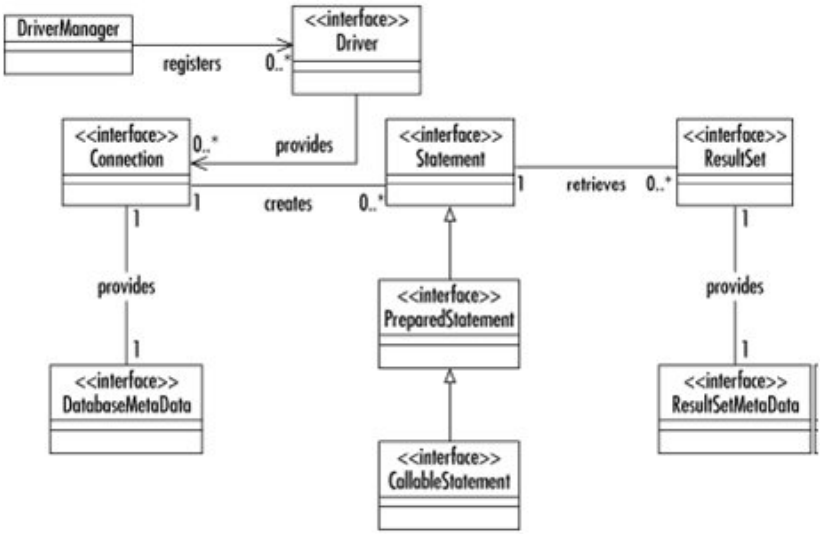


Figura 9.1.Principales clases e interfaces JDBC.

Sun clasifica los drivers JDBC en base a los siguientes tipos [FEB03]:

- **Tipo 1.** Estos drivers implementan un puente para acceder a la base de datos. Un ejemplo de este tipo de driver es el puente JDBC-ODBC que viene con la JDK 1.2. Proporciona una puerta de enlace a la API ODBC. Serán las implementaciones de esta API las que realmente realicen el acceso a la base de datos. Las soluciones basadas en puentes normalmente requieren software instalado en los clientes, lo que implica que no son adecuadas para aplicaciones que no permitan instalar software en el cliente.
- **Tipo 2.** Son drivers nativos. Esto significa que el driver contiene código Java que hace llamadas nativas a C o métodos C++ proporcionados por los vendedores de bases da datos, que son los que realizan realmente el acceso a la base de datos. Esta también solución requiere software instalado en el cliente.
- **Tipo 3.** Estos drivers proporcionan al cliente una API de red genérica que será en los accesos concretos a la base de datos a nivel del servidor. Es decir, el driver JDBC en el cliente utiliza sockets para llamar a una aplicación middleware en el servidor que traduce las peticiones del cliente en una API específica al driver deseado. Este tipo de driver es extremadamente flexible debido a que no requiere código instalado en el cliente y un simple driver puede proporcionar acceso a múltiples bases de datos.

- **Tipo 4.** Estos drivers se comunican directamente con la base de datos utilizando sockets y protocolos de red propietarios del SGBD. Se trata de la solución completamente Java más directa. Debido a que raramente la documentación de esos protocolos de red está disponible, este tipo de driver casi siempre es proporcionado por el fabricante del SGBD.

La principal ventaja de la arquitectura de JDBC es que el desarrollador no tiene que preocuparse de la implementación de las clases que subyacen a la API más allá de las sentencias relativas a la utilización del driver concreto. En el mismo sentido, una aplicación puede cambiar el SGBD utilizada únicamente modificando el código de utilización del driver.

JDBC hace uso de diversas características de la plataforma Java [Kra96], como la gestión automática de memoria, el manejo adecuado de cadenas de caracteres y la API de contenedores para simplificar el uso programático de SQL.

JDBC proporciona además una correspondencia transparente de los tipos básicos de datos tales como String o Int con los tipos SQL correspondientes. En concreto, JDBC define un conjunto de "tipos JDBC" [FEB03] que actúan como posibles intermediarios entre los tipos Java y los tipos SQL, y que son traducidos de una manera transparente al tipo adecuado de la base de datos subyacente por los drivers JDBC.

JDBC gestiona la complejidad del mantenimiento de las conexiones a la base de da-tos y también proporciona un abanico de mecanismos para mejorar el rendimiento y la escalabilidad. Por ejemplo, las "sentencias preparadas" (prepared statements) son habitualmente compiladas y cacheadas por el SGBD y pueden reutilizarse muchas veces con diferentes argumentos. También pueden agruparse numerosas sentencias por lotes con el fin de reducir el número de accesos al servidor.

Para permitir interactuar con las variaciones de los diferentes sistemas de bases de datos SQL, JDBC ofrece un rico interfaz de gestión de metadatos que permite a las aplicaciones descubrir en tiempo de ejecución que características específicas soporta un tipo de base de datos concreta, así como el acceso al esquema de la base de datos.

La API JDBC continúa evolucionando. Las mejoras de la versión 3.0 incluyen pooling de sentencias, que permite almacenar en caché sentencias preparadas para su reutilización por parte de múltiples conexiones lógicas, y también un mejor soporte para características avanzadas del estándar SQL 99 [EM99].

9.3.3. SQLJ

SQLJ [CSH+98] es una tecnología que permite a un programa Java acceder a una base de datos utilizando sentencias SQL incrustadas en el código de la aplicación. SQLJ fue desarrollado por The SQLJ Group, un consorcio compuesto por Sun Microsystems y los principales fabricantes de bases de datos, Oracle, IBM, Compac, Informix and Sybase. Una vez desarrollada la especificación del estándar, ésta fue remitida al Comité técnico H2 sobre Bases de Datos del INCITS (International Committee for Information Technology Standards). La especificación SQLJ comprende tres partes:

- La parte 0 especifica cómo incrustar sentencias SQL en código Java. Únicamente permite incrustar sentencias SQL estáticas, debiendo ser las sentencias SQL dinámicas gestionadas con sentencias JDBC. Soporta correspondencia de tipos entre Java y SQL definido por JDBC.
- La parte 1 abarca el uso de Java desde rutinas SQL. Permite invocar métodos escritos en Java desde código SQL. Los métodos Java proporcionan la implementación de los procedimientos SQL. Para asociar los procedimientos SQL y los métodos Java, que deberán ser estáticos, cada parámetro SQL y su equivalente Java deben ser compatibles así como los retornos de las funciones.
- La parte 2 define extensiones SQL para utilizar clases Java como tipos de datos de SQL. Permite la traducción de tipos definidos por el usuario según la especificación SQL 99 [EM99] a clases Java. También permite importar un paquete Java dentro de la base de datos SQL mediante la definición de tablas conteniendo columnas cuyos tipos de datos se especifican como clases Java. Los tipos estructurados son asociados con clases, los campos con atributos, y los inicializadores (initializers) con constructores.

SQLJ consta de dos componentes: el traductor (translator) y las librerías de tiempo de ejecución (runtime libraries). El traductor es un preprocesador que lee los archivos de código fuente Java que contienen las sentencias SQL incrustadas y las traduce en llamadas a las librerías de tiempo de ejecución. El traductor se encarga también de compilar el código fuente Java resultante. Las llamadas a las librerías de tiempo de ejecución en el código compilado son las que realizan realmente las operaciones con la base de datos.

En un programa SQLJ una sentencia SQLJ puede aparecer en cualquier lugar donde sería válida una sentencia Java normal. Todas las sentencias comienzan con el símbolo reservado *#sql* con el fin de diferenciarlas de las sentencias Java normales.

SQLJ fue concebido como una alternativa a JDBC, si bien SQLJ utiliza JDBC en las rutinas generadas por el preprocesador. A pesar del soporte recibido por grandes empresas como Oracle su impacto en el mercado ha sido notablemente reducido. En [Ree00] se apunta como la principal razón a que se trata de una alternativa basada en un modelo de acceso a bases de datos desfasado: el paradigma utilizado por SQLJ resulta familiar para programadores de C o COBOL pero va en contra de la naturaleza orientada a objetos de Java.

9.3.4. Mecanismo de Serialización de Objetos Java

Se trata del mecanismo estándar de serialización de objetos introducido en la plataforma Java desde su versión 1.1

Una clase se declara serializable implementando el interfaz java.io.serializable. Este interfaz no define ningún método, únicamente identifica los objetos que pueden ser serializables con el objeto de prevenir la serialización de datos sensibles [Jor04].

Presenta un mecanismo de persistencia por alcance [AM95]. Empezando por un objeto raíz, la serialización incluye todos los objetos alcanzables desde éste, transitivamente, desde la raíz, siguiendo aquellos campos del objeto que referencian a otros objetos.

El mecanismo de serialización se basa en el sistema de flujos de la plataforma Java (streams). Para serializar, se escribe en el flujo una representación de cada objeto, incluyendo su clase y sus campos. La deserialización consiste en leer un flujo previamente escrito. La información de la clase se limita al nombre y a la codificación de la signatura de los métodos. En concreto, el código de los métodos no se serializa.

El sistema presenta un mecanismo general para que el desarrollador intervenga en el proceso de serialización, lo que puede ser utilizado, entre otras cosas, para transformar u omitir ciertos campos del objeto.

En la práctica, este mecanismo debe ser utilizado mucho más a menudo de lo que puede parecer en un principio, con el fin de incrementar la eficiencia del mecanismo de serialización. El mecanismo de serialización y deserialización de objetos se realiza en una misma operación, no importa cuantos objetos comprenda el árbol que se almacena. Esto causa graves problemas de rendimiento cuando el número de objetos es numeroso [Jor04].

9.3.5. Serialización de Java Beans con XML

Este mecanismo de serialización fue inicialmente desarrollado para serializar objetos de la API gráfica de Java Swing [DW99]. El objetivo era poder intercambiar modelos de interfaces gráficos entre los diferentes entornos de desarrollo que soportaban su diseño visual. Finalmente, el mecanismo fue introducido como un mecanismo de serialización de Java Beans [Jav96] en la versión 1.4.1 de la plataforma Java.

El mecanismo, en su funcionamiento por defecto, es capaz de serializar y deserializar cualquier objeto que:

- Utilice los patrones de nomenclatura de los Java Beans para definir sus propiedades.
- Todo su estado venga definido en base a dichas propiedades.

El lenguaje XML que describe los objetos serializados, lo que realmente describe es el conjunto de invocaciones que deben realizarse para reconstruir el estado de dicho objeto.

Por ejemplo, en el Código Fuente 9.1 se muestra un botón Swing. Para reconstruirlo únicamente hará falta llamar a su constructor por defecto e invocar al método setText() con el parámetro "Hello, world".

```
<object class="javax.swing.JButton">
   <void method="setText">
          <string>Hello, world</string>
   </void>
</object>
```

Código Fuente 9.1.Ejemplo de un botón Swing serializado con XML.

El sistema utiliza un algoritmo para minimizar el tamaño del fichero XML generado: evita guardar propiedades que tengan valores por defecto. Este mecanismo implica pagar el precio de clonar la estructura inicial de datos, lo que puede resultar un problema de escalabilidad cuando se tienen grandes volúmenes de datos.

Si un objeto no expone todo su estado a través de propiedades Java Beans el sistema por defecto no funciona. En este caso, se permite la personalización de cómo se construye el fichero XML. Para ello, se ofrece una API orientada a objetos con la que construir sentencias Java (invocaciones a métodos, a constructores, establecimiento de propiedades). Utilizando esta API, pueden construirse objetos "delegados" que se asociarán con la clase del objeto que serializan, y que serán los encargados de realizar la generación de sentencias Java o realizar su interpretación. Estos objetos serán invocados durante el proceso de serialización/deserialización de cada objeto de la clase a la que están asociados.

9.3.6. Herramientas de Correspondencia Objeto/Relacional

Las herramientas de correspondencia objeto/relacional (Object/Relational Mapping, ORM), también denominadas mediadores objeto-relacionales, envoltorios o wrappers [Dev97] son productos que ofrecen capas de software orientado a objetos que permiten trabajar sobre una base de datos relacional. Estos sistemas ofrecen al desarrollador la sensación de estar trabajando con un SGBDOO (Sistema de Gestión de Bases de Datos Orientadas a Objetos), ocultando los mecanismos necesarios para traducir las clases en tablas relacionales y los objetos en tuplas. Estas herramientas generalmente emplean SQL para interactuar con la base de datos relacional.

Una solución ORM comprende 3 componentes [BK04] básicos:

- Una API para ejecutar las operaciones CRUD (Create, Retrieve, Update, Delete) básicas sobre los objetos de las clases persistentes.

- Una API o un lenguaje para especificar consultas que hagan referencia a las clases o a las propiedades de las clases.
- Un mecanismo para especificar los metadatos de correspondencia. Las herramientas ORM necesitan un formato de metadatos para que la aplicación especifique cómo se realizará la traducción entre clases y tablas, propiedades y columnas, asociaciones y claves ajenas, tipos Java y tipos SQL.

Las herramientas ORM pueden ser implementadas de diversas maneras. Mark Fussel [Fus97] define los siguientes niveles de calidad para las herramientas de correspondencia objeto/relacional.

- Relacional pura (Pure relational). Toda la aplicación, incluido el interfaz de usuario, se diseña siguiendo el modelo relacional y las operaciones relacionales basadas en SQL. Como principal ventaja destaca que la manipulación directa de código SQL permite muchas optimizaciones. Sin embargo inconvenientes tales como la dificultad de mantenimiento y la falta de portabilidad son muy significativos. Las aplicaciones en esta categoría utilizan intensivamente procedimientos almacenados (stored procedures), desplazando parte de la carga de trabajo desde la capa de negocio hasta la base de datos.
- Correspondencia de objetos ligero (Light object mapping). Las entidades se representan como clases que se corresponden de forma manual sobre tablas relacionales. La codificación manual de SQL puede quedar separada de la lógica de negocio si se utilizan patrones de diseño como Data Access Object (DAO) [ACJ01]. Esta alternativa está ampliamente extendida para aplicaciones con un número pequeño de entidades, o aplicaciones con unos modelos de datos genéricos dirigidos por metadatos.

- Correspondencia de objetos medio (Medium object mapping). La aplicación se diseña en torno a un modelo de objetos. SQL se genera en tiempo de compilación utilizando una herramienta de generación de código o en tiempo de ejecución utilizando el código de un framework. El sistema de persistencia soportan las relaciones entre objetos y las consultas pueden especificarse utilizando un lenguaje de expresiones orientadas a objetos. Una gran parte de las soluciones ORM soportan este nivel de funcionalidad. Son adecuadas para aplicaciones de un tamaño medio, con algunas transacciones complejas, en particular cuando la portabilidad entre diferentes bases de datos es un factor importante.
- Correspondencia de objetos total (Full object mapping). Estas herramientas soportan características avanzadas del modelado de objetos: composición, herencia, polimorfismo y persistencia por alcance [AM95]. La capa de persistencia implementa persistencia transparente y las clases persistentes no necesitan heredar ninguna clase base especial ni implementar un determinado interfaz. Además se incluyen diversas optimizaciones que son transparentes para la aplicación, tales como diversas estrategias para cargar los objetos de forma transparente desde el almacén utilizado cuando se navega por un árbol de objetos, o la utilización de cachés de objetos.

Las herramientas de correspondencia objeto relacional presentan diversas ventajas que justifican su gran aceptación como sistema de persistencia en plataformas orientadas a objetos.

- Incrementan la productividad al eliminar evitar que sea el propio programador el que tenga que hacer la conversión de clases y objetos a tablas. El código necesario para esta traducción puede suponer entre un 25\% y un 40\% del total.
- Favorecen la mantenibilidad de la aplicación, al quedar separado el código que interactúa con la base de datos del código correspondiente a la lógica de negocio.
- Eliminan los errores derivados de la traducción objeto/relacional programática por parte del desarrollador.

- Permiten aprovechar las ventajas derivadas del dominio absoluto que tienen los SGBD relacionales en el mercado actual, en especial, del mayor rendimiento de los motores relacionales sobre los que implementan otros paradigmas. Además, favorecen la no dependencia de un fabricante concreto, puesto que el estándar SQL [MS93] es ampliamente soportado por los diferentes SGBD.

Por otra parte, la traducción automática entre ambos paradigmas también supone un coste debido a la desadaptación de impedancias [Mai89]: un coste computacional, puesto que se añade una nueva capa software adicional; y un coste de desarrollo, puesto que el desarrollador habitualmente deberá especificar mediante algún mecanismo los meta-datos que configuran cómo se realizará la traducción.

9.3.6.1. Hibernate

Hibernate, también conocida como H8, es una herramienta de correspondencia objeto/relacional para la plataforma Java. Se trata de un proyecto de código abierto distribuido bajo licencia LGPL que fue desarrollado por un grupo de programadores Java de diversos países liderados por Gavin King.

Hibernate proporciona un framework sencillo de utilizar para traducir un modelo dominio orientado a objetos a una base de datos relacional. Ofrece un mecanismo de persistencia cuya transparencia se refleja en su soporte para objetos POJO. El término POJO (Plain Old Java Object o también Plain Ordinary Java Object) fue acuñado en 2002 por Martin Fowler, Rebecca Parsons, and Josh Mackenzie. Se tratan de objetos que son esencialmente JavaBeans [Jav96] utilizados en la capa de negocio (muchos desarrolladores utilizan el término POJO y JavaBeans independientemente). Un objeto POJO declara métodos de negocio que definen comportamiento y propiedades que representan estado. Lo que busca el término es diferenciar este tipo de objetos de los Entity Beans [RSB05], cuya gestión es mucho más tediosa y son menos naturales desde el punto de vista del desarrollador.

Hibernate está pensado para trabajar con un modelo de dominio implementado mediante POJOs. A este modelo se le imponen una serie de requerimientos que intentan coincidir con las mejores prácticas de desarrollo con POJOs, de tal manera que se asegure la compatibilidad de la mayor parte de objetos POJO sin necesidad de realizar modificaciones. El modelo por lo tanto se trata de una mezcla de la especificación de JavaBeans, las mejores prácticas POJO y requerimientos específicos de Hibernate. Estos requerimientos son:

- Un constructor sin argumentos para toda clase persistente.
- Se recomiendan propiedades accesibles mediante métodos de acceso get() y set(), según la convención de nombres definida para los JavaBeans \cite{javabeans:1996} aunque este requisito no es obligatorio.
- Cuando se modelen asociaciones, deben utilizarse las Collections de Java para el atributo que modele la relación y además deben utilizarse para la referencia del correspondiente atributo interfaces no implementaciones concretas.

Para ilustrar el funcionamiento de Hibernate en el Código Fuente 9.2 se muestra cómo hacer persistir un objeto de dominio representado por la clase persona.

```
package example.hibernate;
public class Person{
   protected String name;
   public void setName(String name){
          this.name = name;
   }
public String getName(){
          return name;
   }
}
```

Código Fuente 9.2. Código de la clase persona.

Para especificar los metadatos que configuran cómo se realiza el correspondencia entre el modelo de objeto y el relacional Hibernate permite utilizar dos aproximaciones distintas.

En primer lugar pueden utilizarse un documento XML que especifica cómo se realiza la traducción. En el Código Fuente 9.3 se muestra un fichero XML que especifica cómo realizar la traducción para la clase Persona. En este fichero se especifica que la clase Person se almacenará en la tabla PERSON, que el identificador de cada objeto persona se almacenará en la columna PERSON_ID como clave primaria y será gestionado internamente por Hibernate (posteriormente se explicarán los diferentes mecanismos de gestión identidad en hibernate), y finalmente que la propiedad name de cada objeto persona se almacenará en la columna NAME de la tabla PERSON. Debe señalarse que, si bien muchos desarrolladores optan por escribir este fichero XML a mano, existen diversas herramientas que pueden generar este documento automáticamente, como por ejemplo Xdoclet [WR03].

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
   <class
          name="example.hibernate.Person"
          table="PERSON">
          <id column="PERSON_ID">
                 <generator class="native"/>
          </id>
          <property
                 name="name"
                 column="NAME"
                 type="string"/>
   </class>
</hibernate-mapping>
```

Código Fuente 9.3.Documento XML con los metadatos de mapeo para la clase Persona.

La segunda aproximación para especificar los metadatos de mapeo que soporta Hibernate es utilizar la denominada programación orientada a atributos, que fue introducida dentro de la plataforma Java de la mano de XDoclet [WR03]. Este sistema permite utilizar el formato de las etiquetas Javadoc (@attribute) para especificar atributos de metadatos a nivel de clases, atributos o métodos. En el Código Fuente 9.4 se muestra como puede anotarse el código fuente de la clase persona utilizando XDoclet, con el objeto de incluir la misma información que en el fichero recogido en Código Fuente 9.3.

```
package example.hibernate;
/**
* @hibernate.class
* table="PERSON"
*/
public class Person{
    protected String name;
    public void setName(String name){
            this.name = name;
    }

    /**
    * @hibernate.property
    */
    public String getName(){
            return name;
    }
}
```

Código Fuente 9.4. Código de la clase persona anotado con XDoclet.

Con el lanzamiento de la versión 1.5 de la versión estándar de la plataforma Java se introdujo un soporte para las introducir metadatos en el código fuente en forma de anotaciones. Se trata de un mecanismo mucho más sofisticado que el ofrecido por XDoclet. En el momento de escribir este documento el soporte por parte de Hibernate del sistema de anotaciones de la JDK 1.5 está siendo desarrollado.

Para almacenar un objeto en la base de datos, Hibernate utiliza un gestor de persistencia que viene definido por el interfaz Session. El código necesario para hacer persistir un objeto persona se muestra en el Código Fuente 9.5.

```
Session session = getSessionFactory().openSession();
Transaction tx = session.beginTransaction();
Person person = new Person();
session.save(person);
tx.commit();
session.close();
```

Código Fuente 9.5.Ejemplo de persistencia de un objeto con Hibernate.

El ciclo de vida de los objetos se modela mediante estados. Hibernate define únicamente tres estados, escondiendo la complejidad de su implementación interna al código del cliente. Estos estados son: transient, persistent and detached [Hib06] y el correspondiente diagrama de estados se ilustra con la Figura 9.2.

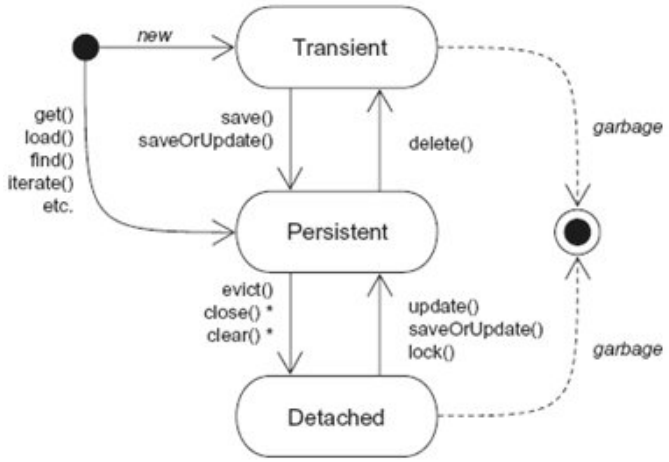


Figura 9.2.Ciclo de vida de un objeto persistente.

- **Transient (temporal)**. Un objeto es temporal si acaba de ser instanciado usando el operador new pero aún no ha sido asociado con una Session Hibernate. Este tipo de objetos no tiene una representación persistente ni tampoco un identificador asociado.
- **Persistent (persistente)**. Una instancia persistente tiene una representación en la base de datos y un identificador asociado. Está por definición dentro del ámbito de un objeto Session, bien sea porque haya sido guardado o recuperado de la base de datos. Hibernate detectará cualquier cambio que se realice sobre el estado del objeto y lo sincronizará con la base de datos cuando finalice la unidad de trabajo en curso.
- **Detached (separado)**. Una instancia "separada" es un objeto que ha sido persistente, pero cuya sesión asociada ha sido cerrada. La referencia al objeto todavía es válida y su estado podría ser modificado. Una instancia separada puede volver a ser unida con una nueva Sesión en un momento posterior, haciéndola a ella y a todas las modificaciones realizadas persistentes de nuevo.

Para recuperar objetos de la base de datos Hibernate proporciona los siguientes mecanismos [BK04]:

- Navegar por todo el grafo de objetos, una vez que un objeto ya ha sido cargado. El proceso de carga mientras se navega por el grafo es realizado por Hibernate de una manera transparente al usuario. Se ofrece además la posibilidad de configurar cómo se traen los objetos desde la base de datos a memoria permitiendo la configuración de distintas estrategias. Un ejemplo es la estrategia Lazy fetching mediante la cual sólo se consulta la base de datos para traer los objetos o colecciones de objetos a memoria la primera vez que son accedidos (a no ser que el objeto asociado haya sido cacheado).
- Recuperar los objetos a partir de su identificador, que es el método más eficiente cuando se conoce el identificador de un objeto.
- Utilizar el lenguaje HQL (Hibernate Query Language), el cual es un dialecto orientado a objetos de la familia de lenguajes de consulta relacionales SQL, si bien a diferencia de SQL no incluye características de definición ni manipulación de datos.
  Utilizar la API Hibernate Criteria, la cual proporciona a través del interfaz Criteria un modo seguro y orientado a objetos de ejecutar consultas sin necesidad de manipular cadenas de caracteres.
- Utilizar sentencias nativas SQL. En este caso Hibernate se encarga de traducir los resultados de las consultas JDBC (result sets) a grafos de objetos persistentes.

Con respecto a la actualización de los objetos persistentes, ésta se realiza de forma transparente al usuario, que únicamente tiene que codificar explícitamente la finalización de las unidades de trabajo abiertas. En el caso más sencillo, al confirmarse una transacción se actualizan en la base de datos todos los cambios que haya sufrido el estado del objeto en memoria. Respecto al borrado de objetos persistentes, debe invocarse explícitamente utilizando el objeto Session. Hibernate ofrece además multitud de operaciones que dan control al usuario sobre cómo y cuándo se realiza la actualización [Hib06].

Con respecto a la gestión de las identidades de los objetos Hibernate ofrece dos alternativas:

- Definir una propiedad "identificador" para cada clase persistente. En este caso el identificador de la propiedad siempre deberá ser id. Su valor será la clave primaria de la fila correspondiente en la tabla relacional. En este caso puede optarse por una gestión manual de la asignación de identificadores o dejar que Hibernate asigne automáticamente un valor a la propiedad id.
- Dejar que sea Hibernate quien gestione totalmente la generación y asignación de claves. En este caso el identificador asociado a una instancia puede obtenerse mediante el método Session.getIdentifier(Object object).

En ambos casos, si se opta por delegar la generación de identificadores a Hibernate, se permite la configuración de diferentes estrategias de generación de identificadores. La estrategia más utilizada es la denominada native (nativa) la cual selecciona automáticamente entre las estrategias restantes según el SGBD concreto que haya sido configurado.

Para la persistencia de las asociaciones entre objetos éstas serán codificadas siguiendo las convenciones habituales para objetos POJO: con atributos para las asociaciones uno a uno y objetos Collection para los casos de cardinalidad múltiple, proporcionando los correspondientes métodos de acceso get() y set() en ambos casos. La única restricción es que los atributos del tipo Collection se declaren utilizando un interfaz, no una implementación concreta. Será en la especificación de los metadatos de mapeo donde se defina la cardinalidad de la relación y si ésta es bidireccional o no para permitir a Hibernate la gestión de su persistencia.

Cuando se utilicen objetos del tipo Collection hay que prestar una atención especial a los métodos equals() y hashcode() definidos en la clase java.lang.Object, de la que heredan todos los objetos Java. Por ejemplo, una colección del tipo Set (conjunto) utilizará el método equals() de cada objeto que se introduzca para prevenir elementos duplicados. La implementación por defecto de equals() realiza una comparación de las identidades Java de los objetos comparados, es decir, compara el valor de las referencias. Esta implementación resulta válida siempre y cuando sólo se mantenga una sesión abierta, puesto que Hibernate garantiza que sólo habrá una única instancia por cada fila de la base de datos dentro de una misma sesión.

Si se manejasen instancias desde múltiples sesiones, sería posible tener un conjunto conteniendo dos objetos, cada uno de los cuales representan la misma fila de la tabla de la base de datos, pero que no tienen la misma identidad Java. Se trata por tanto de un caso semánticamente incorrecto que deberá ser solucionado sobrescribiendo el método equals() en las clases persistentes. Además, el método equals() debe de ser consistente con el método hashcode(), de tal manera que si dos objetos son iguales tengan el mismo código hash [Gosling96]. Por ello, si se redefine el método equals() el método hashcode() debe ser redefinido también de la manera apropiada. Una estrategia para la implementación de estos métodos es utilizar los identificadores de los objetos. En caso de equals() se compararían los identificadores de los objetos comparados y en caso de hashcode()se devolverían el hash del identificador del objeto. La limitación de esta estrategia es que es que debe proporcionarse una propiedad identificador a cada objeto del dominio.

Hibernate ofrece dos mecanismos de transitividad para la persistencia [BK04]

- **Persistencia por alcance**: Es un mecanismo ampliamente utilizado [AM95]. Su fundamento es que cualquier instancia se convierte en persistente en el mismo momento en el que la aplicación crea una referencia a dicha instancia desde otra instancia que ya es persistente.
- **Persistencia en cascada** (cascading persistence). Se trata de un mecanismo propio de Hibernate. Se basa en el mismo principio que la persistencia por alcance, pero permite especificar una estrategia de propagación en cascada para cada asociación, lo que ofrece una mayor flexibilidad y control para las transiciones entre los estados de persistencia.

Hibernate expone el metamodelo de los metadatos utilizados para configurar la correspondencia objeto/relacional con el fin de que éste pueda ser consultado y modificado en tiempo de ejecución [Hib06]. En Código Fuente 9.6 muestra cómo puede modificarse la configuración del mapeo utilizada para hacer persistente la clase Person, de tal manera que se añada una nueva propiedad age que se almacenará en la columna AGE de la tabla PERSON. Debe señalarse que, una vez que una SessionFactory ha sido creada, su configuración de mapeo es inmutable, por lo que para volver a obtener una nueva sesión con la nueva configuración es necesario crear una nueva fábrica. No obstante, una aplicación sí puede leer el metamodelo de la configuración utilizando el método getClassMetadata().

```
Configuration configuration = new Configuration();
// Obtiene el metamodelo de la configuración del mapeo objeto/relacional
PersistentClass personMapping=configuration.getClassMapping(Person.class);
// Define una nueva columna AGE para la tabla PERSON
Column column = new Column();
column.setType(Hibernate.INT);
column.setName("AGE");
column.setNullable(false);
column.setUnique(true);
personMapping.getTable().addColumn(column);
// Envuelve la columna en un objeto Value
SimpleValue value = new SimpleValue();
value.setTable( userMapping.getTable() );
value.addColumn(column);
value.setType(Hibernate.INT);

// Define la nueva propiedad para la clase Persona
Property prop = new Property();
prop.setValue(value);
prop.setName("age");
personMapping.addProperty(prop);

// Se construye una nueva SessionFactory utilizando la nueva configuración
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

Código Fuente 9.6.Ejemplo de manipulación del metamodelo de configuración Hibernate.

En cuanto a su funcionamiento interno, Hibernate no trabaja procesando el bytecode Java de las clases compiladas como hace JDO [Roo03]. En sus orígenes, utilizaba únicamente las posibilidades de introspección de la plataforma Java ofrecidas por la API Reflection [JCR97]. No obstante, para implementar la estrategia lazy fetching a la hora de navegar por el árbol de objetos persistentes esta API se mostraba insuficiente.

Para implementar la estrategia de lazy fetching Hibernate usa proxies [GHJ+95]. Un proxy es un objeto que implementa el interfaz público de otro objeto e intercepta todos los mensajes enviados a ese objeto por sus clientes. En el caso de Hibernate, esta interceptación es utilizada para cargar el estado del objeto la primera vez que es utilizado. En el caso de las asociaciones con cardinalidad múltiple en alguno de sus extremos Hibernate utiliza implementaciones propias de los interfaces Collection definidos en java.util. En el caso de asociaciones de cardinalidad simple, es decir, un objeto que hace referencia a una clase definida por el usuario, se necesita un mecanismo más sofisticado.

Para éste último caso, Hibernate no hace uso de las Java Dinamic Proxy Classes introducidas en la versión 1.3 de la plataforma. Este mecanismo permite crear en tiempo de ejecución un proxy que implemente una serie de interfaces. El problema de cara a su aplicación en Hibernate es que no sirve para construir un proxy si el objeto al que se accede no implementa ningún interfaz. Por ello, Hibernate apostó por utilizar CGLIB, un proyecto de código abierto que implementa un paquete de reflectividad alternativo al ofrecido por Java. En concreto, CGLIB permite crear proxies en tiempo de ejecución que hereden de una clase y que implementen interfaces, permitiendo de esta manera a Hibernate implementar la estrategia de lazy fetching de una manera completamente transparente.

Para actualizar un objeto en la base de datos cuando finalice una unidad de trabajo en curso y su estado haya cambiado es necesario utilizar un mecanismo que permita comprobar si el objeto está en un estado ``sucio'' o no. Hibernate utiliza para implementar esta funcionalidad una estrategia denominada "inspección" (inspection) [BK04]. Lo que hace es comprobar el valor de las propiedades de un objeto al finalizar una transacción con una copia de su estado guardada cuando fue cargado desde la base de datos.

Otra alternativa para implementar este mecanismo es la estrategia denominada "interceptación" (interception). En este caso la herramienta objeto/relacional intercepta todas las asignaciones de valores a los campos persistentes del objeto detectando así cuando se modifica su estado. Debido a las características de la máquina virtual de Java está estrategia sólo puede ser implementada interviniendo en tiempo de compilación o en tiempo de carga de clases. Esta es la estrategia que utilizan las implementaciones de JDO [Roo03].

9.3.6.2. Java Data Objects (JDO)

Java Data Objects (JDO) es una especificación [JDO03] de un sistema de persistencia de objetos para el lenguaje Java basado en interfaces. Esta especificación describe el almacenamiento, consulta y recuperación de objetos de almacenes de datos.

Las raices de JDO se encuentran en ODMG (Object Data Management Group). Este grupo definió un enlace o binding de su modelo de objetos [Cat96] para el lenguaje Java y los diversos fabricantes de bases de datos orientadas a objetos desarrollaron implementaciones para este binding. Este intento fue un fracaso, entre otras razones debido a que la total ausencia de estándares en el terreno de las bases de datos orientadas a objetos marcó el desarrollo del estándar ODMG, así como la no definición por parte del ODMG de un conjunto de pruebas de conformidad que asegurasen la interoperabilidad de las distintas implementaciones [Jor04]. Con la incorporación de la Java Community Process (JCP) se reemplazó este intento por la definición de una propuesta más ambiciosa: Java Data Objects.

La característica fundamental de JDO es que soporta la persistencia transparente, lo cual queda reflejado en los siguientes aspectos [Roo03]:

- JDO gestiona de una manera transparente el mapeo de instancias JDO al almacén de datos subyacente, es decir, la desadaptación de impedancias [Mai89]
- JDO es transparente a los objetos Java que se hacen persistentes. No es necesario que las clases persistentes hereden de una clase raíz, añadir a las clases nuevos métodos o atributos ni alterar los modificadores de visibilidad de los miembros de éstas. En concreto, se soportan atributos privados así como atributos que no presenten métodos de acceso get() y set().
- Con JDO se puede trabajar contra multitud de tipos de almacenes de datos que responden a distintos paradigmas. Entre otros: bases de datos relacionales, bases de datos orientadas a objetos, sistemas de archivos, documentos XML y datos almacenados en aplicaciones heredadas.

- JDO es transparente a su vez al almacén de datos utilizados, de manera que las aplicaciones pueden ser portadas para utilizar cualquier almacén de datos para el cual esté disponible una implementación JDO adecuada. La especificación JDO garantiza la compatibilidad binaria de las diferentes implementaciones, lo que significa que ésta característica puede conseguirse incluso sin necesidad de recompilación.
- Si una aplicación hace referencia a un objeto persistente y altera de cualquier forma su estado persistente en memoria, la implementación JDO se encarga implícitamente de actualizar el almacén de datos cuando la transacción sea confirmada. Así se elimina la necesidad de que el programador codifique reiteradamente operaciones explícitas de almacenamiento.

En comparación con otros estándares de la plataforma Java, JDO busca eliminar el código dedicado necesario cuando se utiliza JDBC y SQL, que nada tiene que ver con los requerimientos de negocio de una aplicación. Por otra parte trata de evitar la complejidad del modelo de persistencia de los Enterprise Java Beans (EJB) [Jav03].

Para utilizar JDO, en primer lugar se codifican las clases para las cuales se necesitan los servicios de persistencia, que habitualmente constituyen el modelo del dominio de objetos. Con el objetivo de mostrar cómo funciona JDO se va a desarrollar un sencillo ejemplo donde los objetos de dominio serán personas. El Código Fuente 9.7 recoge el código de la clase persona.

```
package example.jdo;
public class Person{
   protected String name;
   public void setName(String name){
          this.name = name;
   }
   public String getName(){
          return name;
   }
}
```

Código Fuente 9.7.Código de la clase Persona.

A continuación se escribe un documento XML denominado "descriptor de persistencia". Este documento XML, en su forma más sencilla, únicamente identifica los nombres de las clases persistentes. Se ofrece la posibilidad de especificar qué elementos se hacen persistentes con un mayor nivel de detalle, por ejemplo, especificar qué campos de la clase son persistentes. El descriptor de persistencia que hace persistente la clase persona utilizada en el ejemplo se muestra en el Código Fuente 9.8.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE jdo SYSTEM "file:///jdos/dtd/jdo.dtd">
<jdo>
<package name="example.jdo">
   <class name="Person" />
</package>
</jdo>
```

Código Fuente 9.8.Descriptor de persistencia.

Finalmente, debe lanzarse un proceso de "realce" (enhancement process) que se encarga de leer el descriptor XML y añadir dentro de cada clase a la que se haga referencia los métodos adicionales que son necesarios para el establecimiento y recuperación de atributos por parte de una implementación JDO. Estos métodos vienen definidos en el interfaz PersistenceCapable. Las diferentes implementaciones de JDO realizan este realce a nivel de bytecode Java [GJS04] aunque el estándar JDO [JDO03] no especifica ningún mecanismo. Si bien no se recomienda, también es posible implementar el interfaz PersistenceCapable manualmente, sin necesidad de tener que utilizar una herramienta de realce.

En la invocación del proceso pueden generarse scripts en el lenguaje de definición de datos correspondiente para la definición de la estructura de almacenamiento necesaria en un almacén de datos específicos. Algunos vendedores de implementaciones JDO proporcionan una herramienta de definición de esquemas específica con este propósito, habitualmente para bases de datos relacionales, mientras que este paso no suele ser requerido para bases de datos orientadas a objetos.

```
PersistenceManagerFactory persistenceManagerFactory = ... ;
PersistenceManager persistenceManager=
 persistenceManagerFactory.getPersistenceManager();
Transaction transaction = persistenceManager.getCurrentTransaction();
Person person = new Person();
person.setName("Pepe");

transaction.begin();
persistenceManager.makePersistent(person);
transaction.commit();
```

Código Fuente 9.9.Ejemplo de persistencia de un objeto con JDO.

Como es natural, no serán los objetos de dominio los que invoquen los servicios de persistencia, sino que serán los objetos de aplicación los que realicen esta tarea con el objetivo de hacer persistir y recuperar los objetos de dominio. El desarrollador codifica estas invocaciones utilizando un interfaz estándar JDO denominado PersistenceManager. En el Código Fuente 9.9 muestra un ejemplo de cómo se puede hacer persistente un objeto de la clase persona.

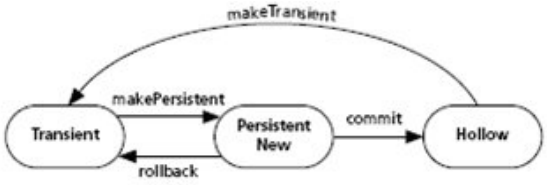


Figura 9.3.Estados correspondientes a la operación de hacer una instancia persistente.

El ciclo de vida de los objetos se modela mediante estados, los cuales son potencialmente visibles para la aplicación. Por ejemplo, el estado "hollow" (hueco) especifica que los campos de un objeto aún no han sido leídos del correspondiente almacén de datos. La transición entre estados se realiza la mayor parte de las veces explícitamente con invocaciones por parte del desarrollador. En la Figura 9.3 puede verse la transición de estados correspondientes a la operación de hacer una instancia persistente. También existen transiciones de estados implícitas, como por ejemplo cuando se modifica un campo de una instancia y esta pasa a marcarse como sucia, tal y como se muestra en la Figura 9.4. El borrado de objetos persistentes siempre será explícito haciendo uso de los servicios del objeto PersistenceManager.

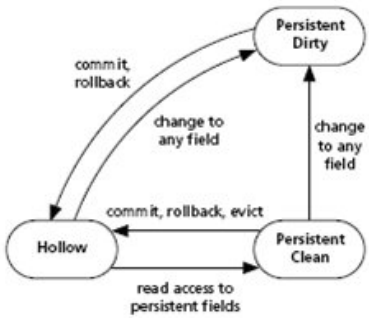


Figura 9.4.Estados correspondientes a la modificación de los campos de una instancia.

Con respecto a la recuperación de instancias persistentes del almacén de datos, JDO permite que las aplicaciones naveguen de una manera transparente por los atributos de un objeto persistente que referencian a otros objetos persistentes. Cuando la aplicación utiliza uno de estos objetos, aquellos que no están presentes en la caché del gestor de persistencia son cargados desde el almacén de datos. De esta manera se ofrece a la aplicación la impresión de que todo el grafo interconectado de instancias persistentes está disponible al instante en memoria. Así pues, el principal problema desde el punto de vista de la aplicación es cómo obtener la primera instancia persistente. JDO ofrece tres métodos [Roo03]:

Figura 9.5.Interfaz Query.

- Utilizar el identificador del objeto persistente para obtener una instancia de éste a través del gestor de persistencia. La aplicación debe por lo tanto ser capaz de construir el objeto identificador adecuado para realizar la consulta. Posteriormente se analizará como realiza JDO la identificación de objetos.
- Extensiones JDO. Se denomina extensión (extent) de una clase a todas las instancias persistentes de una clase o jerarquía de clases. Puede obtenerse la extensión de una clase a través del objeto PersistenceManager. y recorrerse las instancias persistentes haciendo uso de un iterador [GHJ+95]. Esta opción es recomendable cuando se desean procesar todas las instancias pero es muy ineficiente cuando reintenta recuperar una instancia específica o un grupo pequeño de ellas. En el Código Fuente 9.10 se muestra un ejemplo de cómo recuperar todas las instancias de la clase persona que han sido almacenadas imprimiendo su nombre en la pantalla.
- Sistema de consultas JDO [JDO03]. Este caso se trata de un sistema ofrecido por JDO para construir y ejecutar consultas mediante un mecanismo orientado a objetos e independiente del almacén de datos utilizado. Las consultas serán representadas mediante objetos que implementen el interfaz Query. Estos objetos consulta pueden construirse a través del gestor de persistencia y configurarse utilizando el interfaz definido. JDO define además el lenguaje JDOQL (JDO Query Language). Se trata de un sencillo lenguaje utilizado para poder personalizar determinados aspectos de las consultas tales como el orden de las mismas, la declaración de parámetros o los criterios de búsqueda, que serán representados a través de objetos filtro. Varias de las operaciones definidas en el interfaz Query (Figura 9.5) reciben como parámetro una cadena de caracteres que debe responder a este lenguaje. Su implementación corresponde al vendedor de la implementación JDO y siempre utilizarán los métodos de ejecución más eficientes disponibles en el almacén de datos concreto utilizado.

```
transaction.begin();
Extent personExtent =
   persistenceManager.getExtent(Person.class, false);
Iterator iterator = personExtent.iterator();

while (iterator.hasNext()) {
   System.out.println(iterator.next());
}

personExtent.close(iterator);
t.commit();
```

Código Fuente 9.10.Recuperación de las instancias persistentes.

Una vez que el descriptor de persistencia ha sido escrito, los objetos de dominio realzados, el medio de almacenamiento definido de la manera adecuada y se ha codificado la invocación al gestor de persistencia por parte de la aplicación, la aplicación está lista para ser ejecutada.

Debe hacerse hincapié en que el almacén de datos que utilice JDO no está limitado a un tipo concreto. JDO por sí mismo proporciona los interfaces mediante los cuales pueden invocarse los servicios de persistencia. Estos servicios serán invocados sobre una implementación JDO concreta, la cual será escogida no por su funcionalidad, que ya está especificada por JDO, sino precisamente por la calidad que ofrezca para trabajar contra un almacén de datos en concreta. Por ejemplo, para funcionar contra una base de datos relacional concreta, debería definirse el esquema de almacenamiento en esa base de datos y utilizar una implementación JDO adecuada. Del mismo modo debería hacerse para funcionar contra una base de datos orientada a objetos utilizando una implementación JDO alternativa. Lo más importante es que, desde el punto de vista del desarrollador, no se requiere trabajo para migrar una aplicación que utilice JDO desde un almacén de datos a otro, ni siquiera cuando cambie el paradigma de almacenamiento.

Con el fin de mejorar la transparencia con la cual JDO puede ser aplicado a los modelos de objetos existentes, JDO define su propio concepto de identidad de objetos, que es independiente de los conceptos de igualdad y equivalencia del lenguaje Java. En JDO existen tres tipos de mecanismos para establecer la identidad de los objetos [Roo03]:

- **Identidad del almacén de datos (datastore identity)**. Se trata del mecanismo de identidad por defecto. Se asocia una identidad al objeto en el momento en el que éste se hace persistente. La manera de asignar un objeto identidad y la naturaleza depende de cada implementación JDO y del almacén de datos. Una vez que la identidad ha sido fijada puede ser utilizada en futuras peticiones para recuperar un objeto en particular.
- **Identidad de aplicación (application dentity)**. En este caso es la aplicación la responsable de las identidades de las instancias, las cuales serán derivadas de los valores de un subconjunto de sus atributos persistentes. En el descriptor de persistencia se especifican uno o más campos persistentes que conformarán una clave primaria para la instancia, así como el nombre de la clase que representará el objeto identidad a establecer. Esta clase normalmente es programada por el desarrollador de la aplicación, aunque determinadas herramientas de realce son capaces de generarlas cuando realzan las clases que utilizan este mecanismo de identidad.
- **Identidad no duradera (non-durable identity)**. Se utiliza para objetos persistentes donde no tiene sentido distinguir unos de otros. Se utiliza por razones de eficiencia, para permitir la persistencia rápida de nuevas instancias.

El mecanismo de asignación identidad deseado para cada instancia se especifica en el descriptor de persistencia. Internamente, la implementación de JDO es responsable de asegurar que existe, como mucho, una instancia persistente JDO asociada con un objeto específico del almacén de datos por cada gestor de persistencia. Este proceso es denominado en inglés uniquing.

JDO divide las instancias en objetos de primera clase y objetos de segunda clase [JDO03]. Esta distinción es muy importante para el desarrollador, puesto que la manera de trabajar con instancias compartidas es muy diferente dependiendo de la clase de la instancia.

Los objetos de primera clase son instancias de una clase PersistenceCapable que tienen una identidad JDO y por lo tanto soportan la unicidad (uniquing) en la caché del PersistenceManager.

Cuando se modifican los valores de los atributos de un objeto de primera clase, el estado de ese objeto se marca como "sucio", estado en el que permanecerá hasta que se confirme la transacción a realizar (commit) o se cancele (rollback).

Un objeto de segunda clase es una instancia de una clase PersistenceCapable, o, en el caso de las clases de la API estándar de Java, una instancia de una clase que no implementa el interfaz PersistenceCapable pero para la cual la implementación JDO proporciona un soporte específico de persistencia. Se diferencian de los objetos de primera clase porque no tienen una identidad JDO y por lo tanto no soportan el proceso de unicidad (uniquing). Únicamente serán almacenados como parte de un objeto de primera clase.

Cuando se modifica un objeto de segunda clase éste no asume por sí mismo un estado sucio. En lugar de eso, transmite el hecho de que ha cambiado a su objeto de primera clase poseedor, el cual sí se marca como sucio.

La principal razón de ser de los objetos de segunda clase es permitir diferenciar los artefactos de programación que no tienen representación en el almacén de datos pero que son utilizados únicamente pare representar relaciones. Por ejemplo, una Collection de objetos PersistenteCapable podría no almacenarse en el almacén de datos, sino que sería utilizada para representar la relación en memoria. A la hora de efectuarse la transacción y hacer persistente el objeto, el artefacto en memoria se descarta y la relación se representa únicamente con las relaciones del almacén de datos.

Aunque la especificación de JDO no utiliza este término, en [Roo03] se denominan a los arrays Java objetos de tercera clase. Esto es debido a que su soporte por parte de las implementaciones JDO es opcional. Desde el punto de vista práctico, el realce a nivel de bytecode de los arrays no es válido puesto que los arrays en la plataforma Java son objetos especiales que no tienen un fichero ".class" asociado. Lo que se recomienda en este caso es implementar la funcionalidad mediante la cual los cambios en los contenidos del array se reflejen marcando como sucio el objeto de primera clase contenedor del array.

JDO ofrece un sistema de persistencia por alcance [AM95] a través del concepto de cierre transitivo de un objeto. Todos los objetos referenciados, directa o indirectamente, desde un objeto persistente son hechos persistentes. Esto significa que, aunque un objeto no sea marcado específicamente como persistente en el descriptor de persistencia, si forma parte del cierre de objetos referenciado a través de los atributos persistentes de otra instancia persistente, entonces el objeto es a su vez persistente.

JDO ofrece además la ilusión de que la aplicación tiene acceso en memoria al cierre transitivo de todas las instancias persistentes referenciadas desde cualquier instancia persistente, independientemente del hecho que la mayor parte de este conjunto de instancias, potencialmente enorme, estará normalmente en disco y no en realmente en memoria.

9.4. FRAMEWORKS DE PERSISTENCIA

Un framework puede definirse como una colaboración de clases adaptables que definen una solución para un problema dado. Un framework define una colección de abstracciones fundamentales junto con sus interfaces y especifica las interacciones entre los objetos, con el fin de poder reutilizar la arquitectura y el diseño. Además, suele proporcionar una serie de implementaciones por defecto para favorecer la reutilización de código.

En este apartado se analizarán dos frameworks de persistencia que utilizan aproximaciones muy diferentes: Enterprise Java Beans (EJB) [Jav03] y Spring [SPR].

9.4.1. Enterprise Java Beans

Enterprise Java Beans (EJB) [Jav03] define una arquitectura de componentes para el desarrollo de aplicaciones distribuidas. EJB forma parte de la especificación de la plataforma Java 2 Enterprise Edition (J2EE). Los desarrolladores que hagan uso de esta arquitectura, obtienen un soporte transparente para los aspectos de distribución, persistencia, transacciones y seguridad [Jor04].

Un enterprise bean es un componente software que se ejecuta en el lado del servidor. Necesitan un servidor de aplicaciones J2EE que disponga de un contenedor (container) que debe implementar la especificación EJB Los enterprise beans se despliegan sobre ese contenedor que se será el gestor del ciclo del ciclo de vida del bean y se interpondrá en todos los accesos al mismo.

Existen tres tipos diferentes de enterprise beans RSB05]:

- Session beans. Modelan los procesos de negocio. En el caso más típico representan un único tipo de cliente dentro del servidor de aplicaciones. Para acceder a una aplicación que ha sido desplegada en el servidor, el cliente invoca a los métodos del session bean y éste ejecuta las tareas de negocio dentro del servidor, ocultando su complejidad al cliente.
- Entity beans. Representan un objeto de negocio dentro de un mecanismo de almacenamiento persistente. El sistema de almacenamiento típico es una base de datos relacional pero la arquitectura permite otros mecanismos tales como bases de datos orientadas a objetos con el requisito de que el fabricante del servidor de aplicaciones los soporte.
- Message-driven beans. Son enterprise beans que permiten a una aplicación J2EE procesar mensajes de una manera asíncrona. Actúan como escuchadores de los mensajes que son enviados a cualquier componente J2EE, permitiendo procesar dichos mensajes y realizar acciones cuando se intercepten.

Desde el punto de vista de este trabajo de investigación, los componentes que mayor interés representan de cara a la persistencia son los entity beans. Centrándonos en ellos, puede hablarse de dos mecanismos de gestión de persistencia dependiendo de quién se encargue de la interacción con el almacén de datos:

- Bean-managed persistence (Persistencia gestionada por el bean). En este caso son los propios entity beans los que realizan las operaciones necesarias para hacerse persistentes. En otras palabras, los desarrolladores son los responsables de escribir el código necesario para transportar los objetos en memoria al almacén de datos subyacente. Esto incluye las operaciones de guardar, cargar y encontrar datos dentro del componente. Debe utilizarse por lo tanto una API de persistencia, típicamente JDBC.
- Container-managed persistence (Persistencia gestionada por el contenedor). En este caso es el contenedor J2EE el que se encarga de gestionar todos los accesos a base de datos que requiere el entity bean. El código del bean no está ligado a ningún mecanismo específico de almacenamiento persistente, por lo que se podría reutilizar el bean en otro servidor de aplicaciones distinto que utilizase otros sistemas de almacenamiento sin necesidad de recompilar el código del bean. La persistencia gestionada por el contenedor puede ser implementada a su vez utilizando algún mecanismo de persistencia, como JDO o Hibernate.

Para desarrollar un Entity Bean (y también un Session Bean) deben crearse los siguientes ficheros [Jav03]:

- Un descriptor de despliegue (Deployment descriptor). Es un fichero XML que especifica información acerca del bean, por ejemplo el tipo de persistencia y las características de los mecanismos de transacción.
- La clase del Enterprise bean: Implementa los métodos definidos en los interfaces descritos a continuación.
- Interfaces: Los clientes accederán a los beans únicamente a través de los métodos definidos en sus interfaces. EJB distingue entre el acceso local y remoto diferenciando dos tipos de interfaces:

- Para permitir un acceso remoto al bean deben proporcionarse un interfaz remoto y otro home. El interfaz remoto define los métodos de negocio específicos del componente y el interfaz home define los métodos relacionados con el ciclo de vida del componente: create() y remove(). En el caso específico de los entity beans el interfaz home también define métodos de búsqueda y métodos home. Los primeros se utilizan para localizar los entity beans. Los segundos son métodos de negocio que son invocados en todas las instancias de la clase del entity bean.
- Para un acceso local, deben proporcionarse un interfaz local y otro local home. El interfaz local define los métodos de negocio del componente y el interfaz local home define los métodos relacionados con su ciclo de vida y métodos de búsqueda.

Cuando se utiliza el sistema de persistencia gestionada por el contenedor éste se encarga también de realizar la gestión de las relaciones entre objetos. Para ello, se declara en el descriptor de persistencia información acerca de las relaciones que deben ser gestionadas. Con esta información el contenedor se encarga de generar todo el código que gestiona las relaciones dentro de una nueva clase que hereda del entity bean correspondiente. Esta es una de las razones por las cuales el código de los entity beans difícilmente puede ser utilizado fuera de los contenedores J2EE.

Con respecto a la gestión de las claves primarias, también cambia dependiendo del escenario arquitectónico que se usen los EJB. En el caso de la persistencia gestionada por el propio bean, el desarrollador debe codificar cómo se generan las claves. Cuando la persistencia la gestiona el contenedor J2EE, puede optarse por una generación automática de claves. En ambos casos, pueden utilizarse clases definidas por el usuario que representen los objetos clave si bien estas clases deberán cumplir unos determinados requisitos, que serán más estrictos en el caso de la persistencia gestionada por el contenedor.

EJB define además un lenguaje de consulta denominado EJBQL (EJB Query Language), el cual está basado en una mezcla de SQL y OQL. EJBQL puede ser utilizado para implementar los métodos de búsqueda, que son usados por clientes externos del entity bean, así como por métodos de selección, utilizados habitualmente por los entity beans.

9.4.2. Spring

El framework Spring [SPR] se define como una solución ligera para el desarrollo rápido de aplicaciones J2SE y J2EE. El framework se presenta con una arquitectura modular con el objeto de que el desarrollador pueda utilizar módulos concretos de funcionalidad sin tener que preocuparse del resto. Esta arquitectura se muestra en la Figura 9.6.

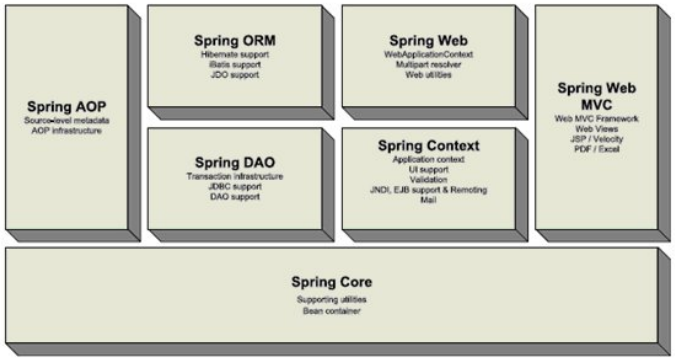


Figura 9.6.Arquitectura de Spring.

Desde el punto de vista de este documento, los módulos que mayor interés presentan de cara a la persistencia son los módulos Spring DAO y Spring ORM (Object/Relational mappers). Antes de estudiar estos módulos es necesario explicar los fundamentos de funcionamiento de Spring, los cuales vienen explicados por el patrón Dependency Injection (Inyección de dependencia), también conocido como Inversion of Control (Inversión de control, IoC).

El módulo Spring Core representa el corazón del framework e implementa las funcionalidades del patrón Dependency Injection, el cual es utilizado intensivamente por todos los módulos de Spring para la gestión y configuración de recursos. El componente básico es el interfaz BeanFactory, el cual aplica el patrón factory method [GHJ+05] para definir una fábrica de Java Beans [Jav96]. Esta fábrica elimina la necesidad crear y configurar programáticamente los Java Beans, típicamente utilizando fábricas singleton [GHJ+05]. Permite desacoplar la configuración y especificación de las dependencias de la lógica del programa. La manera de especificar cómo se crean y configuran los objetos puede ser potencialmente cualquiera, mediante diferentes implementaciones del interfaz BeanFactory. El mecanismo más utilizado es la configuración mediante ficheros XML utilizando la implementación dada por la clase XMLBeanFactory. En la nomenclatura de Spring, tanto el interfaz BeanFactory como el interfaz ApplicationContext, que extiende al primero y añade nuevas funcionalidades, se denominan contenedores IoC (Inversion of control).

El funcionamiento de la Inversión de control se ilustrará con un ejemplo. El java bean a configurar vendrá dado en la clase Person que se muestra en el Código Fuente 9.11.

```
package example.spring;
public class Person{
   protected String name;
   protected Person spouse;

   public void setName(String name){
          this.name = name;
   }

   public String getName(){
          return name;
   }

   public void setSpouse(Person spouse){
          this. spouse = spouse;
   }

   public Person getSpouse(){
          return spouse;
   }

}
```

Código Fuente 9.11.Código de la clase persona.

En el Código Fuente 9.12 se muestra el fichero XML necesario para configurar un objeto persona de nombre Pepe cuyo cónyuge es otra nueva persona de nombre María. El atributo singleton permite configurar si existirá una única instancia del objeto para un identificador dado o si por el contrario cada vez que se solicite el objeto se creará una nueva instancia.

```
<bean id="pepe" class="example.spring.Person" singleton="false">
  <property name="name"><value>Pepe</value></property>
    <property name="spouse">
      <bean class="org.springframework.beans.TestBean">
        <property name="name"><value>Maria</value></property>
   </bean>
    </property>
</bean>
```

Código Fuente 9.12.XML de configuración del objeto persona.

Desde el código de la aplicación sería necesario instanciar un objeto XMLBeanFactory y configurarlo a partir del fichero XML creado, se muestra en el Código Fuente 9.13.

```
XmlBeanFactory factory =new XmlBeanFactory(new FileInputStream("beans.xml"));
Person pepe = (Person)factory.getBean("pepe", example.spring.Person.class);
```

Código Fuente 9.13.Utilización de un BeanFactory para obtener un Bean.

El modulo Spring DAO (Data Access Object) facilita la utilización del patrón DAO [ACJ01]. Este patrón propone utilizar un objeto, denominado Data Access Object, que medie entre el cliente que accede a unos datos y el sistema de almacenamiento que contiene los datos, de tal manera que los mecanismos necesarios para interactuar con el almacén queden ocultos de cara al cliente. Esto permite poder cambiar el sistema de almacenamiento sin necesidad de modificar el código de la aplicación cliente, puesto que el código de acceso al almacén de datos y el código de la aplicación son independientes. En la Figura 9.7 se muestra el diagrama de clases de este patrón en su versión más general. Además del objeto DAO, el objeto Data se corresponde con otro patrón J2EE, el patrón Transfer Object, utilizado para transportar datos puros [ACJ01]. Su uso en este patrón es opcional.

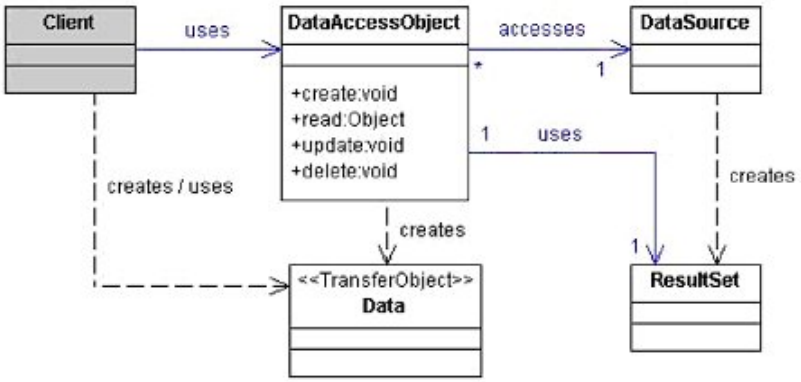


Figura 9.7.Diagrama de clases del patrón Data Access Object.

El soporte DAO en Spring se enfoca principalmente a facilitar el trabajo con diferentes tecnologías como JDBC, Hibernate o JDO haciendo que éste se realice de una manera estándar y uniforme, con el objetivo de poder cambiar la tecnología escogida de una manera sencilla. Para ello ofrece una jerarquía común de excepciones y un mapeo automático de las excepciones específicas de cada tecnología con las excepciones comunes de Spring.

Spring proporciona un conjunto de clases abstractas DAO que pueden ser extendidas por el desarrollador. Estas clases abstractas proporcionan métodos para establecer la fuente de datos (data source) y otros aspectos de configuración que son específicos de la tecnología utilizada. Estas clases se denominan JdbcDaoSupport, HibernateDaoSupport y JdoDaoSupport dependiendo de la tecnología a la que den soporte. Están pensadas para ser extendidas facilitando la creación de objetos DAO. Para ello ofrecen acceso a un objeto denominado respectivamente JdbcTemplate, HibernateTemplate o JdoTemplate. Este objeto permite ejecutar acciones específicas con la tecnología subyacente encargándose de realizar gran parte de las tareas repetitivas que son necesarias para ejecutar la acción, como por ejemplo, la necesidad de adquirir y liberar recursos. El enfoque utilizado para permitir la configuración de acciones específicas y su ejecución en un contexto determinado es utilizar el patrón Command [GHJ+95] para suministrar las acciones que serán ejecutadas vía callback en el contexto de la plantilla. Se trata por tanto del patrón Template method [GHJ+95] siendo el método a ejecutar modelado como un objeto comando.

Para el caso de JDBC, el soporte al patrón DAO lo proporciona principalmente la clase JdbcTemplate. Su función es simplificar el uso de JDBC encargándose de gestionar la creación y liberación de recursos. Ayuda a evitar errores comunes como por ejemplo olvidarse de cerrar la conexión. Se encarga de ejecutar las tareas comunes de JDBC como la creación de sentencias y su ejecución, dejando que sea el código de la aplicación el encargado de proporcionar el SQL necesario y extraer los resultados. Además se encarga de capturar las excepciones JDBC y las traduce a la jerarquía genérica de excepciones DAO de Spring. En el Código Fuente 9.14 puede observarse la utilización de JdbcTemplate para ejecutar una sentencia SQL. La creación de un objeto con la propiedad dataSource permite configurar dicha propiedad mediante inversión de control.

```
public class ExecuteAStatement {
  private JdbcTemplate jdbcTemplate;

   private DataSource dataSource;

   public void doExecute() {
     jdbcTemplate = new JdbcTemplate(dataSource);
     jdbcTemplate.execute
       ("create table mytable(id integer,name varchar(100))");
   }
   public void setDataSource(DataSource dataSource) {
     this.dataSource = dataSource;
   }
}
```

Código Fuente 9.14.Ejecución de una sentencia SQL utilizando el soporte de Spring.

Además, se ofrece un conjunto de objetos que permiten modelar las consultas, actualizaciones y procedimientos almacenados como objetos independientes. Spring implementa esta funcionalidad utilizando JDO [Roo03], si bien los objetos devueltos no están obviamente ligados a la base de datos.

El soporte para la integración de herramientas de mapeo objeto/relacional viene implementado en el módulo Spring ORM. Spring puede integrarse con Hibernate, JDO, Oracle TopLink, Apache OJB e iBATIS SQL Maps. Esta integración se produce en términos de gestión de recursos, soporte para la implementación del patrón DAO y estrategias de transacciones. Además es consistente con las jerarquías genéricas de excepciones DAO y transacciones de Spring.

Existen dos enfoques para esta integración: extender las clases DAO de Spring o codificar los DAOs utilizando la API de la tecnología correspondiente. En ambos casos, los DAO se configurarán utilizando la Inyección de Dependencias (dependency injection).

Con respecto a la gestión de recursos, Spring permite gestionar mediante inversión de control la creación y configuración de los objetos necesarios para utilizar las herramientas de mapeo objeto/relacional soportadas.

Por ejemplo, en el Código Fuente 9.15 se muestra cómo puede configurarse un DataSource JDBC y utilizarlo como una de las propiedades con las que se configura un SessionFactory Hibernate. De este modo, la fábrica de sesiones Hibernate podría estar disponible desde cualquier objeto de la aplicación, incluso pudiendo obtenerse de una manera declarativa como parte de una propiedad de un java bean que se configure mediante XML.

```
<beans>
  <bean id="myDataSource"
   class="org.apache.commons.dbcp.BasicDataSource"
   destroy-method="close">
    <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
    <property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/>
    <property name="username" value="sa"/>
    <property name="password" value=""/>
  </bean>

  <bean id="mySessionFactory"
  class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"/>

    <property name="mappingResources">
      <list>
        <value>product.hbm.xml</value>
      </list>
    </property>

    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">
          net.sf.hibernate.dialect.MySQLDialect
        </prop>
      </props>
    </property>
  </bean>
...
</beans>
```

Código Fuente 9.15.Configuración de los recursos Hibernate desde Spring.

La clase de HibernateTemplate se encarga de asegurar que las sesiones hibernate son abiertas y cerradas adecuadamente, además de interactuar automáticamente con el sistema de transacciones. Las acciones a ejecutar se crean implementando el interfaz HibernateCallback. Spring proporciona además la clase base HibernateDaoSupport que define la propiedad sessionFactory, que podría ser configurada por inversión de control, y el método getHibernateTemplate() para que sea utilizado por las subclases. De este modo, se simplifica la creación de objetos DAOs para los requerimientos más habituales. El soporte a JDO en Spring es muy similar al de Hibernate.

Una de las abstracciones más importantes del framework Spring es la gestión de transacciones. Entre las características que ofrece destacan [SPR] las siguientes:

- Proporciona un modelo de programación consistente transversal a diferentes APIs de transacciones como las proporcionadas por JTA, JDB, Hibernate y JDO.
- Proporciona una API para la gestión programática de transacciones más sencilla de utilizar que la mayor parte de las APIs anteriores.
- Se integra con la abstracción de acceso de a datos de Spring.
- Soporta la gestión de transacciones declarativa.

El elemento clave del sistema de transacciones de Spring es la "estrategia de transacción". Esta estrategia se representa con el interfaz PlatformTransactionManager. (Código Fuente 9.16)

```
public interface PlatformTransactionManager{
   TransactionStatus getTransaction(TransactionDefinition definition)
     throws TransactionException;
  void commit(TransactionStatus status) throws TransactionException;
  void rollback(TransactionStatus status) throws TransactionException;
}
```

Código Fuente 9.16.Interfaz PlatformTransactionManager.

Las diferentes implementaciones de este interfaz se definirán como cualquier otro objeto dentro de un contenedor IoC. En el Código Fuente 9.17 se muestra el código XML necesario para crear un gestor de transacciones Hibernate.

```
<bean id="txManager"
class="org.springframework.orm.hibernate.HibernateTransactionManager">

    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

Código Fuente 9.17.Creación de un gestor de transacciones Hibernate.

Para la gestión programática de transacciones, se proporciona el objeto TransactionTemplate. Este objeto funciona de forma similar a otros objetos template como HibernateTemplate. Utiliza un enfoque de callback para evitar a la aplicación la necesidad de adquirir y liberar recursos. Para poder utilizar un objeto TransactionTemplate éste debe ser configurado con un objeto PlatformTransactionManager. Esta configuración se realizará normalmente mediante inversión de control. Es importante señalar que, si se desease cambiar la tecnología de acceso a datos utilizada, únicamente habría que modificar el fichero XML donde se configura el gestor de transacciones específico (Código Fuente 9.18).

```
Object result = transactionTemplate.execute(new TransactionCallback(){
           public Object doInTransaction(TransactionStatus status) {
             operation1();
             return operation2();
            }
});
```

Código Fuente 9.18.Gestión programática de transacciones.

Spring ofrece además la posibilidad de realizar una gestión de transacciones declarativa. Para ello hace uso del módulo Spring AOP (Aspect Oriented Programming), el cual permite utilizar características de programación orientada a aspectos para extender la funcionalidad del framework.

La principal ventaja de la gestión de transacciones declarativa es que reduce al mínimo el impacto de ésta en el código de la aplicación. Otras características destacables son:

- La gestión declarativa de transacciones funciona con cualquiera de los entornos de persistencia soportados.
- El sistema puede aplicarse a cualquier objeto POJO (Plain Old Java Object).

- Pueden configurarse reglas de rollback de una forma declarativa. Estas reglas permiten especificar qué excepciones deberían provocar un rollback de una transacción al ser lanzadas. Esta especificación se realiza de forma declarativa. Esto evita que los objetos de negocio dependan de la infraestructura de transacciones. En concreto, no necesitan importar ninguna API de Spring ni de transacciones.
- Spring ofrece la oportunidad de personalizar el comportamiento transaccional utilizando programación orientada a aspectos. Por ejemplo, puede insertarse un determinado comportamiento a ejecutarse cuando se produzca un rollback de una transacción.

La manera habitual de trabajar con transacciones declarativas es utilizar la clase TransactionProxyFactoryBean. El módulo de AOP de Spring es implementado haciendo uso de proxies generados dinámicamente (si bien acepta su integración con otros sistemas AOP más sofisticados, como AspectJ). Para ello hace uso de las Java Dinamic Proxy Classes para crear proxies para clases que implementan interfaces y de GCLIB cuando no hay interfaces implementados en la clase para la cual se genera el proxy. TransactionProxyFactoryBean es únicamente una versión especializada de la clase genérica ProxyFactoryBean que, además de crear un proxy para envolver el objeto correspondiente (target object), le añade un objeto TransactionInterceptor. Este objeto contiene la información relativa a qué métodos y propiedades requieren un tratamiento transaccional y cómo se configura ese tratamiento.

En el Código Fuente 9.19 se muestra un ejemplo de la gestión declarativa de transacciones con Spring. En él se crea un proxy que envolverá al objeto Persona de identificador pepe cuya configuración se muestra en el Código Fuente 9.2. Este proxy se configura para hacer uso del gestor de transacciones con el identificador txManager Código Fuente 9.17.

Finalmente se proporcionan los atributos que configurarán el comportamiento transaccional. Los atributos se basan en la nomenclatura utilizada por los contenedores EJB. En el caso del atributo PROPAGATION_REQUIRED, declara que la transacción utilizada para ejecutar el método asociado podrá ser una de las transacciones en curso o, de no existir ninguna, deberá crearse una nueva. En el ejemplo, esta propiedad se ha asociado a los métodos del tipo set(). Al resto de métodos del objeto se les asocia además la propiedad de "transacción de sólo lectura" mediante el atributo readOnly.

```
<bean id="personStore"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
   <property name="transactionManager" ref="txManager"/>

   <property name="target" ref="pepe"/>

   <property name="transactionAttributes">
          <props>
                 <prop key="set*">PROPAGATION REQUIRED </prop>
                 <prop key="*">PROPAGATION REQUIRED,readOnly</prop>
          </props>
   </property>
</bean>
```

Código Fuente 9.19.Gestión de transacciones declarativa con Spring.

Debe señalarse que el sistema de gestión de transacciones de Spring no necesita un servidor de aplicaciones para funcionar como el caso de los Enterprise Java Beans con las transacciones gestionadas por el contenedor (Container managed transactions, CMT) [RSB05].

10. GENERACIÓN AUTOMÁTICA DE INTERFACES GRÁFICAS DE USUARIO

El objetivo de este capítulo es mostrar como abordar la generación automática de interfaces de usuario a partir de modelos conceptuales. Solo con modelos claros y precisos, es posible diseñar e implementar aplicaciones con interfaces que tengan calidad.

Para conseguir este objetivo primero se revisarán dos conceptos fundamentales en la creación de interfaces cómo son la accesibilidad y usabilidad. Posteriormente se estudiará el Modelo Vista Controlador (MVC) que es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

10.1. ACCESIBILIDAD

Los seres humanos son diferentes entre si y todas las interfaces de usuario deberían acomodarse a esas diferencias de forma que cualquier persona sea capaz de utilizarlas sin problemas.

Hay que evitar diseñar atendiendo a características de grupos de población específicos, imponiendo barreras innecesarias que podrían ser evitadas prestando más atención a las limitaciones de éstos. Cuando una diferencia individual supera un límite más o menos arbitrario recibe la etiqueta de discapacidad.

Lo que caracteriza a muchas de estas discapacidades está presente en mayor o menor grado entre muchas personas consideradas ``normales'', por lo que tener en cuenta las recomendaciones pertinentes no sólo es importante para aquellos con limitaciones mayores. [PC06b]

Las diferencias individuales en un grupo de aproximadamente 30 personas puede llegar a ser a menudo de un factor de 20 [Ega88]. Estas diferencias son mucho más importantes si los usuarios son "discapacitados".

Todos los desarrolladores pretenden conseguir el mayor número de usuarios para sus productos, pero no todos están dispuestos a realizar los esfuerzos necesarios para lograrlo. Existe la idea de que el volumen de población no es lo suficientemente importante. Microsoft en el año 2000 estimó que uno de cada cinco estadounidenses tiene algún tipo de discapacidad. Además empresas de gran tamaño e instituciones contratan como parte de su política de personal a un grupo fijo de individuos calificados legalmente como discapacitados. Muchos gobiernos han incluido reglamentos y leyes que especifican requisitos que deben cumplir los productos utilizados en la administración pública y en las organizaciones que dependen de ella.

Las barreras que suponen las discapacidades que restringen la movilidad de las personas pueden verse paliadas con Internet siempre que los productos software puedan emplearse fácilmente. [PC05]

10.1.1. Principios del Diseño Universal

El Diseño Universal [UD97] es el diseño de productos y entornos de fácil uso para el mayor número de personas posible, sin la necesidad de adaptarlos o rediseñarlos de una forma especial.

El propósito del diseño universal es simplificar la realización de las tareas cotidianas mediante la construcción de productos, servicios y entornos más sencillos de usar por todas las personas y sin esfuerzo alguno. El diseño universal, así pues, beneficia a todas las personas de todas las edades y habilidades.

El diseño universal se base en 7 principios:

- **Uso equitativo.** El diseño ha de ser usable y con un precio razonable para personas con diferentes habilidades
- **Uso flexible.** El diseño se ha de acomodar a un rango amplio de personas con distintos gustos y habilidades.
- **Uso simple e intuitivo.** El diseño ha de ser fácil de entender, independiente de la experiencia del usuario, conocimiento, nivel de concentración,...
- **Información perceptible.** El diseño comunica la información necesaria de manera efectiva al usuario, independientemente de las condiciones ambientales para las habilidades sensoriales del usuario.
- **Tolerancia para el error.** El diseño minimiza posibles incidentes por azar y las consecuencias adversas de acciones no previstas.
- **Esfuerzo físico mínimo.** El diseño se ha de poder usar eficientemente y confortablemente con un mínimo de fatiga.
- **Tamaño y espacio para poder aproximarse y usar el diseño.** El diseño ha de tener un espacio y un tamaño apropiado para la aproximación, alcance y uso del diseño.

10.2. USABILIDAD

La usabilidad se puede considerar como la capacidad del producto software para permitir que usuarios específicos logren realizar tareas específicas con productividad, efectividad, seguridad y satisfacción en determinados casos de uso. El objetivo de un producto es que posea la calidad necesaria y suficiente para que satisfaga las necesidades de usuario explícitas e implícitas.

Sin embargo, la Web es diferente de un producto software, y la naturaleza de la Web plantea nuevos desafíos a los diseñadores que están intentando incorporar usabilidad en sus aplicaciones Web.

- En el diseño de productos y software, los usuarios pagan de antemano y experimentan la usabilidad después.
- En la Web, los usuarios experimentan primero la usabilidad y pagan después.
- Las expectativas infladas del usuario de la tecnología del Internet pueden ser difíciles de satisfacer.
- Un sitio Web puede atender a multiplicidad de perfiles de usuario y sus distintos requerimientos. El construir un sitio orientado a audiencias no siempre es fácil
- En la Web el usuario no ha hecho una inversión en un sitio particular, y otras opciones son fácilmente disponibles y accesibles
- La mayoría de los sitios Web requieren estar operativos en un corto espacio de tiempo y esto dificulta el nivel de rigor de varias actividades del proceso

10.2.1. Situación actual

El sitio Web de una organización es una entrada a su información, productos y servicios. Como tal, debe idealmente ser una reflexión de las necesidades de los clientes que sirve. Desgraciadamente, el diseño y el desarrollo del sito web es conducido a menudo por la tecnología o por la estructura de organización u objetivos de negocio, más que por lo que el usuario necesita. En años recientes sin embargo, los dueños y diseñadores del sitio Web han comenzado gradualmente a reconocer la importancia de la usabilidad.

¿Cuál es el propósito del sitio web?

- Qué servicios se proporciona a los usuarios
- Apoyo técnico a los clientes
- Información del producto a clientes posibles

Dado el índice de crecimiento de la naturaleza de la Web, está claro que la usabilidad está llegando a ser tan importante que los diseñadores de Web deben tratarla. Los usuarios del Web quieren la satisfacción inmediata. Los servicios Web deben tratar los criterios de la usabilidad para ser claros, atractivos y fáciles de utilizar.

10.2.2. Definiciones de Usabilidad

Por medio de la usabilidad se asegura un diseño útil y se prueba que el usuario detecta de modo apropiado el uso del sistema y no de manera equívoca.

Existen diferentes definiciones

- Definición 1. La usabilidad [OGL+99] es una característica de alto nivel -que se puede medir mediante el cálculo a partir de métricas directas e indirectas- y representa la capacidad o potencialidad del producto para ser utilizado, comprendido y operado por los usuarios, además de ser atractivo. Incluye a subcaracterísticas como comprensibilidad, operabilidad y comunicatividad entre otras como estética y estilo que hacen que el artefacto sea agradable de usar.
- Definición 2. Podemos definir también la usabilidad [Nie01] como la medida en la cual un producto puede ser usado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado.
    - Efectividad: precisión y la plenitud con las que los usuarios alcanzan los objetivos especificados. A esta idea van asociadas la facilidad de aprendizaje (en la medida en que este sea lo más amplio y profundo posible), la tasa de errores del sistema y la facilidad del sistema para ser recordado (que no se olviden las funcionalidades ni sus procedimientos).

  - Eficiencia: recursos empleados en relación con la precisión y plenitud conque los usuarios alcanzan los objetivos especificados. A esta idea van asociadas la facilidad de aprendizaje (en tanto que supone un coste en tiempo; igualmente, si se requiere un acceso continuo a los mecanismos de ayuda del sistema), la tasa de errores del sistema y la facilidad del sistema para ser recordado (una asimilación inapropiada puede traducirse en errores de usuario).
  - Satisfacción: ausencia de incomodidad y la actitud positiva en el uso del producto. Se trata, pues, de un factor subjetivo.

- Definición 3. La usabilidad, hace referencia, a la rapidez y facilidad con que las personas llevan cabo sus tareas propias a través del uso del producto objeto de interés, idea que descansa en cuatro puntos:
  - Una aproximación al usuario: Usabilidad significa enfocarse en los usuarios. Para desarrollar un producto usable, se tienen que conocer, entender y trabajar con las personas que representan a los usuarios actuales o potenciales del producto.
  - Un amplio conocimiento del contexto de uso: Las personas utilizan los productos para incrementar su propia productividad. Un producto se considera fácil de aprender y usar en términos del tiempo que toma el usuario para llevar a cabo su objetivo, el número de pasos que tiene que realizar para ello, y el éxito que tiene en predecir la acción apropiada para llevar a cabo. Para desarrollar productos usables hay que entender los objetivos del usuario, hay que conocer los trabajos y tareas del usuario que el producto automatiza, modifica o embellece.

- El producto ha de satisfacer las necesidades del usuario: Los usuarios son gente ocupada intentando llevar a cabo una tarea. Se va a relacionar usabilidad con productividad y calidad. El hardware y el software son las herramientas que ayudan a la gente ocupada a realizar su trabajo y a disfrutar de su ocio.
- Son los usuarios, y no los diseñadores y los desarrolladores, los que determinan cuando un producto es fácil de usar.

La usabilidad no es sólo un número sino que tiene cinco características que claramente se deducen de los tres párrafos anteriores:

- Facilidad de aprender: ¿Cómo de rápido puede un usuario que nunca ha visto el interfaz a usar ejecutar tareas básicas?
- Eficacia de uso: ¿Una vez que el usuario haya aprendido a utilizar el sistema, cómo de rápido puede lograr tareas?
- Facilidad del sistema para ser recordado: ¿Si un usuario ha utilizado el sistema anteriormente, recuerda lo bastante para utilizarlo con más eficacia la próxima vez o no?
- Frecuencia y severidad del error: ¿Con qué frecuencia los usuarios cometen errores, cómo de serios son estos errores y cómo de fácil es solventar estos errores?
- Satisfacción subjetiva: ¿Al usuario le gusta utilizar el sistema, está satisfecho?

Algunas de las características son más importantes que otras. Para la Web, la facilidad de aprendizaje es a menudo la cualidad más importante de la usabilidad puesto que los usuarios no suelen pasar mucho tiempo en un sitio web que no entienden o no saben manejar. La satisfacción subjetiva también es un factor importante ya que los usuarios pueden irse libremente a otro sitio.

10.2.3. Razones de Servicios Web no usables

Los diseñadores y desarrolladores de un servicio Web se centran en la implementación técnica en lugar de centrarse en el usuario.

Los diseñadores de servicios Web piensan de forma distinta a como lo hace el usuario debido a sus características técnicas.

Los diseñadores Web siempre proponen el uso de los últimos avances y tecnologías y esto no siempre es beneficioso para el usuario.

Es necesario pues educar al diseñador sobre la necesidad de una buena usabilidad.

La usabilidad se basa en el DCU (Diseño Centrado en el Usuario), es decir, es contar con el usuario en el ciclo de diseño o rediseño.

10.2.4. Errores de usabilidad en sitios Web

El 90\% de los sitios Web presentan fallos de usabilidad. Los fallos más comunes son los siguientes:

- Indefinición del sitio (¿Qué es esto?): Es bueno que informemos claramente de que es lo que se puede encontrar aquí y cuales son las ventajas de utilizar esta página.
- Indefinición de las funciones a realizar (¿Dónde está lo que busco?): Hay que enseñar al usuario la ruta claramente de principio a fin. Si los caminos son complejos hay que simplificarlos.
- Búsquedas que no ofrecen resultados. Los buscadores son gratis y baratos, no pierdas el tiempo escribiendo uno. Implementa un buscador de palabras clave que funcione correctamente.
- Mala navegación / Enlaces perdidos / time out. La mayoría de los problemas de navegación se resolverán pintando la situación del usuario.
- Enlaces Perdidos. Los enlaces perdidos sólo demuestran un mal producto. Simplifica y reduce el número.
- Time Out. El time out se suele producir cuando la programación es redundante, servidores poco potentes o exceso de seguridad. No son justificables.
- Exceso de peso Se resuelve quitando imágenes que no sean necesarias y dando el formato a la página utilizando los estándares de HTML.

10.2.5. Principios o reglas de Usabilidad

Vamos a exponer someramente los principios de Steve Krug [Kru01] y de Jakob Nielsen [Nie01].

Steve Drug

- No me hagas pensar
- No le importa cuantos clicks sino que no sean difíciles
- Quitar la mitad de las palabras de la mitad

Jakob Nielsen

Según Nielsen existen 8 reglas básicas sobre la usabilidad en la Web, que son:

- En Internet el usuario es el que manda.
- En Internet la calidad se basa en la rapidez y la fiabilidad.
- En Internet cuenta que tu pagina sea más rápida que bonita, fiable que moderna, sencilla que compleja, directa.
- La confianza es algo que cuesta mucho ganar y se pierde con un mal enlace.
- Si quieres hacer una página decente, simplifica, reduce, optimiza: La gente no se va a aprender tu sitio por mucho que insistas, así que por lo menos hazlo sencillo, reutiliza todos los elementos que puedas, para que de este modo los usuarios se sientan cómodos y no se pierdan cada vez que necesiten encontrar algo en tu sitio.
- Pon las conclusiones al principio: El usuario se sentirá más cómodo si ve las metas al principio. De esta forma no tendrá que buscar lo que necesita y perderá menos tiempo en completar su tarea. Si completa su tarea en menos tiempo se sentirá cómodo y quizás se dedique a explorar tu sitio o quizás se lo recomiende a un amigo.

- No hagas perder el tiempo a la gente con cosas que no necesitan: Cuidado con cruzar promociones, si lo haces por lo menos hazlo con cuidado. Procura que la selección de productos a cruzar sea consecuente y no lo quieras "vender todo" en todas las páginas. Según avance el usuario en su navegación procura dejarle mas espacio libre. Puede ocurrir que cuando este punto de comprar algo vea una oferta que le distraiga y pierdas esa venta.
- Buenos contenidos: escribir bien para Internet es todo un arte. Pero siguiendo las reglas básicas de (1) poner las conclusiones al principio y (2) escribir como un 25\% de lo que pondrías en un papel, se puede llegar muy lejos. Leer en pantalla cuesta mucho, por lo que, en el caso de textos para Internet, reduce y simplifica todo lo que puedas.

10.3. EL MODELO VISTA CONTROLADOR (MVC)

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software inventado por Trygve Reenskaug [Ree] e introducido en el entorno de desarrollo del SmallTalk 80 [Sma] desarrollado en XEROX PARC.

El MVC reparte las responsabilidades de la aplicación entre tres elementos:

- Modelo. Es la representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de dominio o negocio. La lógica de negocio añade significado a los datos. El modelo encapsula los datos y reglas específicos de la aplicación, es decir, la capa de negocio y la capa persistencia. El modelo nos aporta:
    - Métodos para el manejo de datos y servicios
    - Métodos para acceder al estado del sistema.
    - Mantiene registro de las diferentes vistas y controladores para notificar los cambios (Modelo de eventos).

- Vista. Presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario, por tanto es el mecanismo encargado de realizar la correspondencia entre los datos provenientes del modelo y la interfaz. Cuando el Modelo cambia, la vista es informada. Sus funciones básicas son por tanto: solicitar al modelo la información y responsabilizarse de actualizar la pantalla
- Controlador. Responde a eventos, es decir, realiza el control del flujo de navegación de la aplicación. Intercepta los eventos de entrada provenientes del usuario del sistema y traduce los eventos en invocaciones al modelo de la aplicación. Además activa o desactiva los elementos de la interfaz de usuario.

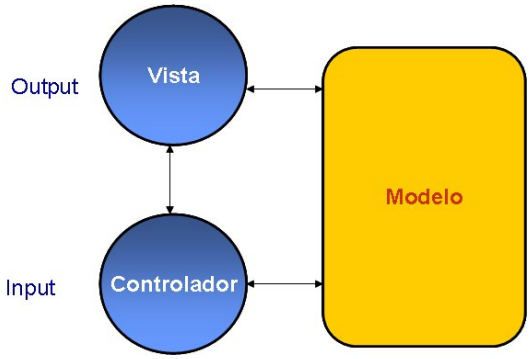


Figura 10.1.Arquitectura MVC.

10.3.1. Flujo de la Información

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

- El usuario interactúa con la interfaz de usuario de alguna forma, por ejemplo pulsando un botón.
- El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos.
- El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario.

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

10.3.2. Implementaciones

El patrón MVC se ha aplicado recientemente para interfaces Web, en este contexto el MVC se suele llamar "Model 2". Las aplicaciones Web son más complejas que las aplicaciones tradicionales, y el MVC se utiliza como una solución potencial de esas dificultades.

Las implementaciones orientadas a la Web del MVC más utilizadas actualmente son:

- JavaServer Faces
- Apache Struts
- XForms

Para realizar la implementación del MVC en Java, habitualmente se utilizan los siguientes convenios:

- Para la implementación del **Controlador** se utilizan Servlet:
    - Servlet central recibe peticiones, procesa URL recibida y delega procesamiento a JavaBeans
    - Servlet guarda resultado de procesamiento realizado por JavaBeans en el contexto de la petición, la sesión o la aplicación

- Servlet transfiere control a un JSP que lleva a cabo la presentación de resultados

- Para la implementación del **Modelo** se utilizan JavaBeans (o EJBs para aplicaciones más escalables).
- La **Vista** se utilizan JSP

10.3.2.1. JavaServer Faces

La tecnología JavaServer Faces [JSF] es un marco de trabajo de interfaces de usuario del lado de servidor para aplicaciones Web basadas en tecnología Java.

Los principales componentes de la tecnología JavaServer Faces son:

- Un API(Application Programming Interface) y una implementación de referencia para: representar componentes de interfaz de usuario y manejar su estado, realizar el manejo de eventos, validación del lado del servidor y conversión de datos, definir la navegación entre páginas, soportar internacionalización y accesibilidad y proporcionar extensibilidad para estas características.
- Una librería de etiquetas JavaServer Pages [JSP] personalizadas para dibujar componentes de interfaz de usuario dentro de una página JSP.

El modelo de programación y la librería de etiquetas para componentes de interfaz de usuario facilitan la tarea de la construcción y mantenimiento de aplicaciones Web con interfaces de usuario del lado del servidor, dado que:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Traducir componentes de interfaz a una página de datos del lado del servidor.
- Construir un interfaz con componentes reutilizables y extensibles.
- Grabar y restaurar el estado del interfaz más allá de la vida de las peticiones de servidor.

- Como se puede apreciar en la siguiente Figura 10.2, el interface de usuario que se crea con la tecnología JavaServer Faces (representado por myUI en el gráfico) se ejecuta en el servidor y se renderiza en el cliente.

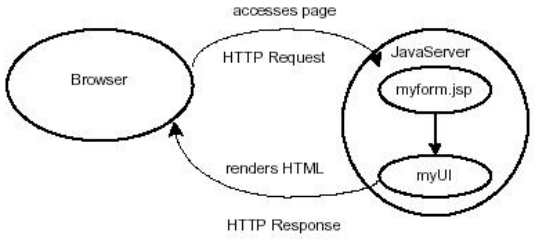


Figura 10.2.Representación de JavaServer Faces.

La página JSP, myform.jsp, dibuja los componentes del interface de usuario con etiquetas personalizadas definidas por la tecnología JavaServer Faces. El interfaz de la aplicación Web (representado por myUI en la magen) maneja los objetos referenciados por la página JSP: los objetos componentes que traducen las etiquetas sobre la página JSP, los manejadores de eventos, validadores, y los conversores que está registrados en los componentes, y por último, los objetos del modelo que encapsulan los datos y las funcionalidades de los componentes específicos de la aplicación.

10.3.2.2. Apache Struts

Fué creada por Craig McClanahan y donada a la Apache Software Foundation en el 2000 (pertenece a Apache Jakarta). Struts [STR] proporciona:

- Un servlet (ActionServlet) que actúa como controlador MVC totalmente configurable
- Clases base que son extendidas para implementar la lógica de la aplicación web: Struts Action y Struts ActionForm
- Un conjunto de etiquetas personalizadas JSP que cooperan con el controlador para su uso en la capa view de MVC
- Varias opciones para la validación de entrada de usuario en formularios HTML: ActionForm o Validator Framework
- Mecanismos para el manejo de errores

- Soporte para la internacionalización (i18n) a través de ficheros de recursos y Java Locales
- Soporte para fuentes de datos

Para crear una aplicación con Struts se deben seguir los siguientes pasos:

1. Diseñar la aplicación en términos de las acciones, vistas y estados del modelo.
2. Añadir las librerías Java de Struts y las etiquetas personalizadas del proyecto.
3. Configurar un fichero web.xml para que envíe peticiones HTTP al ActionServet.
4. Configurar el ActionServlet definiendo elementos <actionmappings> y <form-beans> en struts-config.xml.
5. Definir las clases Action y ActionForm.
6. Definir las clases adicionales Java que representan la lógica de negocio.
7. Definir las páginas de presentación JSP.
8. Pasar a explotación la aplicación.

El control de flujo en Struts (Figura 10.3) se puede resumir de la siguiente forma:

La clase org.apache.struts.action.ActionServlet es el eje de Struts. Dada una petición de entrada HTTP:

- Crea un objeto ActionForm donde guarda y valida los parámetros de entrada
- Decide que objeto Action se debe invocar y le pasa el objeto ActionForm creado
- Transfiere control a la siguiente etapa de procesamiento de la petición (típicamente un JSP).

El fichero de configuración web.xml contiene los url mappings para enviar las peticiones de llegada al ActionServlet, mientras que el fichero de configuración de Struts struts-config.xml contiene los mappings a acciones

Los form beans creados por ActionServlet deben ser implementados por el programador, extendiendo org.apache.struts.action.ActionForm.

El programador deberá definir un conjunto de getters y setter y sobreescribir los métodos validate() y reset()

Los objetos Action invocados deben ser desarrollados por el programador y extienden org.apache.struts.action.Action. Tienen un método execute() que ejecuta la lógica de negocio

La acción devuelve un objeto ActionForward al servlet que especifica el siguiente paso a ejecutar, normalmente se transfiere el control a un JSP para que visualice los resultados.

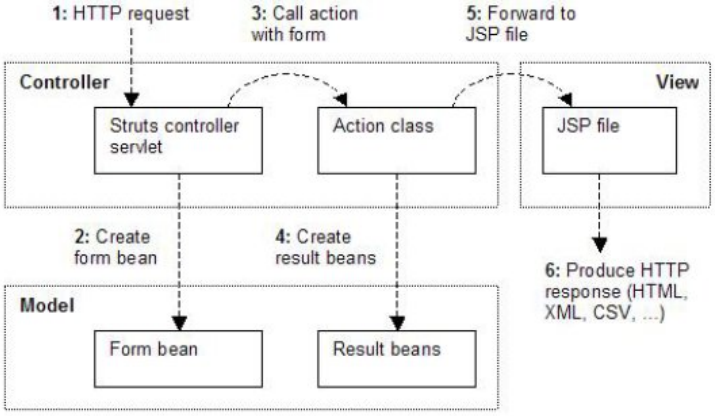


Figura 10.3.Control de Flujo en Struts.

10.3.2.3. XForms

XForms es una recomendación oficial del W3C [XFo04]. Es un nuevo lenguaje de etiquetado para formularios Web, diseñado para ser el sustituto de los formularios tradicionales HTML, y que permite distinguir entre el propósito del formulario y la presentación. Esta separación clara entre contenido y presentación ofrece grandes ventajas en términos de:

- Reutilización: los módulos XForms pueden reutilizarse independientemente de los datos que recogen.
- Independencia de Dispositivo: gracias a que los controles de la interfaz de usuario son abstractos y sólo se indican sus características genéricas, lo que permite su representación en diferentes dispositivos

- Accesibilidad: al separar presentación y contenido la información está disponible de forma más sencilla para los usuarios que precisen de ayudas técnicas para la navegación en la Web.

XForms está compuesto por secciones separadas que describen lo que hace el formulario y cómo se va a mostrar. Los formularios tradicionales de HTML no separaban el propósito de la presentación, sin embargo XForms sí que lo hace a través de esas secciones separadas. Las tres partes en las que se divide un formulario XForms son el Modelo XForms, los datos de la instancia y la interfaz del usuario, lo que proporciona una gran flexibilidad en las opciones de presentación.

```
<model>
   <submission action="http://ejemplo.com/buscar" method="get" id="s"/>
</model>
```

Código Fuente 10.1.Ejemplo del encabezado de un formulario XForms.

En un formulario con XForms los detalles de los valores recogidos y los detalles de la forma de enviar esos valores se recoge en un encabezado del documento, en concreto dentro de un elemento llamado "model" (modelo), como se muestra en el Código Fuente 10.1. En el cuerpo del documento sólo aparecerán los controles como se muestra en el Código Fuente 10.2.

```
<h:html xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns="http://www.w3.org/2002/xforms">
 <h:head>
   <h:title>Buscar</h:title>
   <model>
     <submission action="http://ejemplo.com/buscar" method="get" id="s"/>
   </model>
 </h:head>
 <h:body>
   <h:p>
     <input ref="q"><label>Buscar</label></input>
     <submit submission="s"><label>Enviar</label></submit>
   </h:p>
 </h:body>
</h:html>
```

Código Fuente 10.2.Ejemplo del cuerpo de un formulario XForms

XForms tiene la capacidad de trabajar con gran variedad de estándares o interfaces de usuario (Figura 10.4).La Interfaz de usuario de XForms proporciona un conjunto estándar de controles visuales con el objetivo de reemplazar los controles de XHTML. Estos controles se usan directamente en XHTML y en otros documentos XML, como SVG.

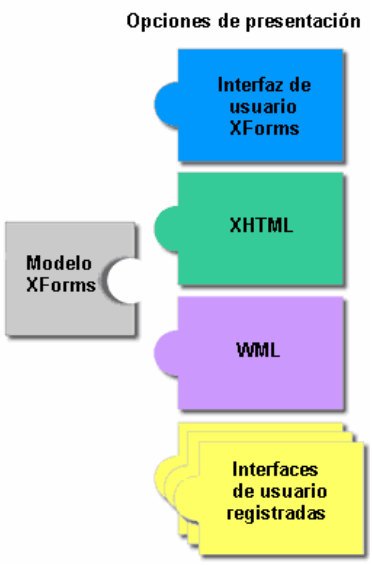


Figura 10.4.Modelo XForms.

Los formularios de XForms recogen datos (Código Fuente 10.3), lo que se expresa como datos de instancia XML. XForms Model se encarga de describir la estructura de esos datos de instancia, lo que muestra un intercambio estructurado de datos.

```
<model>
  <instance><data xmlns=""><q/></data></instance>
  <submission action="http://ejemplo.com/buscar" method="get" id="com"/>
  <submission action="http://ejemplo.org/buscar" method="get" id="org"/>
</model>
```

Código Fuente 10.3. Ejemplo del encabezado de un formulario XForms.

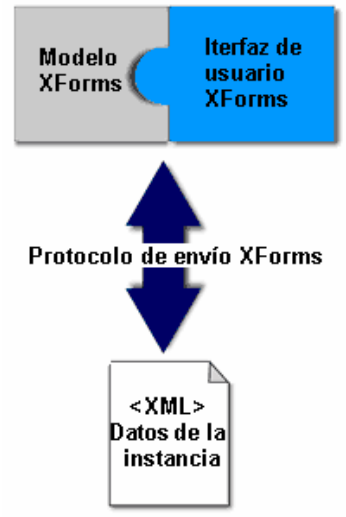


Figura 10.5.Visión global de XForms.

Además, debe existir un canal por el que los datos de instancias se muevan hacia y desde un procesador XForms. Para ello, el Protocolo de envío de XForms define la forma en la que XForms envía y recibe datos, incluyendo la capacidad de cancelar y reanudar la finalización de un formulario (Figura 10.5).

11. PLATAFORMAS: J2EE Y .NET

11.1. LA PLATAFORMA J2EE

Sun Microsystems ha desarrollado su nueva plataforma Java 2 que se presenta en tres versiones diferentes: Standard Edition (J2SE), Enterprise Edition (J2EE) y Micro Edition (J2ME).

Las características más amplias se encuentran presentes en la J2EE por tanto es la que se utiliza de referencia en el trabajo.

J2EE es un grupo de especificaciones diseñadas por Sun que permiten la creación de aplicaciones empresariales, como: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI/CORBA), funciones de correo electrónico (JavaMail), aplicaciones Web (JSP y Servlets), etc.

Es importante notar que J2EE es solo una especificación, esto permite que diversos productos sean diseñados alrededor de estas especificaciones como Tomcat, Weblogic… Aunque varios productos Java están diseñados alrededor de estas especificaciones, no todos cumplen con el estándar completo, por ejemplo Tomcat solo emplea las especificaciones de JSP y Servlets, y otros como Websphere cumplen con todas las especificaciones definidas por Sun.

A continuación se enumeran algunas de las tecnologías que incorpora la plataforma J2EE:

- Java API for XML-Based RPC (JAX-RPC)
- JavaServer Pages (JSPs)
- Java Servlets
- Enterprise JavaBeans (EJBs)
- J2EE Connector Architecture
- J2EE Management Model
- J2EE Deployment API
- Java Management Extensions (JMX)
- J2EE Authorization Contract for Containers
- Java API for XML Registries (JAXR)
- Java Message Service (JMS)
- Java Naming and Directory Interface (JNDI)
- Java Transaction API (JTA)
- CORBA
- JDBC data access API

La ecuación de referencia, que marca los puntos que van a ser desarrollados, es: J2EE = JVM + Lenguaje Java + API Java + Utilidades

11.1.1. JVM - Java Virtual Machine

La Máquina Virtual Java es el núcleo del lenguaje de programación Java. De hecho, es imposible ejecutar un programa Java sin ejecutar alguna implantación de la JVM. En la JVM se encuentra el motor que en realidad ejecuta el programa Java y es la clave de muchas de las características principales de Java, como la portabilidad, la eficiencia y la seguridad.

La especificación de la JVM, define elementos como el formato de los archivos de clases de Java *(.class)*, así como la semántica de cada una de las instrucciones que componen el conjunto de instrucciones de la máquina virtual.

La representación de los códigos de instrucción Java (bytecode) es simbólica, en el sentido de que los desplazamientos e índices dentro de los métodos no son constantes, son nombres simbólicos. Estos nombres son resueltos la primera vez que se ejecuta el método, es decir, el nombre simbólico se busca dentro del archivo de clase y se determina el valor numérico del desplazamiento.

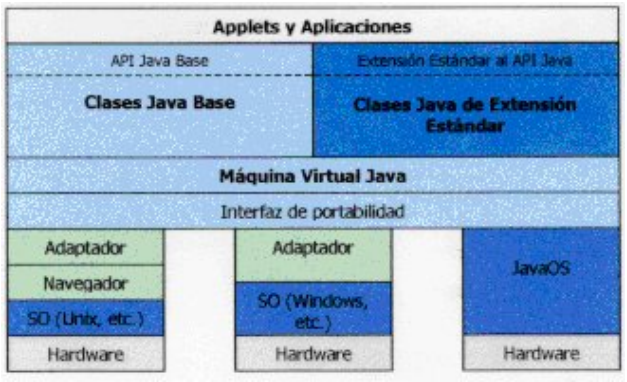


Figura 11.1.La máquina virtual de Java (JVM).

En la Figura 11.1 puede observarse la capa de software que implementa a la máquina virtual Java. Esta capa de software oculta los detalles inherentes a la plataforma, a las aplicaciones Java que se ejecuten sobre ella. Debido a que la plataforma Java fue diseñada pensando en que se implementaría sobre una amplia gama de sistemas operativos y de procesadores, se incluyeron dos capas de software para aumentar su portabilidad. La primera dependiente de la plataforma es llamada adaptador, mientras que la segunda, que es independiente de la plataforma, se le llama interfaz de portabilidad. De esta manera, la única parte que se tiene que escribir para una plataforma nueva, es el adaptador. El sistema operativo proporciona los servicios de manejo de ventanas, red, sistema de archivos, etcétera.

A continuación se enumeran las características más destacables de la máquina virtual de Java:

- Pila de ejecución. La JVM se basa en la utilización de una pila de ejecución y un repertorio de instrucciones que manipulan dicha pila.
- Código multi-hilo. La máquina virtual puede dar soporte a varios hilos (threads) de ejecución concurrente. Estos hilos ejecutan código de forma independiente sobre un conjunto de valores y objetos que residen en una memoria principal compartida. La sincronización de los hilos se realiza mediante monitores, un mecanismo que permite ejecutar a un solo hilo una determinada región de código.
- Compilación JIT. Un programa compilado se representa mediante un conjunto de ficheros de código de bytes (ficheros class) que se cargan de forma dinámica y que contienen una especificación sobre el comportamiento de una clase. La separación en módulos independientes permite la compilación a código nativo de los códigos de bytes en el momento en que se necesita ejecutar un módulo.
- Verificación estática de Código de bytes. Los ficheros class contienen información sobre el comportamiento del módulo que puede verificarse antes de su ejecución. Es posible verificar estáticamente de que el programa no va a producir determinados errores al ejecutarse.
- Gestión de memoria dinámica. La máquina integra un recolector de basura, liberando al programador de gestionar la memoria dinámica.

- Dependencia del lenguaje Java. La máquina ha sido diseñada para ejecutar programas Java, adaptándose fuertemente al modelo de objetos de Java. Incluye instrucciones de gestión de clases, interfaces, etc. Esta característica perjudica la compilación a JVM de lenguajes diferentes a Java.

Lo más destacable de este apartado es que los bytecodes de Java permiten hacer cierto el lema de sus creadores *"write once, run anywhere"*. Se escribe un programa en lenguaje Java y se compila a bytecodes sobre cualquier plataforma que tenga un compilador de Java (Figura 11.2). Los bytecodes pueden ejecutarse sobre cualquier implementación de la JVM, el mismo programa escrito en Java puede ejecutarse en Windows XP, una estación Solaris, o un iMac.

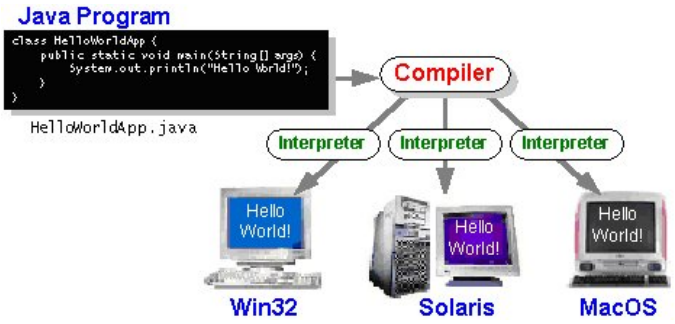


Figura 11.2.Compilación de un programa Java.

11.1.2. Lenguaje Java

Este lenguaje fue desarrollado por J. Gosling en 1993 [GJS04] como un lenguaje orientado a objetos para dispositivos electrónicos empotrados. Alcanzó gran popularidad en 1996 cuando Sun Microsystems hizo pública su implementación.

La sintaxis del lenguaje se inspira en la de C++ [Str98] pero contiene un conjunto más reducido de características. Incluye un sistema de gestión de memoria y un mecanismo de tratamiento de excepciones y concurrencia.

Las implementaciones se basan en una máquina virtual estándar, la JVM. El lenguaje alcanza gran popularidad como lenguaje para desarrollo de aplicaciones en Internet puesto que la JVM es incorporada en muchos servidores y clientes.

Vamos a ver algunas características que el fabricante determina de este lenguaje:

- Sencillo. Algunas de las razones de esta característica son: fácil aprendizaje, no posee sobrecarga de operadores, no tiene herencia múltiple, no posee punteros explícitos, además tiene una gestión de memoria automática con recolector de basura y posee una jerarquía de clases muy rica ya definida.
- Orientado a Objetos Puro.Las características de este paradigma que soporta son: abstracción, encapsulación, herencia simple, polimorfismo y clases abstractas. Además todo se estructura en clases excepto los tipos simples, no existen funciones libres porque todo deben ser métodos de clases. Fomenta la reusabilidad.
- Interpretado y compilado.Utiliza un lenguaje intermedio, bytecode, para la máquina virtual, JVM. Este bytecode puede interpretarse o compilarse.
- Distribuido y con multihilos. Soporta múltiples hilos concurrentemente como se especifico anteriormente. Además posee clases y primitivas para el manejo de la sincronización.
- Independiente de la plataforma y portable. Al utilizar bytecode interpretable por la JVM se puede ejecutar en diversas plataformas el mismo software desarrollado.
- Robusto y seguro. Se realiza un control estricto de tipos en tiempo de compilación y ejecución. Además el lenguaje posee restricciones contra la corrupción de la memoria. Y posee recolector de basura.

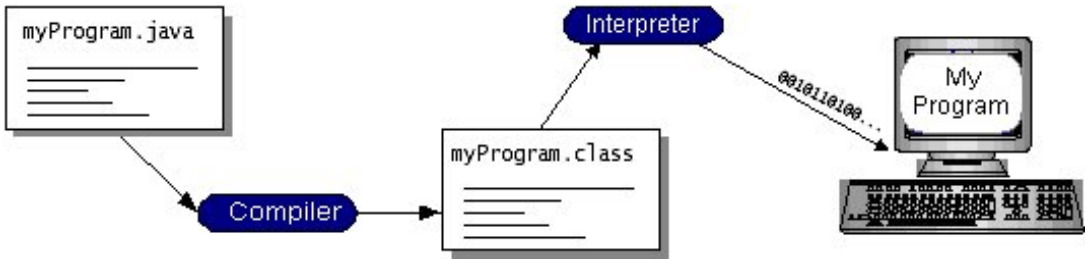


Figura 11.3.Pasos para ejecutar un programa Java.

11.1.3. Java API (Java Application Programming Interface)

El API de Java es una colección extensa de componentes que aportan diferentes capacidades, como interfaces gráficos (GUI). El API de Java esta agrupado dentro de bibliotecas de clases e interfaces, conocidas como paquetes (packages).

En la Figura 11.4 se muestra como se ejecuta un programa dentro de la plataforma J2EE, y tal como muestra la figura, el API y la JVM aíslan el programa del hardware.

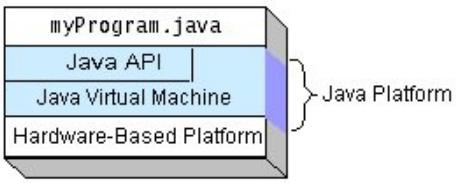


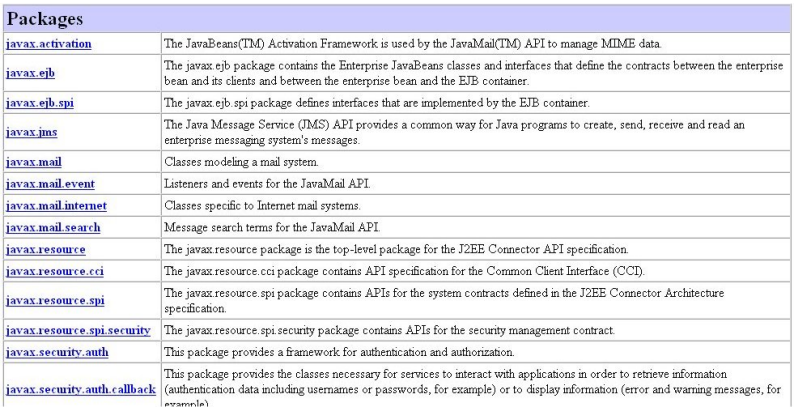
Figura 11.4. Aspecto de la ejecución de un programa.

Estos paquetes de componentes software aportan gran parte de la funcionalidad necesaria. Todas las implementaciones de la plataforma Java, incluida J2EE, dan las siguientes funcionalidades, dentro del API:

- Los esenciales: objetos, strings, hilos, números, entrada y salida, estructuras de datos, propiedades del sistema, tiempo y hora, y algunas mas.
- Applets: el conjuntos de convenciones utilizadas por los applets
- Networking: URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, e direcciones IP (Internet Protocol).
- Internacionalización: ayudas para escribir programas que puedan ser encontrados por usuarios de la web. Los programas pueden, automáticamente, adaptarse a las especificaciones locales y ser visualizados en el lenguaje apropiado.
- Seguridad: a bajo y alto nivel, incluyendo firma electrónica, manejo de claves públicas y privadas, control de acceso y certificados.
- Componentes software: JavaBeans
- Serialización de objetos: comunicación via RMI (Remote Method Invocation)

- Java Database Connectivity (JDBCTM): provee de una acceso uniforme a un amplio rango de bases de datos relacionales.

La plataforma Java además tiene APIs para gráficos de 2D y 3D, accesibilidad, servidores, colaboración, telefonía, conversación, animación y más. En la Figura 11.5 se muestran algunos de estos paquetes.



Packages	
javax.activation	The JavaBeans(TM) Activation Framework is used by the JavaMail(TM) API to manage MIME data.
javax.ejb	The javax.ejb package contains the Enterprise JavaBeans classes and interfaces that define the contracts between the enterprise bean and its clients and between the enterprise bean and the EJB container.
javax.ejb.spi	The javax.ejb.spi package defines interfaces that are implemented by the EJB container.
javax.jms	The Java Message Service (JMS) API provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages.
javax.mail	Classes modeling a mail system.
javax.mail.event	Listeners and events for the JavaMail API.
javax.mail.internet	Classes specific to Internet mail systems.
javax.mail.search	Message search terms for the JavaMail API.
javax.resource	The javax.resource package is the top-level package for the J2EE Connector API specification.
javax.resource.cci	The javax.resource.cci package contains API specification for the Common Client Interface (CCI).
javax.resource.spi	The javax.resource.spi package contains APIs for the system contracts defined in the J2EE Connector Architecture specification.
javax.resource.spi.security	The javax.resource.spi.security package contains APIs for the security management contract.
javax.security.auth	This package provides a framework for authentication and authorization.
javax.security.auth.callback	This package provides the classes necessary for services to interact with applications in order to retrieve information (authentication data including usernames or passwords, for example) or to display information (error and warning messages, for example)

Figura 11.5.Paquetes de J2EE.

11.1.4. Utilidades

La plataforma J2EE no posee ninguna herramienta de utilidad específica. Los fabricantes que han realizado implementaciones de la plataforma si que han incluido una serie de herramientas para facilitar la usabilidad y administración de las aplicaciones realizadas para la plataforma.

Como ejemplo se enumeran a continuación una serie de herramientas incluidas por Sun Microsystems en su implementación de la plataforma J2EE: Java 2 SDK (Standard Development Kit) versión 1.3.1, Enterprise Edition (J2EE SDK).

J2EE Administration Tool: Es un script en línea de comandos que permite añadir y borrar recursos.

- Cleanup Tool: Es un script en línea de comandos que borra todos las aplicaciones del servidor J2EE. No borra los ficheros de componentes (JAR, WAR, EAR)
- Deployment Tool: es una utilidad con dos versiones: GUI y línea de comandos. La versión GUI permite empaquetar componentes y desplegar aplicaciones. La versión en línea de comandos plegar y desplegar aplicaciones.

- J2EE Server: ejecuta el script de la plataforma desde línea de comandos
- Key Tool: esta utilidad crea claves públicas y privadas y genera certificados.
- Packager Tool: es un script en línea de comandos que permite empaquetar componentes J2EE. Se pueden crear los siguientes paquetes de componentes:
    - Fichero de especificación de JavaBeans - EJB JAR
    - Fichero de aplicación Web - WAR
    - Fichero de aplicación cliente - JAR
    - Fichero de aplicación J2EE - EAR
- Realm Tool: es un script que permite añadir y borrar usuarios de J2EE e importer fichero de certificados
- Runclient Script: esta utilidad permite ejecutar una aplicación cliente J2EE
- Verifier Tool: valida un archive J2EE (EAR, WAR, JAR)

11.2. LA PLATAFORMA .NET

La plataforma .NET es desarrollada por Microsoft y aparece en una primera implementación en el año 2000. Esta plataforma intenta simplificar el desarrollo de aplicaciones en un entorno altamente distribuido como es Internet.

Los objetivos básicos que se plantearon cumplir en el diseño de la plataforma .NET [Pla03] eran:

- Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.
- Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.

- Ofrecer un entorno de ejecución de código que garantice la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan secuencias de comandos o intérpretes de comandos.
- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.
- Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código.

La máquina virtual de la plataforma .NET se denomina CLR (Common Language Runtime) y es el fundamento de la tecnología [Gou01]. El motor de tiempo de ejecución se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la interacción remota, al tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que garantizan su seguridad y solidez. El concepto de administración de código es un principio básico del motor de tiempo de ejecución. El código destinado al motor de tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado.

La biblioteca de clases de .NET Framework, es el otro componente principal de la plataforma .NET, es una completa colección orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones que abarcan desde las tradicionales herramientas de interfaz gráfica de usuario (GUI) o de línea de comandos hasta las aplicaciones basadas en las innovaciones más recientes proporcionadas por ASP.NET, como los formularios Web Forms y los servicios Web XML.

La plataforma .NET o .NET Framework, como la empresa propietaria prefiere denominarla, puede alojarse en componentes no administrados que cargan CLR en sus procesos e inician la ejecución de código administrado, con lo que se crea un entorno de software en el que se pueden utilizar características administradas y no administradas. En .NET Framework no sólo se ofrecen varios hosts de motor de tiempo de ejecución, sino que también se admite el desarrollo de estos hosts por parte de terceros.

Por ejemplo, ASP.NET aloja el motor de tiempo de ejecución para proporcionar un entorno de servidor escalable para el código administrado. ASP.NET trabaja directamente con el motor de tiempo de ejecución para habilitar aplicaciones de formularios Web Form y servicios Web XML, que se tratan más adelante en este tema [EHM+05].

Internet Explorer es un ejemplo de aplicación no administrada que aloja el motor de tiempo de ejecución (en forma de una extensión de tipo MIME). Al usar Internet Explorer para alojar el motor de tiempo de ejecución, puede incrustar componentes administrados o controles de Windows Forms en documentos HTML. Al alojar el motor de tiempo de ejecución de esta manera se hace posible el uso de código móvil administrado (similar a los controles de Microsoft ActiveX), pero con mejoras significativas que sólo el código administrado puede ofrecer, como la ejecución con confianza parcial y el almacenamiento aislado de archivos seguros.

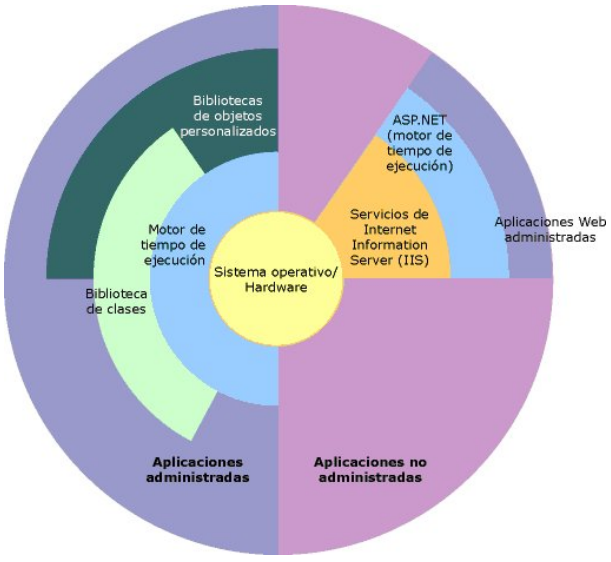


Figura 11.6.Relación de CLR y las aplicaciones.

En la Figura 11.6 se muestra la relación del CLR y la biblioteca de clases con las aplicaciones y el sistema en su conjunto. Además se muestra como funciona el código administrado dentro de una arquitectura mayor.

11.2.1. CLR - Common Language Runtime

Common Language Runtime (CLR) es la máquina virtual de la plataforma .NET. Administra la memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código, compilación y demás servicios del sistema. Estas características son intrínsecas del código administrado que se ejecuta en la CLR.

Con respecto a la seguridad, los componentes administrados reciben grados de confianza diferentes, en función de una serie de factores entre los que se incluye su origen (como Internet, red empresarial o equipo local). Esto significa que un componente administrado puede ser capaz o no de realizar operaciones de acceso a archivos, operaciones de acceso al Registro y otras funciones delicadas, incluso si se está utilizando en la misma aplicación activa.

El motor de tiempo de ejecución impone seguridad en el acceso al código. Por ejemplo, los usuarios pueden confiar en que un archivo ejecutable incrustado en una página Web puede reproducir una animación en la pantalla o entonar una canción, pero no puede tener acceso a sus datos personales, sistema de archivos o red. Por ello, las características de seguridad del motor de tiempo de ejecución permiten que el software legítimo implementado en Internet sea excepcionalmente variado.

El motor de tiempo de ejecución impone la solidez del código mediante la implementación de una infraestructura estricta de comprobación de tipos y código denominado CTS (Common Type System, Sistema de tipos común). CTS garantiza que todo el código administrado es autodescriptivo. Los diferentes compiladores de lenguajes de Microsoft y de terceros generan código administrado que se ajusta a CTS. Esto significa que el código administrado puede usar otros tipos e instancias administrados, al tiempo que se aplica inflexiblemente la fidelidad y seguridad de los tipos.

El entorno administrado del motor de tiempo de ejecución elimina muchos problemas de software comunes. Por ejemplo, el motor de tiempo de ejecución controla automáticamente la disposición de los objetos, administra las referencias a éstos y los libera cuando ya no se utilizan. Esta administración automática de la memoria soluciona los dos errores más comunes de las aplicaciones: la pérdida de memoria y las referencias no válidas a la memoria.

Además, el motor de tiempo de ejecución aumenta la productividad del programador. Los programadores pueden crear aplicaciones en el lenguaje que prefieran y seguir sacando todo el provecho del motor de tiempo de ejecución, la biblioteca de clases y los componentes escritos en otros lenguajes por otros colegas. El proveedor de un compilador puede elegir destinarlo al motor de tiempo de ejecución. Los compiladores de lenguajes que se destinan a .NET Framework hacen que las características de .NET Framework estén disponibles para el código existente escrito en dicho lenguaje, lo que facilita enormemente el proceso de migración de las aplicaciones existentes.

Aunque el motor de tiempo de ejecución está diseñado para el software del futuro, también es compatible con el software actual y el software antiguo. La interoperabilidad entre el código administrado y no administrado permite que los programadores continúen utilizando los componentes COM y las DLL que necesiten.

El motor de tiempo de ejecución está diseñado para mejorar el rendimiento. Aunque CLR proporciona muchos servicios estándar de motor de tiempo de ejecución, el código administrado nunca se interpreta. La compilación JIT (Just-In-Time) permite ejecutar todo el código administrado en el lenguaje máquina nativo del sistema en el que se ejecuta. Mientras tanto, el administrador de memoria evita que la memoria se pueda fragmentar y aumenta la zona de referencia de la memoria para mejorar aún más el rendimiento.

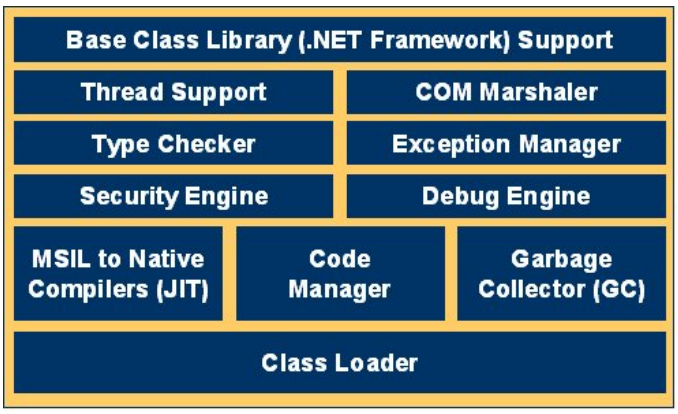


Figura 11.7.Estructura de la CLR.

Además el motor de tiempo de ejecución se puede alojar en aplicaciones de servidor de gran rendimiento, como IIS (Servicios de Internet Information Server). Esta infraestructura permite utilizar código administrado para escribir lógica empresarial, al tiempo que se obtiene un rendimiento de los mejores servidores empresariales del sector que puedan alojar el motor de tiempo de ejecución.

Para clarificar todos estos conceptos se puede decir que CLR es un entorno computacional desarrollado por Microsoft para la plataforma .NET cuyas características principales se muestran en los siguientes apartados.

11.2.1.1. Utilización de una pila de ejecución

La máquina virtual de la plataforma .NET es una máquina de pila.

11.2.1.2. Compilación JIT

Para poder ejecutar el lenguaje intermedio de Microsoft (MSIL), primero se debe convertir éste, mediante un compilador Just-In-Time (JIT) de .NET Framework, en código nativo, que es el código específico de la CPU que se ejecuta en la misma arquitectura de equipo que el compilador JIT. CLR proporciona un compilador JIT para cada arquitectura de CPU compatible, por tanto, con este compilador los programadores pueden escribir un conjunto de MSIL que se puede compilar y ejecutar en equipos con diferentes arquitecturas.

No obstante, el código administrado sólo se ejecutará en un determinado sistema operativo si llama a la plataforma específica, a las API nativas o a una biblioteca de clases específica de la plataforma. La compilación JIT tiene en cuenta el hecho de que durante la ejecución nunca se llamará a parte del código.

En vez de utilizar tiempo y memoria para convertir todo el MSIL de un archivo ejecutable portable (PE) en código nativo, convierte el MSIL necesario durante la ejecución y almacena el código nativo resultante para que sea accesible en las llamadas posteriores.

El cargador crea y asocia un código auxiliar a cada uno de los métodos del tipo cuando éste se carga. En la llamada inicial al método, el código auxiliar pasa el control al compilador JIT, el cual convierte el MSIL del método en código nativo y modifica el código auxiliar para dirigir la ejecución a la ubicación del código nativo. Las llamadas posteriores al método compilado JIT pasan directamente al código nativo generado anteriormente, reduciendo el tiempo de la compilación y ejecución del código [Lid02].

11.2.1.3. Generación de código en tiempo de instalación

El motor de tiempo de ejecución proporciona otro modo de compilación denominado generación de código en el momento de la instalación. Convierte el MSIL en código nativo, como lo hace el compilador JIT normal, aunque convierte mayores unidades de código a la vez, almacenando el código nativo resultante para utilizarlo posteriormente al cargar y ejecutar el ensamblado. Cuando se utiliza la opción de generación de código durante la instalación, todo el ensamblado que se está instalando se convierte en código nativo, teniendo en cuenta las características de los otros ensamblados ya instalados. El archivo resultante se carga e inicia más rápidamente que si se hubiese convertido en código nativo con la opción JIT estándar

11.2.1.4. Verificación estática de tipos

Como parte de la compilación MSIL en código nativo, el código debe pasar un proceso de comprobación, a menos que el administrador haya establecido una directiva de seguridad que permita al código omitir esta comprobación. En esta comprobación se examina el MSIL y los metadatos para determinar si el código garantiza la seguridad de tipos, lo que significa que el código sólo tiene acceso a aquellas ubicaciones de la memoria a las que está autorizado. La seguridad de tipos es necesaria para garantizar que los objetos están aislados entre sí y, por tanto, protegidos contra daños involuntarios o maliciosos. Además, garantiza que las restricciones de seguridad sobre el código se aplican con toda certeza [MRM03]

11.2.1.5. Gestión dinámica de memoria

El recolector de elementos no utilizados de .NET Framework administra la asignación y liberación de la memoria de la aplicación. Cada vez que se utiliza un operador new para crear un objeto, el motor de tiempo de ejecución asigna al objeto memoria del montón administrado. Siempre que haya espacio de direcciones disponible en el montón nativo, el motor de tiempo de ejecución continúa asignando espacio a los objetos nuevos. No obstante, la memoria no es infinita. En ocasiones, el recolector de elementos no utilizados debe realizar una recolección para liberar alguna memoria. El motor de optimización del recolector de elementos no utilizados determina cuál es el mejor momento para realizar una recolección, según las asignaciones que se estén realizando. Cuando el recolector de elementos no utilizados realiza una recolección, comprueba si en el montón administrado hay objetos que la aplicación ya no utiliza y realiza las operaciones necesarias para reclamar su memoria.

11.2.1.6. Independiente del lenguaje

En el diseño del CLR se ha prestado gran importancia a su utilización como plataforma de ejecución de numerosos lenguajes de programación. Aunque se adopta un determinado modelo de objetos determinado, se incluyen facilidades que permiten desarrollar otros mecanismos de paso de parámetros e incluir tipos de datos primitivos en la pila de ejecución. De hecho, la plataforma ha sido adoptada como destino de lenguajes de diversos paradigmas como Basic, Haskell, Mercory, etc.

Para poder interactuar completamente con otros objetos, sea cual sea el lenguaje en que se hayan implementado, los objetos deben exponer a los llamadores sólo aquellas características que sean comunes para todos los lenguajes con los que deben interoperar. Por este motivo se ha definido un conjunto de características de lenguaje, denominado Common Language Specification (CLS), que incluye las características de lenguaje básicas que requiere la mayoría de las aplicaciones.

Las reglas de CLS definen un subconjunto del sistema de tipos común, es decir, todas las reglas aplicables al sistema de tipos común se aplican a CLS, excepto cuando se definan reglas más estrictas en CLS.

CLS ayuda a mejorar y garantizar la interoperabilidad entre lenguajes mediante la definición de un conjunto de características en las que se pueden basar los programadores y que están disponibles en una gran variedad de lenguajes.

CLS también establece los requisitos de compatibilidad con CLS; estos requisitos permiten determinar si el código administrado cumple la especificación CLS y hasta qué punto una herramienta dada admite la programación de código administrado que utilice las características de CLS.

11.2.1.7. Sistema de Componentes

Un programa .NET se forma a partir de un conjunto de ficheros de ensamblados (assembly) que se cargan de forma dinámica. Estos ficheros contienen la especificación de un módulo que puede estar formado por varias clases. Incluyen especificación de control de versiones, firmas criptográficas, etc.

Los ensamblados pueden ser estáticos o dinámicos. Los ensamblados estáticos pueden incluir tipos de .NET Framework (interfaces y clases), así como recursos para el ensamblado (mapas de bits, archivos JPEG, archivos de recursos, etc.). Los ensamblados estáticos se almacenan en el disco, en archivos PE.

También se puede utilizar .NET Framework para crear ensamblados dinámicos, que se ejecutan directamente desde la memoria y no se guardan en el disco antes de su ejecución. Los ensamblados dinámicos se pueden guardar en el disco una vez que se hayan ejecutado [Boc02]. En general, un ensamblado estático está formado por cuatro elementos (ver Figura 11.8):

- El manifiesto del ensamblado, que contiene los metadatos del ensamblado.
- Los metadatos de tipos.
- El código de lenguaje intermedio de Microsoft (MSIL) que implementa los tipos.
- Un conjunto de recursos.

Figura 11.8.Contenido de un ensamblado estático.

11.2.2. Lenguajes .Net

.NET Framework SDK incluye cuatro compiladores que admiten o han sido creados para la CLR. Estos lenguajes son:

Figura 11.9.Interación entre lenguajes y creación de assemblys.

11.2.2.1. Lenguaje Visual Basic

Desde Visual Basic 1.0, que simplificó completamente la escritura de aplicaciones para Windows, hasta Visual Basic 4.0, que ayudó a establecer COM2 como la arquitectura de objetos de Windows estándar, el lenguaje de Visual Basic ha sido la piedra angular de la plataforma Windows durante casi una década. Visual Basic .NET es un lenguaje de programación de alto nivel de .NET Framework y proporciona el punto de entrada más fácil a la plataforma Microsoft .NET [MVB]

11.2.2.2. Lenguaje C#

Es el lenguaje de programación diseñado para crear una amplia gama de aplicaciones empresariales que se ejecutan en .NET Framework [RAC+02]. Es una evolución de C y C++, es simple, moderno, garantiza la seguridad de tipos y está orientado a objetos. El código de C\# se compila como código administrado, lo que significa que se beneficia de los servicios de la CLR. Estos servicios incluyen la interoperabilidad entre lenguajes, la recolección de elementos no utilizados, mayor seguridad y mejor compatibilidad entre las versiones [MSC#].

11.2.2.3. Lenguaje C++ administrado

El lenguaje estándar de C++ se ha extendido, comenzando por Visual C++ .NET, con el fin de proporcionar compatibilidad con la programación administrada. Las Extensiones administradas de C++ se componen principalmente de un conjunto de palabras clave y atributos [MSC+].

11.2.2.4. Lenguaje Jscript .NET

Es un lenguaje de secuencias de comandos moderno con una gran variedad de aplicaciones. Es un auténtico lenguaje orientado a objetos y aún mantiene su espíritu de "secuencias de comandos". Jscript .NET mantiene compatibilidad total con las versiones anteriores de Jscript, a la vez que incorpora características nuevas y proporciona acceso a la CLR.

11.2.2.5. Lenguaje J#

Versión de Java desarrollada por Microsoft para la plataforma .NET. Tiene algunas extensiones para manejar el .NET framework \cite{vjsharp}.

La forma de utilizar estos lenguajes se muestra en la Figura 11.10, donde se muestra la secuencia final de escritura de aplicaciones.

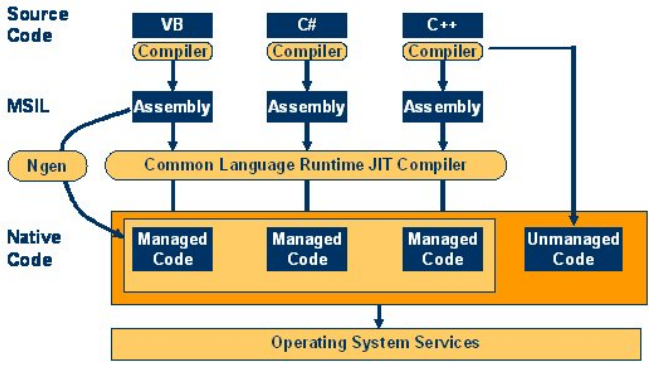


Figura 11.10.Utilización de los lenguajes de programación en la plataforma .NET.

Una de las características más potentes de los lenguajes .NET, que generan MSIL para la CLR, es que pueden interoperar entre ellos directamente sin necesidad de definir ningún tipo de IDL (Interface definición icrosof), gracias al uso de un sistema común de tipos (CTS) (Figura 11.11). Esto permite la reutilización de componentes escritos en diferentes lenguajes en una misma aplicación.

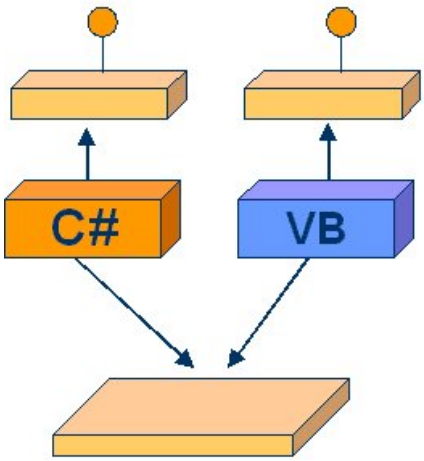


Figura 11.11.Interoperatividad entre lenguajes.

11.2.3. CTS: Sistema Común de Tipos

El sistema de común de tipos define cómo se declaran, utilizan y administran los tipos en el motor de tiempo de ejecución. Es también una parte importante de la compatibilidad del tiempo de ejecución con la integración entre lenguajes. El sistema de tipos común realiza las funciones siguientes:

- Establece un marco de trabajo que permite la integración entre lenguajes, la seguridad de tipos y la ejecución de código con alto rendimiento.
- Proporciona un modelo orientado a objetos que admite la implementación completa de muchos lenguajes de programación.
- Define reglas que deben seguir los lenguajes, lo que ayuda a garantizar que los objetos escritos en distintos lenguajes puedan interactuar unos con otros.

El sistema de tipos común es compatible con dos categorías generales de tipos, que a su vez se dividen en subcategorías:

11.2.3.1. Tipos de valor

Los tipos de valor contienen directamente sus datos y las instancias de los tipos de valor se asignan en la pila o se asignan en línea en una estructura. Los tipos de valor pueden ser integrados (implementados por el motor de tiempo de ejecución), definidos por el usuario o enumeraciones.

11.2.3.2. Tipos de referencia

Los tipos de referencia guardan una referencia a la dirección en memoria del valor y se asignan en el montón. Los tipos de referencia pueden ser tipos autodescriptivos, de puntero o de interfaz. El tipo de un tipo de referencia se puede determinar a partir de los valores de los tipos autodescriptivos. Los tipos autodescriptivos se dividen en matrices y tipos de clase. Los tipos de clase son clases definidas por el usuario, tipos de valor a los que se ha aplicado la conversión boxing y delegados.

Las variables que son tipos de valor tienen, cada una, su propia copia de los datos y, por lo tanto, las operaciones en una variable no afectan a las demás. Las variables que son tipos de referencia pueden hacer referencia al mismo objeto y, por lo tanto, las operaciones en una variable pueden afectar al mismo objeto al que hace referencia otra variable. Todos los tipos se derivan del tipo base System.Object.

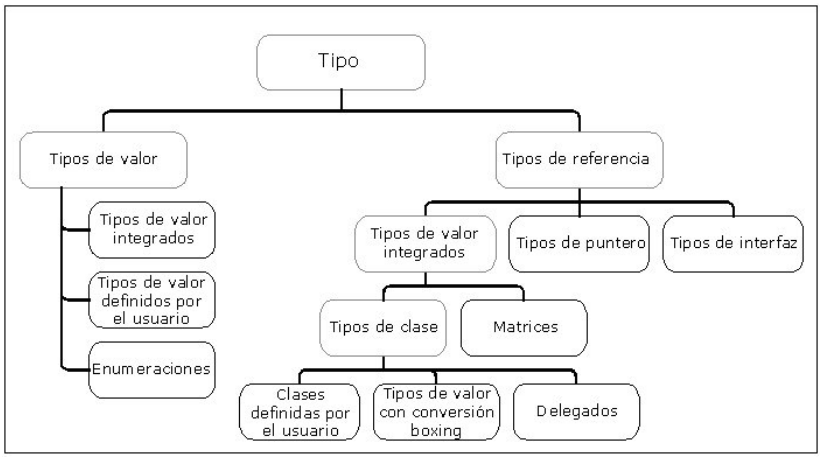


Figura 11.12.Sistema Común de Tipos.

En la Figura 11.12 se ilustra cómo se relacionan estos distintos tipos. Se debe tener en cuenta que las instancias de los tipos pueden ser simplemente tipos de valor o tipos autodescriptivos, aunque haya subcategorías de estos tipos.

Los valores son representaciones binarias de datos y los tipos proporcionan una forma de interpretar estos datos. Un tipo de valor se almacena directamente como una representación binaria de los datos del tipo. El valor de un tipo de referencia es la ubicación de la secuencia de bits que representa los datos del tipo.

Cada valor tiene un tipo exacto que define por completo la representación del valor y las operaciones definidas en el valor. Los valores de los tipos autodescriptivos se llaman objetos. Si bien siempre se puede determinar el tipo exacto de un objeto examinando su valor, ello no se puede hacer con tipo de valor o un tipo de puntero. Un valor puede tener más de un tipo. El valor de un tipo que implementa una interfaz es también un valor de ese tipo de interfaz. De la misma manera, el valor de un tipo derivado de un tipo base es también un valor de ese tipo base.

El motor de tiempo de ejecución utiliza ensamblados para ubicar y cargar tipos. El manifiesto del ensamblado contiene la información que el motor de tiempo de ejecución utiliza para resolver todas las referencias a tipos hechas dentro del ámbito del ensamblado.

Un nombre de tipo del motor de tiempo de ejecución tiene dos partes lógicas: el nombre del ensamblado y el nombre del tipo que se encuentra en el ensamblado. Dos tipos que tengan el mismo nombre pero estén en ensamblados distintos se definen como dos tipos diferentes.

Los ensamblados proporcionan coherencia entre el ámbito de los nombres que ve el programador y el que ve el sistema del motor de tiempo de ejecución. Los programadores escriben tipos en el contexto de un ensamblado. El contenido del ensamblado que está creando un programador establece el ámbito de los nombres que estarán disponibles en tiempo de ejecución.

Desde el punto de vista del motor de tiempo de ejecución, un espacio de nombres no es más que una colección de nombres de tipos. Algunos lenguajes pueden tener construcciones y la sintaxis correspondiente que ayudan a los programadores a formar grupos lógicos de tipos, pero el motor de tiempo de ejecución no utiliza estas construcciones al enlazar tipos. Así, las clases Object y String forman parte del espacio de nombres System, pero el motor de tiempo de ejecución sólo reconoce los nombres completos de cada tipo, que son System.Object y System.String respectivamente.

Se puede generar un único ensamblado que exponga tipos que parezcan proceder de dos espacios de nombres jerárquicos distintos, como System.Collections y System.Windows.Forms. También se pueden crear dos ensamblados que exporten tipos cuyos nombres contengan MyDll.MyClass.

Si se crea una herramienta para representar los tipos de un ensamblado como pertenecientes a un espacio de nombres jerárquico, la herramienta debe enumerar los tipos de un ensamblado o grupo de ensamblados y analizar los nombres de tipo para generar una relación jerárquica.

11.2.4. Biblioteca de clases de .Net Framework

.NET Framework incluye clases, interfaces y tipos de valor que aceleran y optimizan el proceso de desarrollo y proporcionan acceso a la funcionalidad del sistema. Para facilitar la interoperabilidad entre lenguajes, los tipos de .NET Framework cumplen la especificación de lenguaje común (CLS) y, por tanto, se pueden utilizar en todos los lenguajes de programación cuyo compilador satisfaga los requisitos de CLS.

Los tipos de .NET Framework son la base sobre la que se crean aplicaciones, componentes y controles. El Framework .NET incluye tipos que realizan las funciones siguientes:

- Representar tipos de datos base y excepciones.
- Encapsular estructuras de datos.
- Realizar E/S.
- Obtener acceso a información sobre tipos cargados.
- Invocar las comprobaciones de seguridad de .NET Framework.
- Proporcionar: acceso a datos, interfaz gráfica para el usuario (GUI) independiente de cliente e interfaz GUI de cliente controlada por el servidor.

.NET Framework proporciona un conjunto completo de interfaces, así como clases abstractas y concretas (no abstractas). Se pueden utilizar las clases concretas tal como están o, en muchos casos, derivar las clases propias de ellas, derivando una clase de una de las clases de .NET Framework que implementa la interfaz.

El espacio de nombres System es el espacio de nombres base para los tipos fundamentales de .NET Framework.

Este espacio de nombres incluye clases que representan los tipos de datos base que utilizan todas las aplicaciones: Object (base de la jerarquía de herencia), Byte, Char, Array, Int32, String, etc. Muchos de estos tipos se corresponden con los tipos de datos primitivos que utiliza el lenguaje de programación.

Además de los tipos de datos base, el espacio de nombres System (Figura 11.13) contiene casi 100 clases, que comprenden desde las clases que controlan excepciones hasta las clases que tratan conceptos básicos de tiempo de ejecución, como los dominios de aplicación y el recolector de elementos no utilizados.

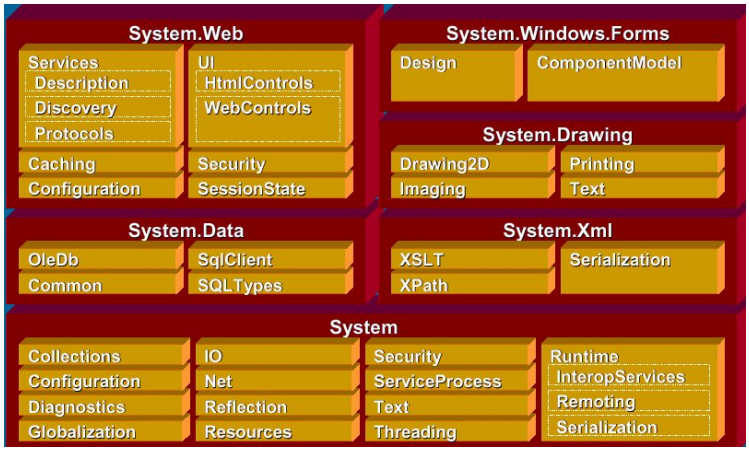


Figura 11.13.Componentes del espacio de nombres System.

11.2.5. Utilidades

.NET Framework proporciona una serie de herramientas o utilidades para el desarrollo de aplicaciones. Se pueden clasificar en cuatro categorías principales:

11.2.5.1. Herramientas de configuración e implementación

- Herramienta Visor de la caché de ensamblados (Shfusion.dll): Permite ver y manipular el contenido de la caché de ensamblados global mediante el Explorador de Windows.
- Herramienta Assembly Linker (Al.exe): Genera un archivo con un manifiesto del ensamblado a partir de uno o más archivos que son archivos de recursos o del Lenguaje intermedio de Microsoft (MSIL).

- Herramienta Registro de ensamblados (Regasm.exe): Lee los metadatos de un ensamblado y agrega las entradas necesarias al Registro, lo que permite a los clientes COM crear clases de .NET Framework de forma transparente.
- Visor de registro de enlaces de ensamblados (Fuslogvw.exe): Muestra detalles de los enlaces de ensamblado erróneos. Esta información ayuda a diagnosticar la causa por la que .NET Framework no puede encontrar un ensamblado en tiempo de ejecución.
- Herramienta Caché de ensamblados global (Gacutil.exe): Permite ver y manipular el contenido de la caché de ensamblados global y de la caché de descarga. Si bien Shfusion.dll proporciona una funcionalidad similar, puede utilizar Gacutil.exe desde secuencias de comandos de generación, archivos MAKE y archivos de proceso por lotes.
- Herramienta Installer (Installutil.exe): Permite instalar y desinstalar recursos de servidor mediante la ejecución de los componentes del instalador de un ensamblado especificado.
- Herramienta Almacenamiento aislado (Storeadm.exe): Enumera o quita todos los almacenes existentes para el usuario que ha iniciado sesión actualmente.
- Generador de imágenes nativas (Ngen.exe): Crea una imagen nativa a partir de un ensamblado administrado y la instala en la caché de imágenes nativa del equipo local.
- Herramienta Configuración de .NET Framework (Mscorcfg.msc): Proporciona una interfaz gráfica para administrar la directiva de seguridad de .NET Framework y las aplicaciones que utilizan servicios de interacción remota. Esta herramienta también le permite administrar y configurar ensamblados en la caché de ensamblados global.
- Herramienta Instalación de servicios de .NET (Regsvcs.exe): Agrega clases administradas a Servicios de componentes de Windows 2000; para ello, carga y registra el ensamblado y genera, registra e instala la biblioteca de tipos en una aplicación COM+ 1.0 existente.

- Herramienta Soapsuds (Soapsuds.exe): Ayuda a compilar aplicaciones cliente que se comunican con los servicios Web XML mediante una técnica denominada interacción remota.
- Exportador de la biblioteca de tipos (Tlbexp.exe): Genera una biblioteca de tipos a partir de un ensamblado de Common Language Runtime.
- Importador de la biblioteca de tipos (TlbImp.exe): Convierte las definiciones de tipos de una biblioteca de tipos COM en las definiciones equivalentes, en formato de metadatos administrado.
- Herramienta Lenguaje de descripción de servicios Web (Wsdl.exe): Genera código para servicios Web XML y clientes de servicios Web XML a partir de archivos de contrato WSDL (Lenguaje de descripción de servicios Web), archivos de esquemas XSD (Lenguaje de definición de esquemas XML) y documentos de descubrimiento .discomap.
- Herramienta Descubrimiento de servicios Web (Disco.exe): Descubre las direcciones URL de los servicios Web XML ubicados en un servidor Web y guarda los documentos relacionados con cada servicio Web XML en un disco local.
- Herramienta Definición de esquemas XML (Xsd.exe): Genera esquemas XML según el lenguaje XSD propuesto por World Wide Web Consortium (W3C). Esta herramienta genera clases DataSet y de Common Language Runtime a partir de un archivo de esquema XSD.

11.2.5.2. Herramientas de depuración

- Herramienta Microsoft Depurador de CLR (DbgCLR.exe): Proporciona servicios de depuración con una interfaz gráfica para ayudar a los programadores de aplicaciones a encontrar y resolver errores en programas en tiempo de ejecución.
- Depurador en tiempo de ejecución (Cordbg.exe): Proporciona servicios de depuración de la línea de comandos mediante la API Debug de Common Language Runtime. Se utiliza para encontrar y resolver errores en programas de tiempo de ejecución.

11.2.5.3. Herramientas de seguridad

- Herramienta Creación de certificados (Makecert.exe): Genera certificados X.509 sólo a efectos de pruebas.
- Herramienta Administrador de certificados (Certmgr.exe): Administra certificados, listas de certificados de confianza (certificate trust lists, CTL) y listas de revocaciones de certificados (certificate revocation lists, CRL).
- Herramienta Comprobación de certificados (Chktrust.exe): Comprueba la validez de un archivo firmado con un certificado X.509.
- Herramienta Directiva de seguridad de acceso a código (Caspol.exe): Permite examinar y modificar las directivas de seguridad de acceso al código en los ámbitos del equipo, el usuario y la organización.
- Herramienta Firma de archivos (Signcode.exe): Firma un archivo ejecutable portable (PE) con una firma digital Authenticode.
- Herramienta Vista de permisos (Permview.exe): Muestra los conjuntos de permisos mínimos, opcionales y rechazados solicitados por un ensamblado. También puede utilizar esta herramienta para ver toda la seguridad declarativa utilizada por un ensamblado.
- Herramienta PEVerify (Peverify.exe): Realiza comprobaciones de seguridad de tipos MSIL y de validación de metadatos en un ensamblado especificado.
- Herramienta Secutil (Secutil.exe): Extrae información de claves públicas de nombres seguros o certificados de compañía de software Authenticode de un ensamblado, en un formato que permite la incorporación al código.
- Herramienta Establecer Registro (Setreg.exe): Permite cambiar la configuración del Registro para las claves Software Publishing State de estado de edición de software, que controlan el comportamiento del proceso de comprobación de certificados.
- Herramienta Prueba de certificados de compañía de software (Cert2spc.exe): Crea un certificado de compañía de software (SPC) a partir de uno o varios certificados X.509, sólo a efectos de pruebas.

- Herramienta Nombre seguro (Sn.exe): Ayuda a crear ensamblados con nombres seguros. Sn.exe proporciona opciones para la administración de claves, así como para la generación y comprobación de firmas.

11.2.5.4. Herramientas generales

- Herramienta Minivolcados de Common Language Runtime (Mscordmp.exe): Crea un archivo que contiene información útil para analizar problemas del sistema en tiempo de ejecución. La herramienta Dr. Watson de Microsoft (Drwatson.exe) llama automáticamente a este programa.
- Herramienta Compilador de licencias (Lc.exe): Lee archivos de texto que contienen información sobre licencias y crea un archivo .licenses que se puede incrustar en un archivo ejecutable de Common Language Runtime.
- Generador de clases con establecimiento inflexible de tipos para administración (Mgmtclassgen.exe): Permite generar rápidamente una clase de enlace en tiempo de compilación en C\#, Visual Basic o Jscript para una clase especificada de Instrumental de administración de Windows (WMI).
- Ensamblador de MSIL (Ilasm.exe): Genera un archivo PE desde el Lenguaje intermedio de Microsoft (MSIL). Puede ejecutar el archivo ejecutable resultante, que contiene MSIL y los metadatos requeridos, para determinar si MSIL se comporta de acuerdo con lo esperado.
- Herramienta Desensamblador de MSIL (Ildasm.exe): A partir de un archivo PE que contiene código MSIL, crea un archivo de texto apropiado como entrada para el Ensamblador de MSIL (Ilasm.exe).
- Herramienta Generador de archivos de recursos (icros.exe): Convierte archivos de texto y .resx (formato de recursos basado en XML) en archivos .resources binarios de Common Language Runtime de .NET, que se pueden incrustar en un archivo ejecutable binario de tiempo de ejecución o compilar en ensamblados satélite.

- Importador de controles ActiveX de Windows Forms (Aximp.exe): Convierte definiciones de tipos de una biblioteca de tipos COM para un control ActiveX en un control de formularios Windows Forms.
- Visor de clases de Windows Forms (Wincv.exe): Busca clases administradas que coincidan con un modelo de búsqueda especificado y muestra información acerca de esas clases mediante la API de reflexión.
- Editor de recursos de Windows Forms (Winres.exe): Permite traducir rápida y fácilmente formularios Windows Forms.

12. TALISMAN: METODOLOGÍA ÁGIL CON MDA

El objetivo de esta Tesis es realizar una recomendación o metodología para el desarrollo de aplicaciones en el ámbito de los MDAs siguiendo los prefectos establecidos por el desarrollo ágil de Software [Bec01].

En este capítulo se divide en dos partes diferenciadas, una primera parte en la que se fundamenta la capacidad de los MDAs con metodologías ágiles frente a los métodos tradicionales y en una segunda parte se realiza la propuesta original de TALISMAN.

12.1. METODOLOGÍAS TRADICIONALES Y MDA

Una manera habitual de ilustrar el gran avance que supondría MDA en la la historia de la ingeniería del software es analizar su impacto en el proceso de desarrollo de software. Los autores [Fra03] [KWB03] [RFW+04] que defienden la necesidad de esta iniciativa, consideran que el desarrollo de herramientas MDA es un requisito indispensable para convertir el proceso de crear software una verdadera disciplina de ingeniería.

12.1.1. Proceso de Desarrollo Tradicional

12.1.1.1. La Iteración como Respuesta al Cambio

En la Figura 12.1 se representan las diferentes etapas generales que habitualmente se siguen a la hora de desarrollar una aplicación:

1. Un análisis de los *requisitos* de la aplicación.
2. La construcción de un *modelo de análisis* que, de una manera formal, especifique qué funcionalidad debe contener la aplicación.
3. El paso a un *modelo de diseño* que recoja con detalle los diferentes componentes del sistema informático que satisfaga los requisitos de análisis.
4. La escritura del *código fuente* que realice el modelo de diseño construido en una plataforma tecnológica determinada.
5. El *despliegue* del sistema operativo en un entorno de ejecución.

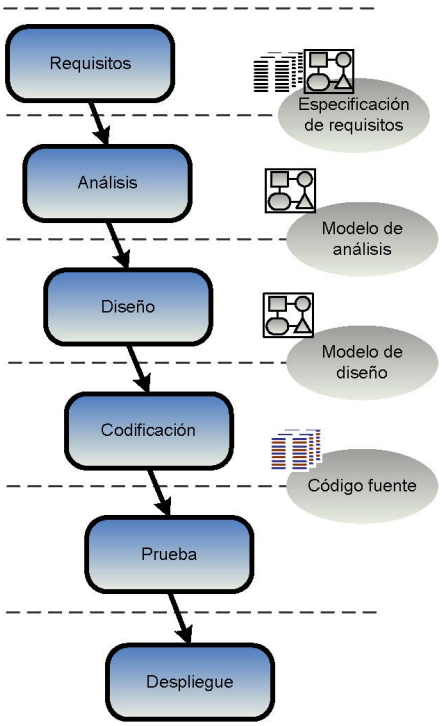


Figura 12.1.Etapas en el modelo de desarrollo de software tradicional.

Estas etapas generales son básicamente las mismas desde que fuesen formalizadas en el proceso de desarrollo propuesto por Royce en 1970 [Roy70] Desde entonces se han producido avances sustanciales en el ámbito de los procesos de desarrollo. Entre éstos, destacan los relativos a los *modelos de ciclos de vida* que sustentan los diferentes procesos.

El modelo de ciclo de vida propuesto por Royce se conoce en la actualidad como modelo de ciclo de vida en cascada (*waterfall model*). En dicho modelo las diferentes etapas de desarrollo se suceden en secuencia, con una iteración nula entre etapas o reducida a la etapa inmediatamente anterior. Este modelo de ciclo de vida se dio por fracasado, entre otras razones, por su dificultad para afrontar los cambios que se produjesen durante el proceso de desarrollo, en especial los cambios en los requisitos.

Debe señalarse que Royce en su propuesta [Roy70] sí contempló la necesidad de iteración entre etapas y el hecho de que esta iteración no necesariamente debe producirse entre etapas consecutivas.

Royce consideró que los cambios en las fases de análisis y codificación tenían un impacto mínimo en el resto de etapas. La etapa fundamental sería la de diseño, y en base a ella, propuso una serie de recomendaciones para afrontar los cambios que se produjesen en el análisis de requerimientos y en el diseño. De una manera un tanto injusta, su propuesta suele recordarse en la actualidad únicamente por los aspectos negativos del modelo de ciclo de vida en cascada y no cómo uno de los primeros intentos de formalización de un proceso de desarrollo en la historia de la ingeniería del software.

Tras el modelo de ciclo de vida en cascada, se sucedieron muchos otros. Entre ellos, el modelo en espiral de Boehm [Boe88], el evolucionario [MZ96] y el modelo incremental [Som01].

Todos estos modelos tienen en común una apuesta por la iteración cómo herramienta principal para abordar los cambios en el software. Se abandona la idea de poder desarrollar completamente cada etapa en una sola iteración. En especial, se acepta que los cambios en los requisitos del software son inevitables [Bec99a], por los que el proceso de desarrollo debe estar preparado para absorberlos (Figura 12.2).

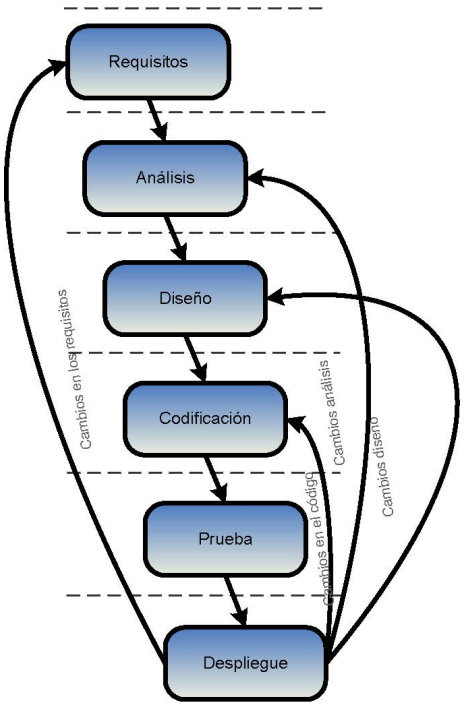


Figura 12.2.Proceso de desarrollo con un ciclo de vida iterativo.

Uno de los métodos iterativos de desarrollo orientado a objetos más populares es el *Unified Proccess* (UP) o Proceso Unificado [JBR99]. La implementación más adoptada de este método es el *Racional Unified Proccess* (RUP) [Kru00].

12.1.1.2. Métodos Ágiles

Tras adoptar los métodos de desarrollo un modelo de ciclo de vida iterativo, a la hora de su puesta en práctica se detectó otro problema, relacionado con los artefactos producidos y consumidos entre las diferentes etapas. En las fases previas a la codificación se producen una gran cantidad de artefactos en forma básicamente de documentos de texto y diagramas, pero estos están desconectados entre sí y del código fuente que representan [KWB03].

Conforme los sistemas van siendo modificados, la distancia existente entre el código y los artefactos de modelado previamente construidos se incrementa. Ante esta situación se presentan dos alternativas opuestas:

- En primer lugar, puede retrocederse a la fase de desarrollo apropiada y acometer el cambio. El problema de esta aproximación son los cambios en cascada que se producen en las fases subsiguientes y en los artefactos producidos en cada una de ellas. Es lo que se ha denominado un "exceso de burocracia" de algunos procesos de desarrollo [Fow05].
- En segundo lugar, puede optarse por realizar los cambios únicamente en el código. Esta es una opción a menudo adoptada en la práctica, cuando no se dispone de tiempo para modificar los artefactos de modelado. Como consecuencia, los productos obtenidos de las primeras fases de desarrollo van progresivamente perdiendo valor. Dejan de representar una especificación exacta del código. Las tareas de análisis, modelado y documentación previas a la codificación dejan de considerarse imprescindibles. Sin embargo, tras décadas de experiencia en el desarrollo de software se acepta que, ante proyectos de cierta envergadura, dichas tareas deben realizarse [KWB03].

Desde mediados de los 90 se produjo un movimiento en el campo de la ingeniería del software que intentó en parte mitigar los problemas mencionados: la aparición de los "métodos ágiles" (*agile methods*) englobados bajo lo que se denominó "desarrollo de software ágil" (*agile software development)* [Lar03]

Suele decirse que la aparición de los métodos ágiles fue provocada en parte como reacción a los "métodos pesados" (*heavyweight methods)*. Éstos comprenden los procesos de desarrollo rígidos, fuertemente regulados y que realizan un gran énfasis en la planificación, fuertemente influenciados por otras disciplinas de ingeniería [Fow05]. En ocasiones también se denomina a estos métodos "métodos predecibles" (*predictive methods*) [Lar04] por su motivación de intentar planificar con gran detalle largos plazos del proceso de desarrollo. Frente a este enfoque, los métodos iterativos en general, y los métodos ágiles en particular, también se denominan"métodos adaptativos". En vez de intentar evitar el cambio, prevén aceptarlo cuándo se produzca, adaptando el software y el proceso en sí mismo para ello.

Los diferentes métodos ágiles comparten una serie de características comunes [Lar04]:

- Acortan el plazo de iteración y apuestan por un desarrollo evolutivo. La forma final del programa será el resultado de sucesivas modificaciones a partir de un entregable inicial.
- Utilizan una planificación adaptativa. La propia planificación del proceso se modifica conforme se avanza en el desarrollo, para adaptarse a las circunstancias que en cada momento concurran.
- Apuestan por la entrega incremental. Las nuevas características y modificaciones sobre el sistema se integran cuanto antes en el sistema entregable.
- Incorporan diversas prácticas de desarrollo que buscan permitir una respuesta rápida y flexible a los cambios.

El método ágil más conocido es *Extreme Programming* (XP) [Bec99a]. Como ejemplo de métodos ágiles anteriores a XP pueden citarse Scrum [SB01] y la familia de métodos Cristal [Coc04]. Con respecto al UP, el hecho de considerarlo pesado o ágil dependerá de su implantación concreta, pues realmente se trata de un framework de procesos que debe adaptarse a cada organización y proyecto en concreto. Como señala Martin Fowler en [Fow05] en la industria pueden encontrarse utilizaciones del UP que van desde rígidos modelos en cascada a instancias perfectamente ágiles.

Los métodos ágiles también han sido objetivo de múltiples críticas habiéndose creado cierta controversia en torno a ellos. Si bien existe un consenso generalizado acerca de la idoneidad de las buenas prácticas sobre las que descansan, se han señalado algunos problemas con los mismos, en especial con XP.

Entre ellos pueden destacarse: la faltan de datos empíricos que prueben su efectividad y su no escalabilidad a la hora de su aplicación en proyectos grandes [TFR02], la excesiva dependencia del conocimiento tácito de los miembros del equipo de desarrollo debido a la ausencia de una documentación formal [Coc01], o el alto acoplamiento entre las diferentes prácticas de XP y la falta de documentación de sus interrelaciones [Van05].

Un error habitual es caracterizar XP como un método anárquico donde lo único importante es el código de los programas. Así, en [KWB03] se llega a afirmar en relación a XP:

> *One of the reasons for this is that it acknowledges the fact that the code is the driving force of software development. The only phases in the development process that are really productive are coding and testing.*

Realmente, para XP la actividad más importante es la de diseño. En el "Manifiesto Ágil" (*Agile Manifesto)* [Bec01], en el principio número 10 se señala que la excelencia técnica y los buenos diseños aumentan la agilidad. Lo que sucede es que los métodos ágiles consideran la dificultad de transformar los diseños en código y proponen diversas prácticas para que ésta transformación sea eficaz.

12.1.1.3. Modelado y Codificación

Un error que ha estado presente en multitud de enfoques metodológicos de desarrollo software ha sido obviar la importancia de la actividad de codificación. En una aproximación a otras disciplinas de ingeniería asentadas, se equiparó la codificación a la fase de construcción de cualquier otra ingeniería. Así, Winston W. Royce afirmaba en 1970 [Roy70], con relación a la fase de codificación:

> *(...) generally these phases are managed with relative ease and have little impact on requirements, design, and testing.*
>
> *(...) If in the execution of their difficult and complex work the analysts have made a mistake, the correction is invariably implemented by a minor change in the code with no disruptive feedback into the other development bases.*

Martin Fowler profundiza en este aspecto [Fow05] analizando las diferencias de los procesos de desarrollo de software con los procesos de construcción de otras ingenierías. En éstas, la actividad de diseño es la más difícil de predecir y requiere la gente con mayor formación y creatividad. La actividad de construcción es más fácil de predecir, requiere gente con menos formación y es, en general, mucho más costosa en términos económicos que el diseño. Una vez que se tiene el diseño, puede planificarse la construcción y ejecutarse ésta de una manera predecible.

Trasladar este modelo al desarrollo de software significaría volver al modelo en cascada, cuyos inconvenientes ya se han mencionado. Martin Fowler añade a estos inconvenientes una interesante pregunta [Fow05]: *¿Puede obtenerse un diseño que haga de la codificación una actividad de construcción predecible?. Y si esto es posible ¿será rentable en términos de costes?*. Esta cuestión ya no hace referencia al intento de capturar todos los requisitos en una iteración, sino a la dificultad de transformar los modelos en código.

Entre las razones de esta dificultad pueden señalarse la inmadurez de los lenguajes y métodos de modelado software, la imposibilidad de probar los modelos, la impedancia que existe a la hora de codificar un modelo software con los lenguajes de programación existentes o las dificultades que añade a la codificación la necesidad de considerar los aspectos tecnológicos de la plataforma final de ejecución.

Estas dificultades han llevado a algunos autores a considerar el código fuente como parte de la especificación del software. Bajo este enfoque, Jack Reeves propone [Ree92] considerar el código fuente como un documento más de diseño y que realmente la fase de construcción del software la realizan el compilador y el enlazador (*linker*).

En este sentido, Steve Cook afirma [Coo04] que la única forma de describir aplicaciones hasta la fecha ha sido el código fuente. Esto ha impedido hacer que el desarrollo de software puede ser implementado como una cadena de producción, donde cada participante acepta una entrada en forma de activos de información, añade sus conocimientos y pasa los resultados a los siguientes participantes en la cadena. Como resultado, afirma que las únicas personas que están profundamente involucradas en la producción de una aplicación son los programadores.

Las anteriores reflexiones reflejan un problema fundamental en el modelado de software que afecta sustancialmente a los procesos de desarrollo. Este problema es la gran distancia existente entre:

- Los modelos software, es decir, las descripciones del software realizadas a un elevado nivel de abstracción.
- El código fuente que comprende la aplicación y que, en última instancia, realiza los modelos desarrollados sobre una plataforma tecnológica concreta.

Ante este problema, un enfoque pragmático y con gran aceptación en la actualidad es el de los métodos ágiles. Éstos le dan una gran importancia al modelado, pero entendido como una herramienta de comunicación, no de documentación [Lar04]. Bajo este enfoque, no se construyen modelos detallados del software ni tampoco se modelan todas las partes de éste. Los problemas más sencillos de diseño se resolverán directamente en la fase de codificación. Los modelos se centran en especificar las partes más complejas o conflictivas de la aplicación.

Esta filosofía de modelado fue bautizada como "modelado ágil" (*agile modelling)* en 2002 [AJ02], aunque ya había sido puesta en práctica por los autores del movimiento ágil desde mucho antes. Entre los autores que defienden este enfoque se encuentran destacados ingenieros como Martin Fowler [Fow03], Craig Larman [Lar04] y Kent Beck [Bec99a].

Otro enfoque distinto al problema es apostar por la generación automática de código a partir de los modelos que especifican el software. MDA utiliza este planteamiento.

12.1.1.4. Problemas en los Procesos de Desarrollo Actuales

Tal y como se ha visto en el apartado anterior, la desconexión entre los artefactos de modelado, y entre éstos y el código fuente del programa, hace que el coste de un modelado formal, detallado e integral dificulte e incluso desaconseje su realización.

Pero el no realizar un modelado formal plantea un importante problema de **mantenimiento**. Si no se dispone de una documentación formal, se produce inexorablemente una dependencia del conocimiento tácito del equipo de desarrollo. Éste conocimiento se define como la suma del conocimiento de todas las personas del equipo que desarrolla el proyecto [Coc01].

Una vez superado el periodo de desarrollo principal del software, es habitual que los equipos de desarrollo reduzcan su tamaño. Con lo que el conocimiento tácito se muestra insuficiente para el mantenimiento del software, para solucionar este problema en el caso de XP, Cockburn propone planificar el desarrollo de documentación utilizando "*story cards*" como si fuesen nuevas características del software a desarrollar. Por otra parte, es habitual que el cliente exija una documentación formal que describa el diseño del sistema, como mecanismo de protección contra la desaparición precisamente del conocimiento tácito del equipo de desarrollo.

Por otra parte, puede señalarse una serie de problemas comunes a los métodos de desarrollo actuales. Centrándonos en los métodos con un ciclo de vida iterativo, que son los que gozan de una mayor aceptación en la actualidad, se detectan problemas de:

12.1.1.4.1. Productividad

Existe una gran redundancia provocada por el mantenimiento manual de los entregables del proceso [RFW+04]. Cada elemento del modelo de análisis aparece de nuevo en el modelo de diseño y cada elemento del modelo de diseño aparece de nuevo en el código fuente. Con unos requisitos cambiando constantemente, esta redundancia puede entorpecer enormemente la transición entre fases y, con ello, el proceso de desarrollo.

Con respecto a la transición entre las etapas de análisis y diseño, para distinguir ambas fases a menudo se dice que el análisis representa el "qué" y el diseño el "cómo" [Boo93]. También que el análisis trata sobre hacer lo correcto y el diseño con hacerlo correctamente [Lar04]. Se acepta que:

- El *análisis* hace énfasis en la investigación del problema y de sus requisitos, en lugar de en su solución.
- El *diseño* hace énfasis en una solución conceptual, a nivel tanto de software como de hardware, que cumple todos los requerimientos. No entra en su ámbito la implementación de la solución.

Sin embargo, en la práctica, la frontera entre lo que se considera análisis y diseño es difusa. De hecho, los métodos de desarrollo actuales apuestan por una transición gradual del análisis al diseño. En este sentido, una de las principales ventajas del análisis y diseño orientado a objetos con respecto al paradigma estructurado, es que se manejan los mismos artefactos de modelado en ambas fases, lo que facilitaba enormemente la transición. Sin embargo este enfoque presenta inconvenientes:

- ¿Cuándo debe dejarse de analizar? Dado que el modelo de análisis no contiene ningún aspecto de la implementación no puede ejecutarse y por lo tanto, tampoco puede probarse formalmente su corrección. El criterio de completud y corrección de los modelos de análisis dependerá por lo tanto de la opinión de los analistas. La única manera de verificar que el modelo de análisis está bien construido, es realizar todo el proceso de transformación y realizar baterías de pruebas sobre el programa final una vez desarrollado.
- Cuando se introducen en el diseño los aspectos tecnológicos de la solución, se obtiene una mezcla de los aspectos relacionados con el problema a resolver con los aspectos relacionados con una solución concreta al mismo [RFW+04]. Estos aspectos, que corresponden a perspectivas totalmente distintas del proceso de desarrollo, se entremezclan manualmente y rápidamente se vuelven inseparables. Esto dificulta tanto abordar en el diseño los cambios que se produzcan en el análisis, así como cambiar aspectos tecnológicos en el diseño.

12.1.1.4.2. Calidad

Cada componente del diseño está sujeto a su propio proceso de transformación manual para convertirse en parte de la aplicación. Por ello, la calidad del código fuente resultante es directamente proporcional a la calidad del programador que realiza la transformación [RFW+04]. Por esto, los métodos ágiles hacen hincapié en la importancia del talento de las personas que realizan el desarrollo, entendiendo la codificación como un aspecto clave de éste [Fow05].

La herramienta más básica de la que disponen los desarrolladores para verificar la calidad del software, entendida ésta como el grado en éste cumple con los requisitos [ISO00], es la realización de pruebas (*testing*) [MB+04]. En los procesos de desarrollo tradicional, el único artefacto sobre el que pueden realizarse pruebas para verificar su corrección es el código fuente. Este código refleja los resultados de todas las fases de desarrollo anteriores: especificación de requisitos, modelo de análisis y modelo de diseño. Esto tiene importantes consecuencias:

- Para verificar cualquiera de los modelos construidos durante el desarrollo de un programa deben acometerse todas las etapas hasta su codificación.
- Cuando se detecta un fallo en las pruebas éste puede deberse a errores de codificación, tecnológicos, de diseño o de análisis. El procedimiento de detección y corrección de errores se entorpece.

Un enfoque interesante para el *testing* es el planteado por XP [Bec99a]. Una práctica fundamental que sustenta esta metodología es la construcción de test unitarios para cada unidad funcional (módulo). Los módulos se integran rápidamente en la aplicación entregable y su codificación termina cuando todos los test creados se ejecutan correctamente. Cada vez que se acomete un cambio en el software en el validan los test, modificándose si son requeridos. Al conjunto de técnicas centradas en la práctica de escribir primero un caso de prueba (*test case)* e implementar posteriormente únicamente el código necesario para pasar la prueba se denominó posteriormente *Test-Driven Developer* (TDD) [Bec02]. Su principal objetivo es cumplir los requisitos manteniendo el código en su estado más simple posible.

Se trata, como es habitual con los métodos ágiles, de un acuerdo pragmático. Si lo único que puede probarse es el código fuente, debe darse a su verificación la importancia que se merece. Ésta debe ser una parte fundamental del proceso de desarrollo. Se busca de este modo incrementar la eficiencia de desarrollo, pero no representa una solución completa a los problemas planteados anteriormente.

12.1.1.4.3. Trazabilidad de requisitos.

Se entiende por trazabilidad de los requisitos como la posibilidad de describir y realizar un seguimiento del ciclo de vida de un requisito durante el proceso de desarrollo [GF04]. Por ejemplo, desde que se descubre en el análisis de requisitos, a través de las fases de desarrollo hasta su codificación y despliegue.

La trazabilidad de requisitos es una herramienta fundamental de otras disciplinas dentro de la ingeniería del software, como por ejemplo la *gestión de la calidad* o el seguimiento (*tracking*) en la *gestión de proyectos*. Su realización se dificulta enormemente por la desconexión entre los artefactos de modelado y el código. Si ya es difícil descubrir el diseño que origina un fragmento de código fuente [Fow05], más lo es descubrir el modelo de análisis y mucho más la especificación de requisitos que lo originó.

Las metodologías con un modelo de ciclo de vida iterativo añaden una complejidad adicional a este proceso, puesto que deben considerarse todos los períodos de refinamiento e iteración en todas estas fases [Ste04].

12.1.1.4.4. Reutilización.

En la búsqueda de evitar la duplicación de esfuerzos se han investigado multitud de técnicas y paradigmas que favorezcan la reutilización de artefactos durante las diferentes etapas de desarrollo. Así, pueden citarse:

- la programación orientada a objetos [Boo93]
- la tecnología de componentes [HC01]
- los patrones arquitectónicos [BMR+96
- los patrones de diseño [GHJ+05]
- los patrones de análisis [fow96]
- la programación orientada a aspectos [KLM+97]

Aunque se han desarrollado avances significativos en este campo, aun se está lejos de conseguir una reutilización plena de los artefactos construidos. En [RFW+04] se señala que la reutilización únicamente es efectiva en la fase de análisis, puesto que es la única que no considera detalles de implementación. En las fases de diseño, y especialmente en la codificación, los detalles tecnológicos de la plataforma de ejecución impregnan los artefactos construidos, dificultando su reutilización.

Las características de mantenimiento, reutilización, calidad, productividad, trazabilidad, seguimiento son una cuestión recurrente en las taxonomías de objetivos a conseguir para un verdadero proceso de ingeniería del software [Tyr01].

Sin embargo, aunque se han realizado grandes avances para su consecución, siguen existiendo dificultades que impiden su plena consecución y las soluciones planteadas son parciales.

12.1.2. Proceso de desarrollo utilizando MDA

El enfoque de MDA supone importantes cambios en cómo se acometen las fases en el proceso de desarrollo. En MDA se habla de una doble transformación automática:

- entre un PIM y un PSM
- entre un PSM y el código fuente de la aplicación.

Este enfoque implica, en primer lugar, cambios sustanciales en las fases de análisis y diseño.

El desarrollo del PIM se corresponderá casi en su totalidad con la fase de análisis en el desarrollo tradicional. Se desarrollará un modelo formal que especifique qué debe realizar la aplicación. Este modelo estará libre de detalles de implementación, pero será extremadamente detallado. Por ejemplo, en el caso de UP, implicarían los sucesivos refinamientos que llevan a satisfacer cada caso de uso: descubrimiento de clases, la asignación de responsabilidades, la creación de objetos, el intercambio de mensajes, etc. Estos aspectos forman parte del diseño, o de la transición del análisis al diseño, en los procesos de desarrollo tradicional, pero en MDA no se contemplan ninguno de los aspectos relativos a la plataforma de ejecución.

El conocimiento experto acerca de la plataforma tecnológica donde se ejecutará la aplicación seguirá siendo necesario y vendrá dado por la definición de la transformación. Pero ahora esta información estará separada de la especificación de la aplicación. Se favorecerá la reutilización de los mismos modelos con diferentes definiciones de transformación, y la utilización de las mismas definiciones de transformación con diferentes modelos.

La doble transformación automática PIM-PSM y PSM-código fuente hace que, desde el punto de vista del desarrollador, los modelos se perciban como ejecutables. Esto permitirá validar los modelos realizados. Además permitirá determinar cuándo se ha terminado de modelar: cuando el PIM verifique todos los casos de pruebas diseñados [RFW+04]. Es análogo a la regla de XP: el desarrollo de un módulo finaliza cuando su código fuente pasa todos los test [Bec99a].

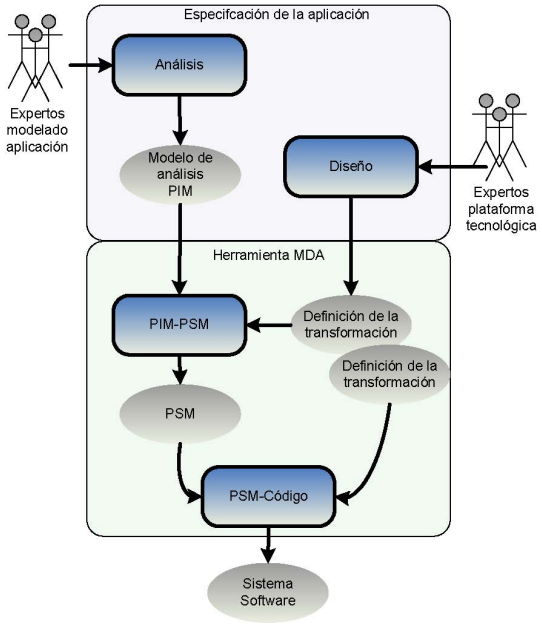


Figura 12.3. Especificación e implementación de aplicaciones con MDA.

Este proceso se representa en la Figura 12.3. MDA haría efectiva la separación de la especificación del modelo de análisis del software y de su diseño, entendido como la realización del análisis en una plataforma concreta. Esto permitiría que ambas actividades fuesen realizadas por grupos de especialistas diferenciados. Los analistas expertos en modelado de software no tienen porqué ser expertos en plataformas tecnológicas concretas y viceversa. En [RFW+04] incluso se propone considerar diseño de la aplicación como uno asunto más a tratar en el análisis.

Tal y como se muestra en la Figura 12.3, la herramienta MDA aceptaría como entradas:

- La especificación de lo que debe realizar la aplicación, dada por el modelo de análisis (PIM).
- La especificación de cómo implementar el modelo de análisis sobre una plataforma concreta, dada por la definición de la transformación.

A partir de estas entradas, se generaría el PSM que podría ser entonces transformado en el código fuente de la aplicación. Ambas transformaciones serían automáticas.

Con respecto al ciclo de vida utilizado, el proceso de desarrollo utilizando MDA no sería muy diferente del proceso de desarrollo tradicional. Los requisitos del software cambian continuamente y la iteración es la manera más eficaz de afrontar el cambio, al igual que sucede en el proceso de desarrollo tradicional.

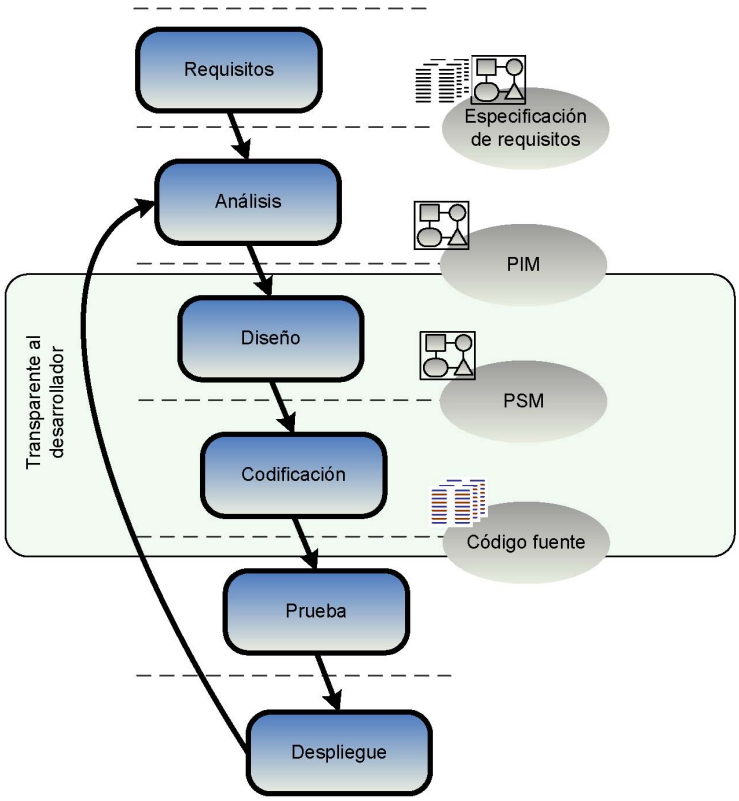


Figura 12.3. Proceso de desarrollo iterativo con MDA.

No obstante, MDA permite hacer que la iteración sea más efectiva. Durante el desarrollo, pueden producirse cambios en los requisitos tecnológicos de la aplicación. Esto implicaría un ajuste en las definiciones de las transformaciones a realizar, reutilizables para la realización de diferentes aplicaciones. Más habituales serán los cambios en los requisitos de negocio. MDA permite absorber estos cambios realizando las modificaciones pertinentes en el modelo de análisis. Al contrario de lo que ocurría con el proceso de desarrollo tradicional, la propagación de estos cambios durante la realización del modelo de análisis son transparentes al desarrollador.

Por otra parte, debido a que la transformación desde el PIM hasta el código fuente de la aplicación es ejecutada automáticamente por la herramienta MDA, el desarrollador podrá validar el funcionamiento de los modelos de análisis ejecutando los PIM construidos.

12.2. ADAPTACIÓN DE LOS MDAS A LOS MÉTODOS DE DESARROLLO ÁGIL DE SOFTWARE: TALISMAN

Los métodos ágiles de desarrollo constituyen por sí mismos una respuesta a la realidad cambiante. Pero, como métodos que son, lo hacen fundamentalmente desde la perspectiva de la gestión de proyectos y, en todo caso, del análisis y el diseño.

El proceso de desarrollo se caracteriza por ser:

- Un proceso iterativo, incremental y prototipado.
- Ágil en base a la XP (programación extrema), que es una nueva disciplina de desarrollo de software creada por Kent Beck, basada en la simplicidad, comunicación, retroalimentación y la refactorización de código.
- Dirigido por modelos.
- Un proceso AMDD (Agile Model Driven Development).

En la Figura 12.4. se muestra el proceso desarrollo de Software con TALISMAN.

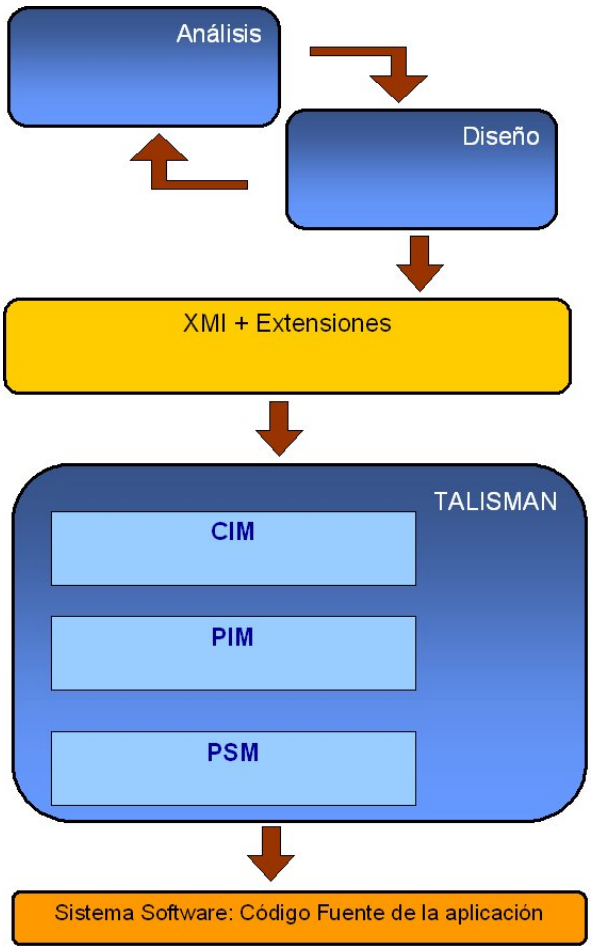


Figura 12.4. Esquema del proceso de desarrollo ágil con TALISMAN

12.2.1. Análisis

El análisis se basa en Casos de Uso que se especifican completamente por medio de Escenarios.

Los Escenarios contienen: los actores, los objetos que intervienen, las precondiciones, las postcondiciones, las excepciones y el proceso (guión ó *story board*) donde se especifica lo que ocurre en cada Escenario.

El proceso o guión de cada Escenario debe ser especificado completamente dado que tiene una correspondencia directa con Diagramas de Secuencia y para los Diagramas de Actividades de la fase de Diseño.

Los Prototipos, ligados a cada Escenario, muestran interfaces de usuario o especificaciones funcionales o de rendimiento de la aplicación. Estos Prototipos tienen como objetivo validar los requisitos de la aplicación. Mientras los Prototipos no sean validados se itera en la fase de Análisis.

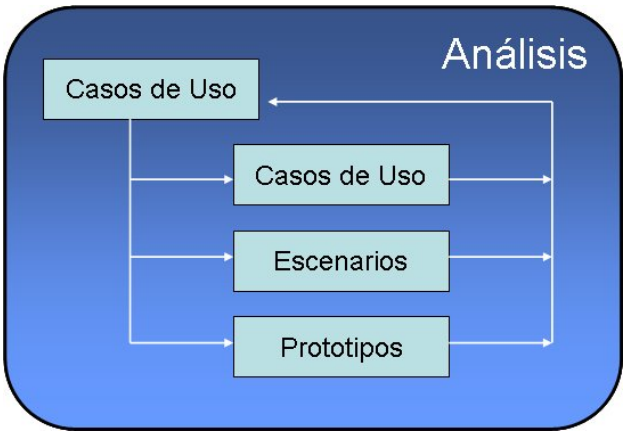


Figura 12.5.Detalle del Análisis en TALISMAN

12.2.2. Diseño

A partir de los escenarios se obtienen las especificaciones estáticas y dinámicas. Los diagramas de estados deber ser extendidos con un lenguaje basado en OCL y otras especificaciones en un lenguaje orientado a objetos neutral que facilite la generación en diversos lenguajes (Java, C#,…).

En el diseño existen varias partes:

- Obtención del Diccionario de Clases: contiene el nombre, atributos y métodos de cada clase.
- Especificación del comportamiento interno de los métodos de cada clase.
- Especificación de la arquitectura software.
- Especificación de la arquitectura hardware donde se instala la arquitectura software.

Los Diagramas de Secuencia y Colaboración, se construyen a partir de las especificaciones de los Escenarios del Análisis, y tienen por objeto construir el diccionario de clases.

El diccionario de clases es la base para la construcción del Diagrama de Clases. Este Diagrama de Clases se enriquecerá con nuevas clases abstractas, interfaces, herencias, agregaciones y se incorporarán patrones de diseño. Esto obliga a reformar los Diagramas de Colaboración y por tanto al refinamiento del diccionario de clases. Este proceso se realizará iterativamente hasta que se obtenga un diccionario de clases estable.

Cuando el diccionario de clases sea estable (que se producen mínimos cambios en cada iteración), se realiza la especificación del comportamiento interno de los métodos de cada clase, a través de los Diagramas de Actividades.

Los Diagramas de Actividades se basan en las especificaciones de detalle de los Escenarios del Análisis. Estos diagramas contienen un nivel de detalle mayor que los diagramas de Secuencia y Colaboración. También tienen referencias a la creación, uso y estado de objetos.

Cada Clase tiene un Diagrama de Estados, que muestra todos los estados por los que puede transitar un objeto en su ciclo de vida. Cada transicción entre estados se realiza a través de llamadas a métodos.

Los Perfiles son un mecanismo que sirven para definir y representar los conceptos de un dominio específico de aplicación.

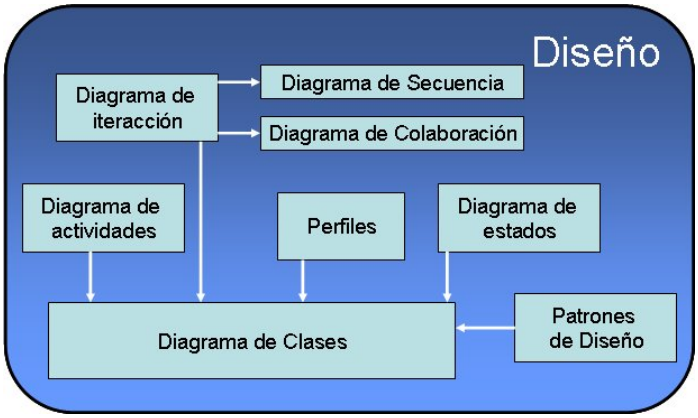


Figura 12.6.Detalle del Análisis en TALISMAN

12.2.3. XMI + Extensiones

Con toda la información del Análisis y el Diseño se genera un archivo XML. Este archivo contiene XMI más extensiones.

Las extensiones serán necesarias para la semántica de la que carece UML y además se necesita especificar el comportamiento de los métodos (algorítmica de los métodos) también la información de los perfiles y la implementación de Patrones de Diseño.

12.2.4. TALISMAN

TALISMAN utiliza el archivo XML (XMI + Extensiones) para generar código realizando la validación semántica, construye el sistema de persistencia con ayuda de motores de persistencia, los interfaces de usuario con ayuda de herramientas generadoras de interfaces y genera el código por medio de transformaciones.

La arquitectura de TALISMAN se muestra en la Figura 12.7.

```
<class name="Casa" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="nombreCasa" visibility="public" type="string"
initial="&quot;NOMBRE_DE_LA_CASA&quot;" changeability="changeable" isStatic="true"
/>
        <property name="direccion" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="telefono" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="precioDiaFest" visibility="public" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="precioDiaLab" visibility="public" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="comoLlegar" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="estado" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="tipoCasa" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="capacidad" visibility="protected" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="foto" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="logo" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="personaC" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="cuentaPersonaC" visibility="protected" type="string"
initial="" changeability="changeable" isStatic="false" />
        <property name="tlfPersonaC" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="ActPropia" multiplicity="multiple" />
        <association target="Zona" multiplicity="1" dependent="false" />
        <association target="Reserva" multiplicity="multiple" />
        <association target="Habitacion" multiplicity="multiple" />
      </associations>
      <methods>
        <method name="UnMetodo" type="normal" visibility="public"
returnType="string" isAbstract="false">
          <parameters>
            <param name="param1" type="string" />
            <param name="param2" type="int" />
          </parameters>
        </method>
      </methods>
    </class>
```

Código Fuente 12.1.Ejemplo de código XMI de TALISMAN

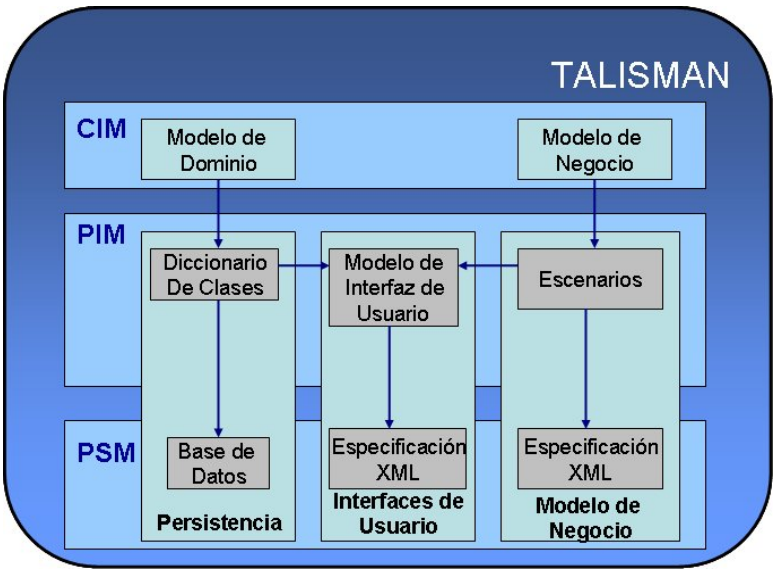


Figura 12.7.Detalle de la arquitectura de TALISMAN

La Arquitectura de TALISMAN posee una serie de modelos:

- CIM
- PIM
- PSM

TALISMAN también posee unas guías de transformación entre modelos:

- PIM a PIM
- PIM a PSM
- PSM a PSM

12.2.4.1. Persistencia

En las aplicaciones modernas el uso de un motor de persistencia se está convirtiendo en una práctica casi obligatoria por varias razones:

- Es una idea muy madura y utilizada.
- Hoy en día el código de las aplicaciones se programa utilizando un modelo orientado a objetos pero sin embargo las bases de datos que se utilizan con frecuencia no siguen este modelo, siguen un modelo relacional. El problema es que si toda la aplicación siguiera el modelo orientado a objetos nos encontraríamos con bases de datos aún muy inmaduras y poco estandarizadas y si toda la aplicación siguiera el modelo relacional se perderían todas las ventajas que ofrece la

orientación a objetos (flexibilidad, mantenimiento, reusabilidad, etc.,…). El motor de persistencia hace de traductor entre objetos y registros y entre registros y objetos, de esta forma para el código del programa todo son objetos y para la base de datos todo son registros.

- El código se abstrae completamente del tipo de base de datos utilizando, permitiendo así el cambio de ubicación de la información sin apenas modificaciones en el código.
- Se puede utilizar el mismo motor de persistencia en todas las aplicaciones con lo que sólo se debe programar una vez.
- Se genera aproximadamente un 40 % menos de código siendo éste mucho más mantenible, robusto y sencillo.

Ofrece optimizaciones de rendimiento que de no ser por el motor de persistencia serían difíciles de conseguir.

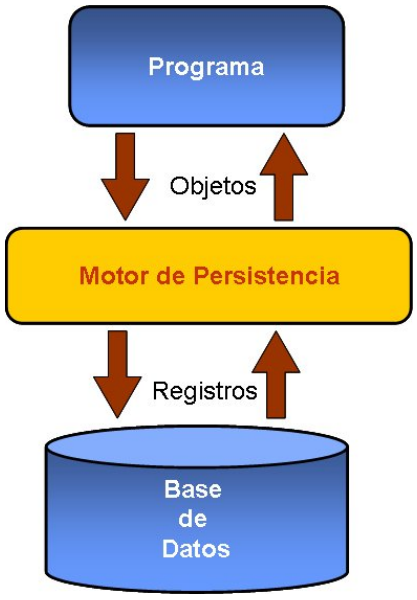


Figura 12.8. Motor de persistencia

El empleo de motores de persistencia provoca la aparición en el software de una nueva capa de programación llamada capa de persistencia, capa de datos.

A la hora de utilizar un motor de persistencia se puede pensar en programarlo directamente pero no es una buena opción porque es muy complejo y realmente tendría un gran coste pero también existen muchas alternativas bien sean de pago o libres listas para ser utilizadas. Por ello he elegido un motor de persistencia ampliamente utilizado en una plataforma madura como JAVA, el Hibernate aunque en su versión adaptada para la plataforma .NET, es decir, NHibernate.

13. CONSTRUCCIÓN DEL PROTOTIPO

El prototipo esta construido para la plataforma .NET de Microsoft y sigue el enfoque de TALISMAN orientado al mundo Web.

El prototipo genera automáticamente código ASP .NET utilizable y permite la posibilidad de añadir salidas para otras plataformas (J2EE, PHP,…) sin que este hecho implique tener que construir un nuevo prototipo, cumpliendo por tanto el objetivo del planteamiento de TALISMAN de crear aplicaciones que en una primera fase sean independientes de la plataforma de desarrollo de los sistemas y que en una última fase se obtenga una salida específica para cada sistema que se desee, automatizando el desarrollo del software para cualquier plataforma.

Como punto de partida, se crean modelos basados en UML y a partir de estos comienza el proceso de desarrollo, transformación y construcción del sistema hasta obtener el código necesario para que el sistema funcione.

Para realizar el prototipo se ha adaptado TALISMAN para los sistemas de información Web. Las aplicaciones tradicionales tienen una serie de características que las hacen diferentes a las aplicaciones Web, por ello para el desarrollo de las aplicaciones Web se mezclan métodos tradicionales de desarrollo, propuestas de ingeniería Web y métodos para hipermedia.

Para los desarrollos Web se busca agilidad por varios motivos:

- *Necesidad de una pronta disponibilidad del software en la red.* También es posible que se provoque la necesidad de realizar cambios en el software implementado.
- *Ciclos de desarrollo generalmente más cortos.* Esto obliga a que el proceso sea ágil.
- *Requisitos de usuario desconocidos.* Si los requisitos son desconocidos o variables es muy recomendado utilizar prácticas ágiles.
- Desarrollos aparentemente muy sencillos.
- Necesidad de entrega de versiones previas a la versión final del software.

La metodología TALISMAN se adapta completamente a esta motivación de agilidad para las aplicaciones Web, unicamente teniendo que definir nuevos artefactos adaptados para la Ingeniería Web.

A partir de los Diagramas y de la información de Análisis y Diseño de una aplicación, se genera la siguiente documentación en formato XML:

- Clases con la información sobre las propiedades, asociaciones y métodos de cada una de las clases que se reflejan en el Diagrama de Clases.
- Fragmentos: información semántica sobre las clases del Diagrama de Clases que se muestran en el Diagrama de Fragmentos.
- Navegación: información sobre la navegación en la aplicación Web que se haya recogida en el Diagrama de Navegación.
- Usuarios: información sobre los tipos de usuarios con sus roles que se encuentra definidos en el Diagrama de Usuarios.

- Servicios Web: especificación de los servicios Web que se encuentran recogidos en el Diagrama de servicios Web
- Servicios Web Cliente, información sobre los servicios Web Cliente que se encuentran reflejados en el diagrama correspondiente.

Toda la información se integra en un único archivo que constituye la entrada a TALISMAN.

13.1. CONSTRUCCIÓN DEL PIM

El modelo independiente de plataforma (PIM) de TALISMAN se almacena en un archivo "pim.xml" que incluye la información de la aplicación a desarrollar generada como archivos XMI con extensiones.

13.1.1. Introducción de la Información sobre la aplicación a generar

En el prototipo se introducen los modelos de entrada que serán utilizados para generar el PSM y posteriormente el código fuente de la salida.

Como entrada se tienen los siguientes modelos:

- Diagrama de clases.
- Diagrama de fragmentos.
- Diagrama de navegación.
- Diagrama de usuarios.
- Diagrama de clases de servicios Web.
- Diagrama de servicios Web cliente.

Figura 13.1. Menú de análisis (PIM)

Una vez realizado esto, aparecerá una pantalla (Figura 13.1) en la que se podrá introducir la información sobre el proyecto.

13.2. CONSTRUCCIÓN DEL PSM

A partir del PIM se genera el PSM y a partir del PSM se genera el código de salida de la aplicación. Este proceso es completamente automático.

Figura 13.2. Menú de diseño (PSM)

13.3. SALIDA GENERADA

13.3.1.1. Generación de Código fuente

Se genera el código fuente necesario para ejecutar la aplicación y será ubicado dentro de una carpeta *Solution* del proyecto (situada en *Projects*).

Para abrir el proyecto simplemente habrá que hacer doble clic en el archivo *Solution.sln* y si se tiene instalado Visual Studio .NET 2005 o cualquier otro entorno capaz de trabajar con soluciones .NET 2.0 se abrirá sin problemas. Lo primero que hay que hacer es generar la solución para que todas las clases sean compiladas y la aplicación pueda ser ejecutada.

Puede observarse que la solución generada constará de 5 proyectos:

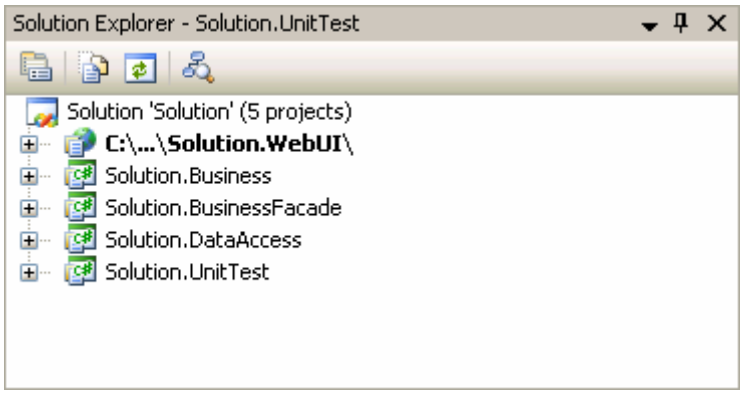


Figura 13.3. Los 5 proyectos generados

13.3.1.1.1. Interfaz de Usuario (WebUI)

Este es el proyecto principal y el que se ejecutará cuando se lance la aplicación. En WebUI se puede encontrar lo siguiente:

- Las páginas WebASPX generadas automáticamente a partir de las hojas de estilo XSLT y de la demás información de entrada, que se utilizarán como entrada del programa.
- El archivo *Web.Config* para configurar la aplicación. Aquí se puede cambiar manualmente la configuración de diversos aspectos como el sistema de Logs o las páginas que están protegidas para entrar junto con los perfiles necesarios para entrar en ellas.
- Los servicios Web que se ofrecen al exterior. Será una clase abstracta de la que habrá que heredar para rellenar el cuerpo de los métodos.
- Los servicios Web que se consumen del exterior. Será una clase "esqueleto", la cual puede utilizarse para consumir los servicios Web del exterior.
- El archivo generador de menús creado a partir de la información de la entrada.

13.3.1.1.2. Lógica de Negocio (Business)

Este proyecto contendrá una clase para cada una de las tablas que se generarán en la base de datos y mediante las cuales se podrá acceder a cada uno de los atributos de las diferentes tablas como si de objetos se tratara. Además también estarán los archivos que utilizará NHibernate para independizar el acceso a datos de la lógica de negocio.

El proyecto *BusinessFacade* es un puente entre la capa de negocio y la capa de acceso a datos, ya que proporciona métodos para cada una de las clases de la capa de negocio con los que se podrá acceder a cada una de las funcionalidades que proporciona el acceso a datos, sin importar como está implementada la clase de la capa de negocio ni la clase de acceso a datos.

13.3.1.1.3. Persistencia (DataAccess)

El proyecto DataAccess consta de un archivo y sirve para independizar el acceso a la base de datos del resto de la aplicación. Contiene métodos para guardar, obtener o borrar elementos de la base de datos, independientemente del tipo de elemento (objeto) con el que se esté tratando. Debido al uso de NUnit para probar el acceso a los datos es necesario modificar las cadenas de conexión (si hiciera falta) en el constructor del archivo *DataAccessBase.cs*

```
cfg.SetProperty("hibernate.connection.provider",
"NHibernate.Connection.DriverConnectionProvider");

cfg.SetProperty("hibernate.dialect",
"NHibernate.Dialect.MsSql2000Dialect");

cfg.SetProperty("hibernate.connection.driver_class",
"NHibernate.Driver.SqlClientDriver");
cfg.SetProperty("hibernate.connection.connection_string","Server=TEC-
BCP\\SQLEXPRESS;initialcatalog=casas_rurales;Integrated
Security=SSPI");
```

Código Fuente 13.1. Ejemplo de cadena de conexión generada automáticamente

13.3.1.1.4. Pruebas (TesttUnit)

El proyecto TestUnit contiene las pruebas unitarias generadas automáticamente que sirven para probar la base de datos mediante las operaciones de salvar, obtener, modificar y borrar con las clases generadas en la capa de negocio. Al final de las pruebas unitarias la base de datos quedará en el mismo estado que al empezarlas. Se debe tener instalado NUnit y las tablas creadas en la base de datos.

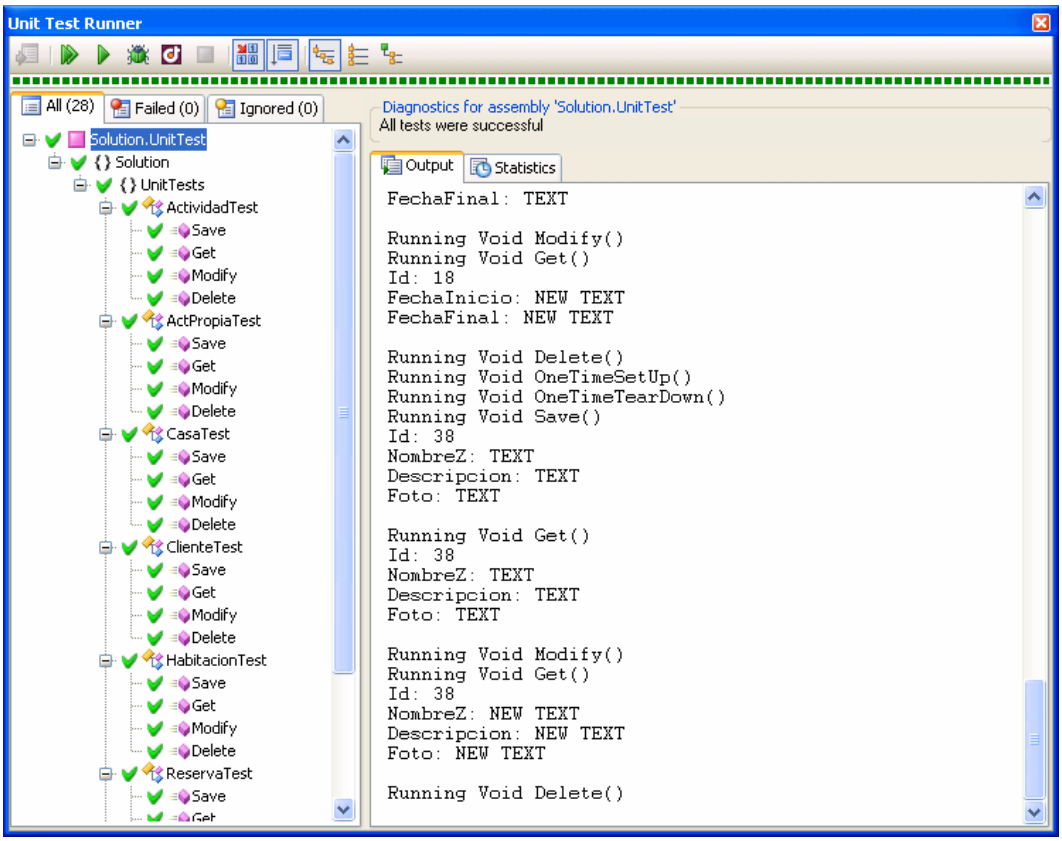


Figura 13.4. Ejemplo de ejecución del proyecto UnitTest

13.3.1.1.5. Tablas de la base de datos

A través de un script (*DataBase*) se genera la estructura de la base de datos.

Por ejemplo para Sql Server 2005 se generará un archivo llamado *DataBase.sql* y crear las tablas en la base de datos es tan sencillo como hacer doble click en el archivo y una vez abierto Micrososft Sql Server Management Studio pulsar *Execute*. Hay que tener en cuenta que para crear las tablas dentro de la base de datos, la base de datos tiene que estar creada previamente.

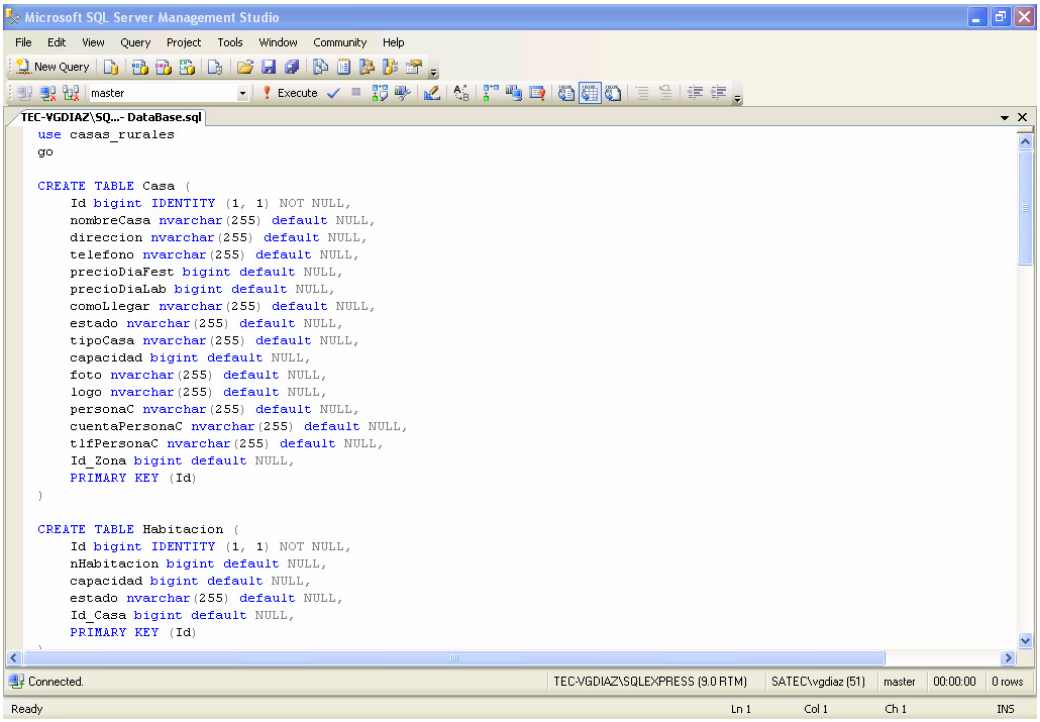


Figura 13.5. Aspecto de Microsoft Sql Server Management Studio

13.3.1.2. Administración de usuarios y perfiles de usuario

Con el diagrama de usuarios se puede restringir el acceso a determinadas zonas del sitio Web y además determinar que perfil de usuario es el que puede visitarlas.

En el archivo Web.Config contiene la forma de acceso, cuando es necesario poseer los permisos de "*Admin*". Además para realizar la autenticación (login) se redirecciona automáticamente a la página *Login.aspx.*

```
  <system.web>
    <roleManager enabled="true" />
    <authentication mode="Forms">
      <forms name="Login" loginUrl="Login.aspx" protection="All"
timeout="45" >
      </forms>
    </authentication>
    <compilation debug="true"/>
  </system.web>

  <location path="Reserva.aspx">
        <system.web>
          <authorization>
            <allow roles="Admin" /><deny users="?" />
          </authorization>
        </system.web>

      </location>
```

Código Fuente 13.2. Ejemplo de sección de Web.Config de restricciones de usuarios

La configuración de usuarios y perfiles se realiza a través de *ASP .NET Configuration.*

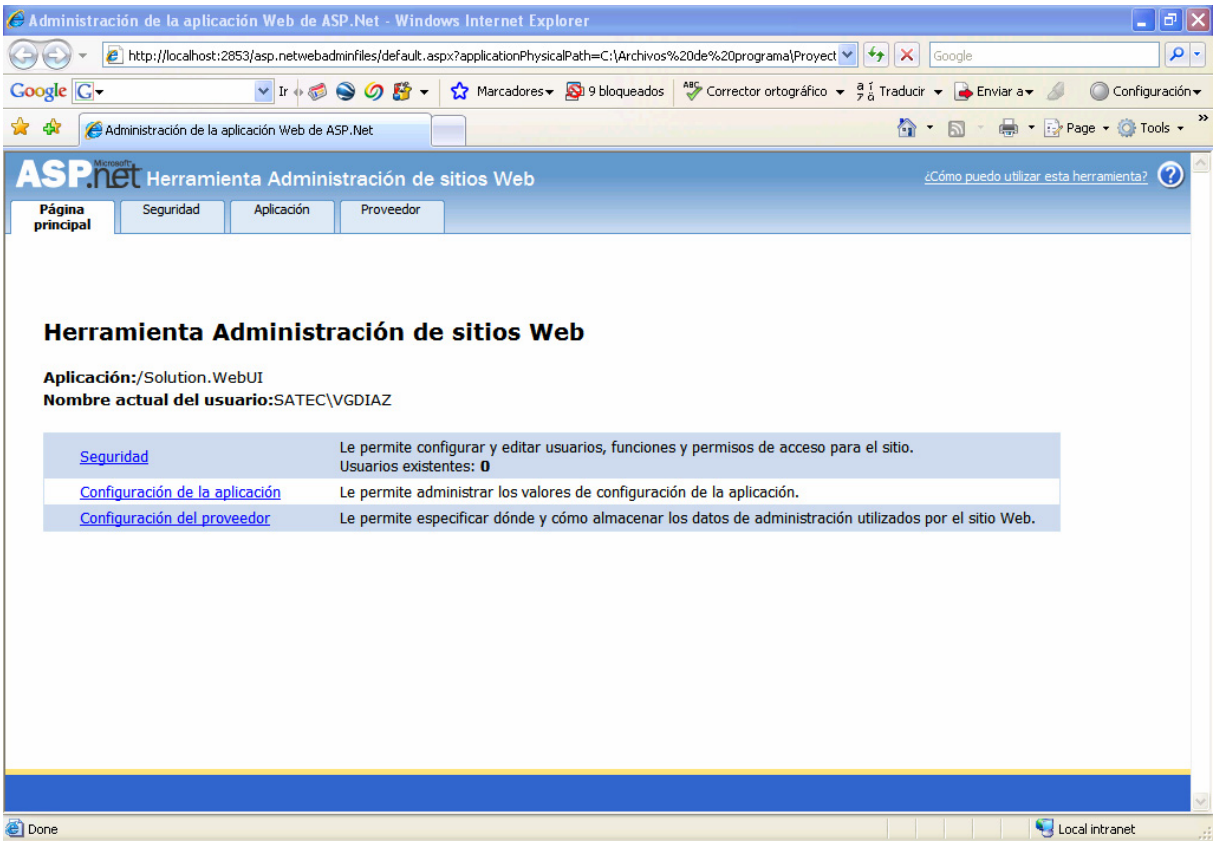


Figura 13.6. Aspecto de la herramienta de configuración de sitios Web

Para crear un usuario habría que ir a la sección *Seguridad → Crear o administrar funciones* y allí crear una función *Admin*. En *Seguridad → Crear usuario* se crearía un usuario con su contraseña indicando que tiene funciones de *Admin* y ese usuario ya estaría en condiciones de acceder al sitio Web.

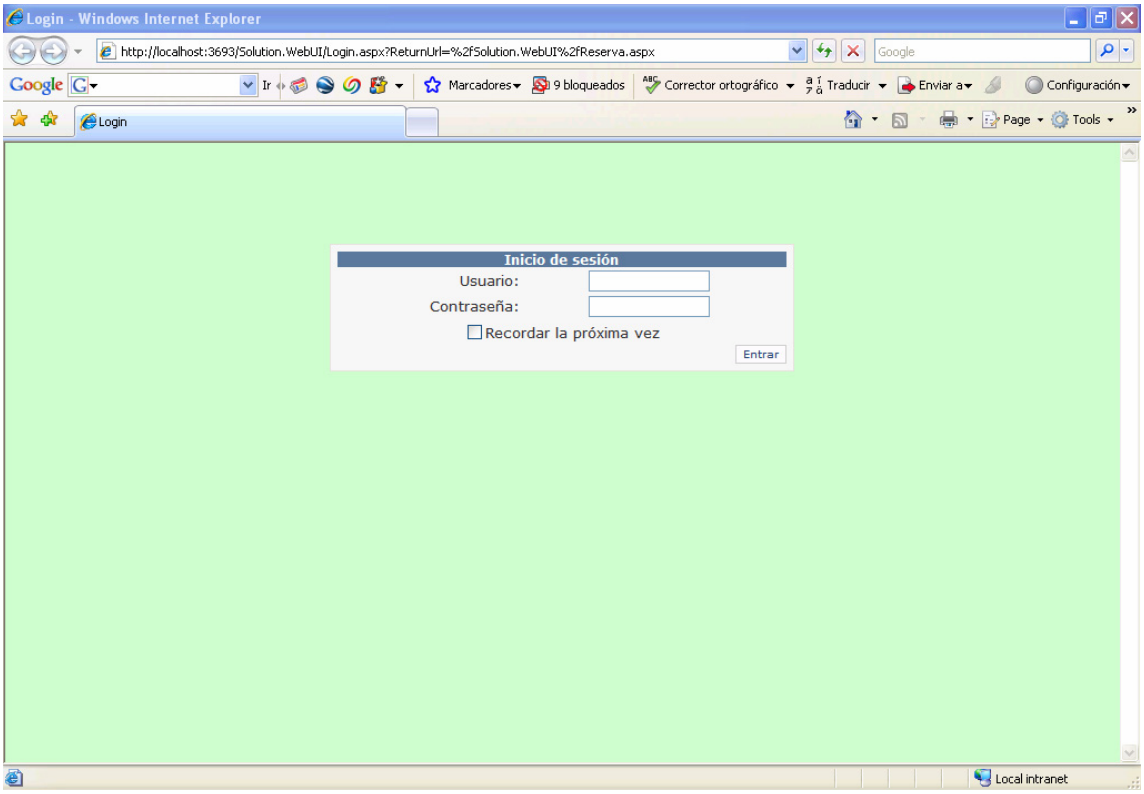


Figura 13.7. Posible aspecto de la página de Login

13.4. REGISTROS (LOGS)

13.4.1. Registro (log) del Proyecto generado

Dentro de la carpeta *Log* del proyecto (situada en *Projects*) aparece un archivo denominado de la misma forma que el nombre del Proyecto. En ese archivo aparecerá un resumen de todas las acciones realizadas por TALISMAN en el proceso de creación del código de salida para la plataforma destino.

El archivo tendrá las siguientes secciones principales:

- Beginning: Básicamente se basa en crear directorios y copiar archivos que son siempre iguales, independientemente del proyecto a crear.
- PIM to PIM: Se divide en contenido, aplicación Web y funcionalidad.
- PIM to PSM: Se divide en contenido, aplicación Web y funcionalidad.
- PSM to PSM: Se divide en contenido, aplicación Web y funcionalidad.
- PSM to Code: Se divide en contenido, aplicación Web y funcionalidad.

13.4.2. Registro (log) de la ejecución del Proyecto generado

Dentro de la carpeta *Solution/Solution.WebUI/Logs* del proyecto (situada en *Projects*) se ubicarán dos archivos, uno de los cuales se llamará *general.log* (aparecerán diferentes acciones que se van realizando mientras un usuario visita las páginas de la aplicación Web) y el otro *nhibernate.log* (aparecerán mensajes que se ven generando a medida que se van visitando páginas de la aplicación Web, todos ellos relacionados con la ejecución de NHibernate).

14. PRUEBAS DEL PROTOTIPO

Para comprobar la validez del prototipo desarrollado, se ha constuido una aplicación denominada "*Casas rurales en Asturias*", que genera una aplicación Web en la que se pueda consultar información a cerca de casas rurales de diferentes zonas en Asturias y de las actividades que en esas casas se pueden realizar. La aplicación generada está definida para la plataforma ASP .NET 2.0 funcionando con el sistema de gestión de bases de datos Sql Server 2005.

14.1. ANÁLISIS Y DISEÑO

El análisis y diseño de la aplicación se puede realizar con cualquier herramienta que soporte UML 2.0 y posteriores. La herramienta debe permitir la exportación de los modelos y diagramas al formato XMI.

Posteriormente el formato XMI debe ser enriquecido con información semántica dando lugar a un archivo XML específico de TALISMAN.

14.1.1. Diagrama de Clases

El Diagrama de Clases de la Figura 14.1 es el resultado del proceso descrito en las secciones 12.21 y 12.2.2

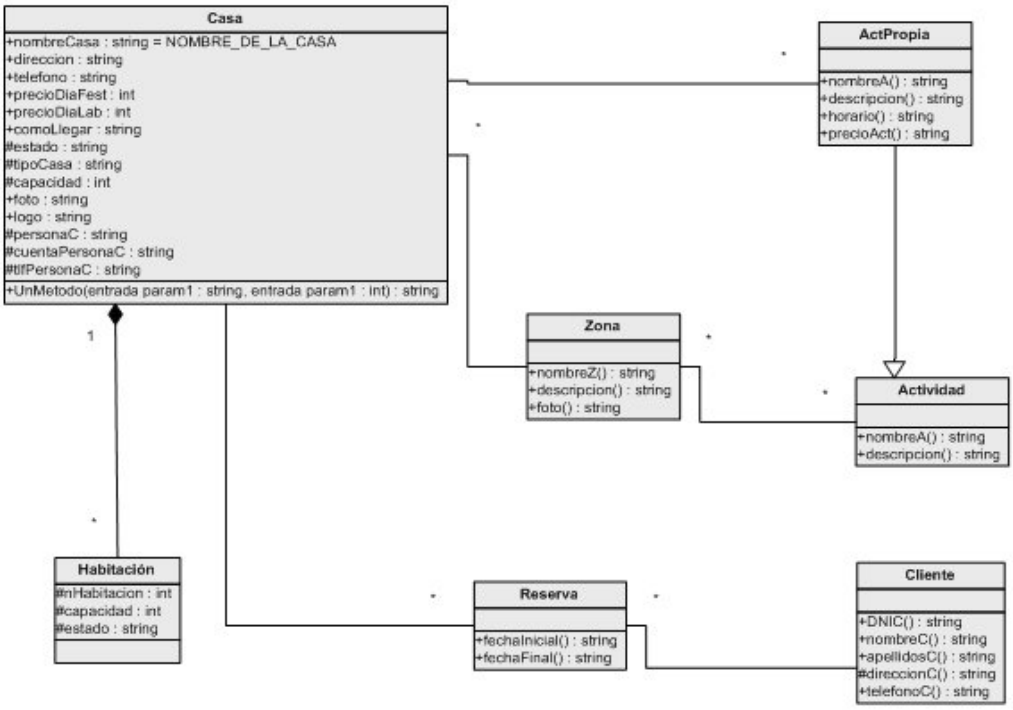


Figura 14.1. Diagrama de clases

14.1.2. Diagramas de Interfaz de Usuario

En este caso la interfaz de usuario será una interfaz Web y se especificará a través de los siguientes diagramas:

- Diagrama de fragmentos
- Diagrama de navegación

14.1.2.1. Diagrama de Fragmentos

En la

Figura ***14.2.*** se especifica el diagrama de fragmentos de la aplicación "Casas rurales".

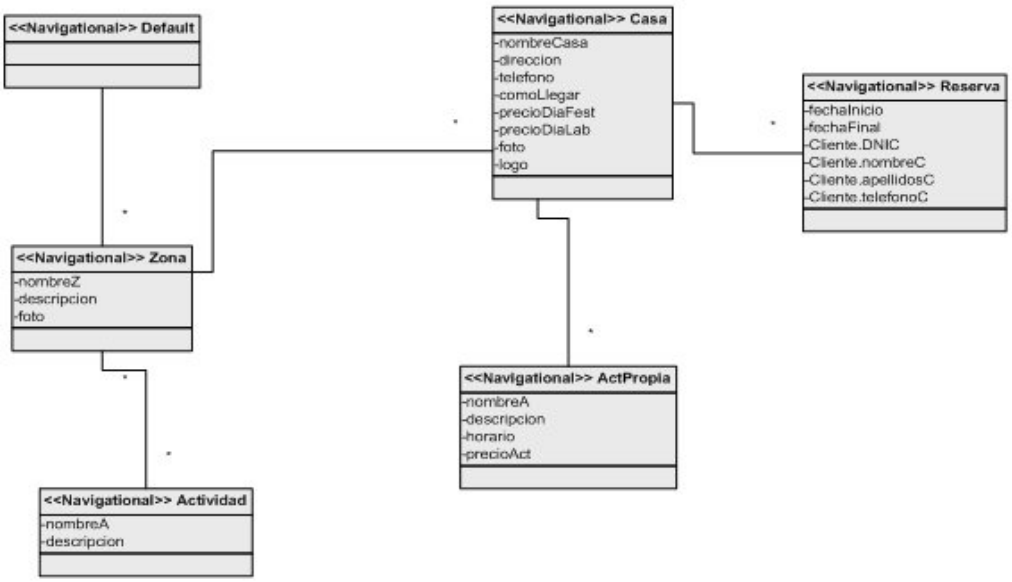


*Figura **14.2**. Diagrama de fragmentos*

14.1.2.2. Diagrama de Navegación

En la Figura 14.3. se especifica el diagrama de navegación de la aplicación "Casas rurales".

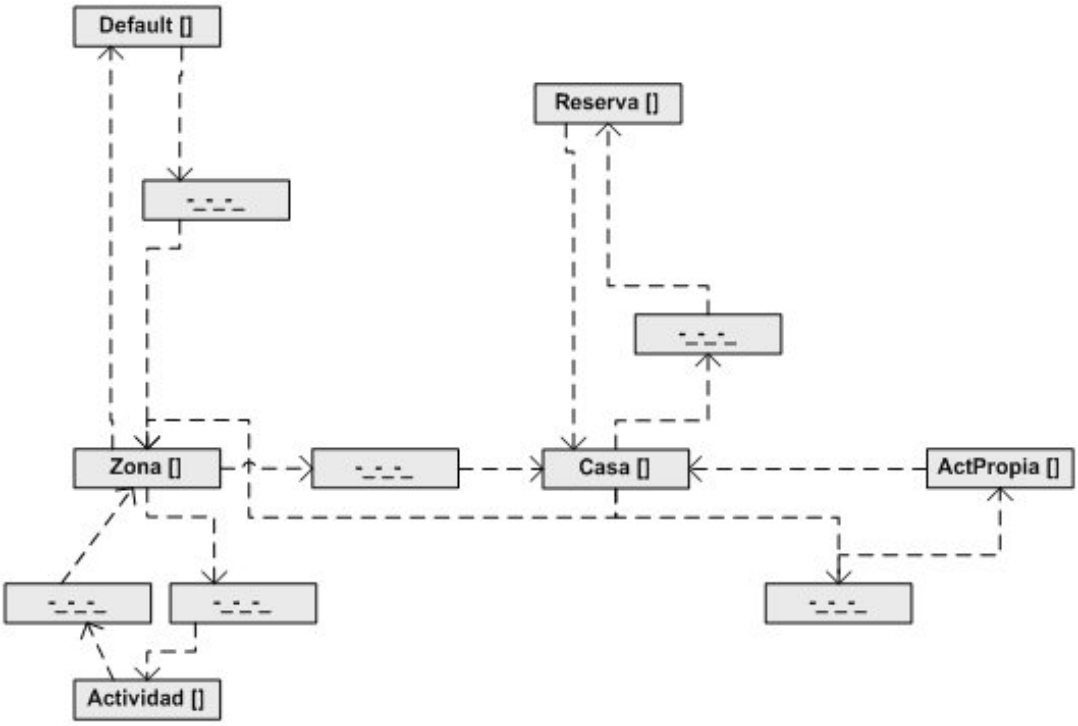


Figura 14.3. Diagrama de navegación

14.1.3. Diseño de la Lógica de Negocio

En este caso la lógica de negocio se describe a través de servicios Web que se especificará a través de los siguientes diagramas:

- Diagrama de clases de servicios Web
- Diagrama de servicios Web Cliente
- Diagrama de usuarios

14.1.3.1. Diagrama de Servicios Web

En la Figura 14.4 se especifica el diagrama de clases de servicios Web de la aplicación "Casas rurales".

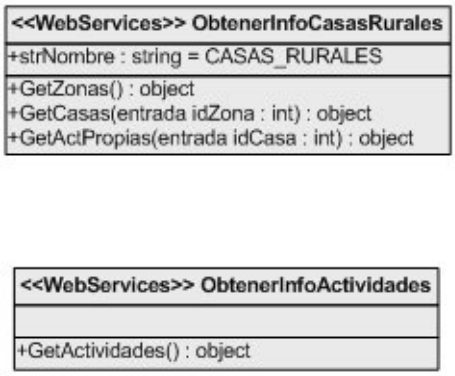


Figura 14.4. Diagrama de clases de servicios Web

14.1.3.2. Diagrama de Servicios Web Cliente

En la Figura 14.5 se especifica el diagrama de servicios Web Cliente de la aplicación "Casas rurales".

Figura 14.5. Diagrama de servicios Web cliente

14.1.3.3. Diagrama de Usuarios

En la Figura 14.6 se especifica el diagrama de usuarios de la aplicación "Casas rurales".

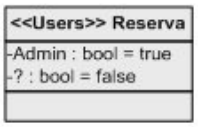


Figura 14.6. Diagrama de usuarios

14.2. XMI + EXTENSIONES

A partir de los Diagramas y de la información de Análisis y Diseño se genera la siguiente documentación en formato XML:

- Clases con la información sobre las propiedades, asociaciones y métodos de cada una de las clases que se reflejan en el Diagrama de Clases.
- Fragmentos: información semántica sobre las clases del Diagrama de Clases que se muestran en el Diagrama de Fragmentos.
- Navegación: información sobre la navegación en la aplicación Web que se haya recogida en el Diagrama de Navegación.
- Usuarios: información sobre los tipos de usuarios con sus roles que se encuentra definidos en el Diagrama de Usuarios.
- Servicios Web: especificación de los servicios Web que se encuentran recogidos en el Diagrama de servicios Web
- Servicios Web Cliente, información sobre los servicios Web Cliente que se encuentran reflejados en el diagrama correspondiente.

Toda la información se integra en un único archivo que constituye la entrada a TALISMAN.

14.3. TALISMAN

14.3.1. Modelo PIM

El modelo independiente de plataforma (PIM) de TALISMAN se almacena en un archivo "pim.xml" que incluye la información de la aplicación a desarrollar generada como archivos XMI con extensiones.

El contenido de este archivo refleja cada uno de los apartados del Análisis y Diseño de los apartados anteriores como se muestra en el Código Fuente 14.1. El archivo completo se encuentra en el *Anexo B. Archivos de Prueba del Prototipo.*

```
<?xml version="1.0" encoding="utf-8"?>
<PIM>
   <classes>
   <fragments>
   <navigation>
   <users>
   <webservices>
   <webservicesClients>
</PIM>
```

Código Fuente 14.1.Esqueleto del archivo PIM

Cada una de las secciones del archivo PIM se corresponde con los diagramas del Análisis y Diseño, en el Código Fuente 14.2 se muestra el contenido de archivo para la clase "*casa*", con la información asociada, tal y como se muestra en el Diagrama de Clases de la Figura 14.1.

```
<class name="Casa" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="nombreCasa" visibility="public" type="string"
initial="&quot;NOMBRE_DE_LA_CASA&quot;" changeability="changeable" isStatic="true"
/>
        <property name="direccion" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="telefono" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="precioDiaFest" visibility="public" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="precioDiaLab" visibility="public" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="comoLlegar" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="estado" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="tipoCasa" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="capacidad" visibility="protected" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="foto" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="logo" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="personaC" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="cuentaPersonaC" visibility="protected" type="string"
initial="" changeability="changeable" isStatic="false" />
        <property name="tlfPersonaC" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="ActPropia" multiplicity="multiple" />
        <association target="Zona" multiplicity="1" dependent="false" />
        <association target="Reserva" multiplicity="multiple" />
        <association target="Habitacion" multiplicity="multiple" />
      </associations>
      <methods>
```

```
        <method name="UnMetodo" type="normal" visibility="public"
returnType="string" isAbstract="false">
          <parameters>
            <param name="param1" type="string" />
            <param name="param2" type="int" />
          </parameters>
        </method>
      </methods>
    </class>
```

Código Fuente 14.2.Fragmento de Código del archivo "pim.xml" para la clase "casa"

Dentro de la sección "*classes*" se encuentran definidas las clases del diagrama de clases de la Figura 14.1.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" >
      <head runat="server">
        <title>
          Casa - <xsl:value-of select="Casa/nombreCasa" />
        </title>
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <menus />
      </head>
      <body>
        <form id="form1" runat="server">
          <div>
              <div class='info'>
              <h1>Casa</h1>
                <table summary="Casa">
                  <tr>
                    <td>
                      <span class='title'>Nombre: </span><span
class='titleInfo'><xsl:value-of select="Casa/nombreCasa" /></span><br/><br/>
                      <span class='title'>Dirección: </span><span
class='titleInfo'><xsl:value-of select="Casa/direccion" /></span><br/><br/>
                      <span class='title'>Teléfono: </span><span
class='titleInfo'><xsl:value-of select="Casa/telefono" /></span><br/><br/>
                      <span class='title'>Como llegar: </span><span
class='titleInfo'><xsl:value-of select="Casa/comoLlegar" /></span><br/><br/>
                      <span class='title'>Precio día festivo: </span><span
class='titleInfo'><xsl:value-of select="Casa/precioDiaFest" />€</span><br/><br/>
                      <span class='title'>Precio día laborable: </span><span
class='titleInfo'><xsl:value-of select="Casa/precioDiaLab" />€</span><br/><br/>
                    </td>
                    <td class="menuTitle" valign="top"><Casa.Zona /></td>
                    <td class="menuTitle" valign="top"><Casa.ActPropia /></td>
                    <td class="menuTitle" valign="top"><Casa.Reserva /></td>
                  </tr>
                </table>
                <p style="text-position:center">
                  <img style="border:none;text-align:center" alt="Foto de la
casa"><xsl:attribute name="src">Hypertext/Media<xsl:value-of select="Casa/foto"
/></xsl:attribute></img>
                </p>
            </div>
            <p>
              <a href="http://validator.w3.org/check?uri=referer"><img
src="Hypertext/Media/valid-xhtml10.png" alt="Valid XHTML 1.0 Transitional"
height="31" width="88" /></a>
              <a href="http://jigsaw.w3.org/css-validator/#validate-by-input"><img
src="Hypertext/Media/vcss.gif" alt="Valid CSS 2.1" height="31" width="88" /></a>
              <a href="http://www.w3.org/WAI/WCAG1-Conformance"><img
src="Hypertext/Media/vaa.gif" alt="Valid WAI-AA" height="31" width="88" /></a>
```

```
            </p>
          </div>
        </form>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Código Fuente 14.3. Código del archivo de transformación "casa.xslt"

14.3.2. Transformaciones mediante archivos XSLT y CSS

Las transformaciones se especifican a través de archivos XSLT (Extensible Stylesheet Language Transformations) y el estilo de la aplicación Web a desarrollar a través de archivos CSS de Hojas de Estilo.

El archivo XSLT de transformación del modelo PIM al modelo PSM de la clase "casa" se muestra en el Código Fuente 14.3.

Además de esta transformación, en la aplicación de "*Casas Rurales*" se han definido los siguientes archivos de transformación, que se encuentran en el *Anexo B. Archivos de Prueba del Prototipo:*

- Actividad.xslt
- ActPropia.xslt
- Casa.xslt
- Default.xslt
- Error.xslt
- Reserva.xslt
- StyleSheet.css
- Zona.xslt

14.3.3. Recursos adicionales

Además de los archivos con los modelos y las transformaciones, para realizar la interfaz de usuario es necesario proporcionar al prototipo todos los archivos de imágenes necesarios.

Estos archivos deben ser ubicados dentro de una carpeta denominada "Media" dentro del proyecto. En el caso de la aplicación de las casas rurales los archivos que se han proporcionado son fotografías de alguna casa rural asturiana, así como los iconos de conformidad con los estándares del W3C. En la Figura 14.7 se muestra el contenido fotográfico de la carpeta Media.

Figura 14.7. Imágenes de la carpeta Media.

14.4. SALIDA GENERADA

En la Figura 14.8 se muestra la aplicación de Casas Rurales generada utilizando el prototipo de la metodología TALISMAN. El prototipo genera todos los archivos necesarios para la ejecución de la aplicación sobre la plataforma ASP .NET.

La lista completa de archivos generados para la aplicación de "Casas Rurales" se encuentra desglosada en el *Anexo B*.

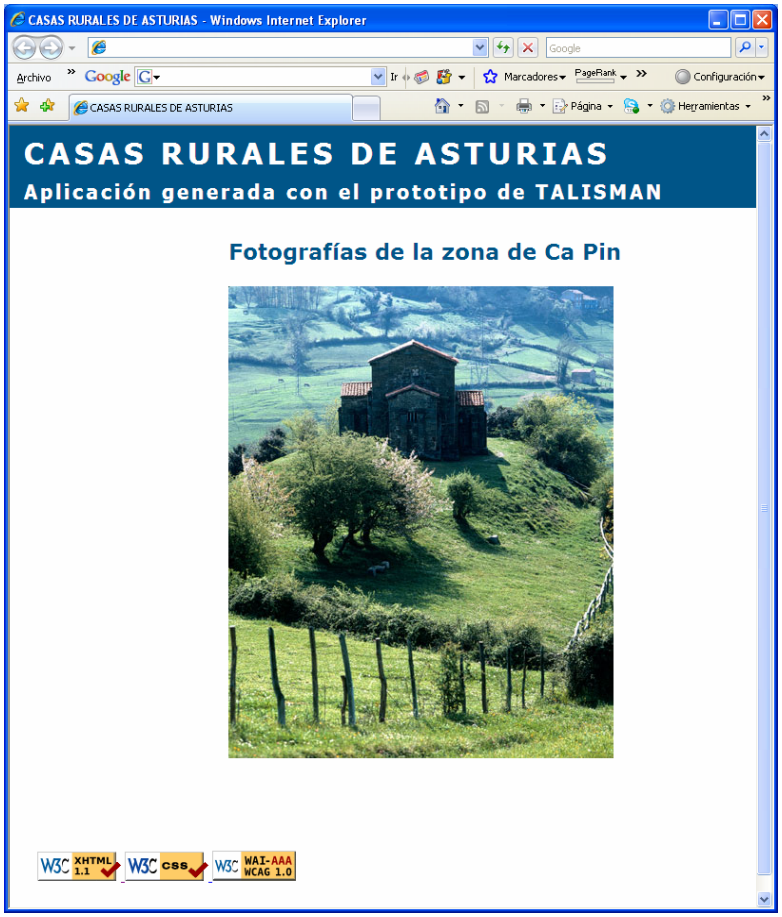


Figura 14.8.Aspecto de la Aplicación de casas rurales.

15. CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS

En este capítulo se exponen las conclusiones principales de la tesis.

En primer lugar, en el apartado 15.1 se hace un repaso y evaluación de las acciones realizadas a partir de la hipótesis de partida.

Por otro lado, en el apartado 15.2 se realiza una comprobación del nivel de cumplimiento de los objetivos presentados en la introducción.

El apartado 15.3 describe de forma resumida las principales aportaciones resultantes de la investigación, destacando aquellas características inéditas que resultan novedosas o innovadoras.

Con el objetivo de dar a conocer a la comunidad científica los avances obtenidos, en el apartado 15.4 se exponen los trabajos presentados en congresos y jornadas internacionales, relacionados con los temas abordados en esta tesis.

Por último, en el apartado 15.5, se presentan las líneas de investigación futuras que se desea seguir a partir de esta tesis.

15.1. EVALUACIÓN DE LA HIPÓTESIS DE PARTIDA

Inicialmente, dentro del "*Capítulo 1. Introducción a la investigación*", se plantearon varios problemas que justicaban el desarrollo de esta investigación. De forma resumida dichos problemas son los siguientes:

- La guía oficial de MDA [OMG] es ambigua con respecto al tratamiento de los modelos iniciales. No aclara cómo deben manejarse los modelos ligados al negocio, es decir, los modelos independientes de la computación ó CIM.
- La guía oficial de MDA [OMG] tampoco describe cómo deben asociarse esos modelos CIM con los primeros modelos del software a desarrollar, es decir, con los modelos independientes de las plataformas ó PIM.
- No existen muchos trabajos de investigación en el área de MDA que se centren en el tratamiento de los modelos iniciales, CIM y PIM. La mayoría de desarrollos basados en MDA comienzan utilizando modelos PIM. Por lo tanto, en la aplicación de esta especificación suele existir un importante hueco entre el dominio del negocio y el tecnológico.
- Las metodologías de desarrollo de software ágiles, tiene como punto fuerte la conexión de los aspectos asociados al negocio con los tecnológicos. Sin embargo no se ajustan a la especificación MDA y por lo tanto no puede aportar los beneficios inherentes a esta especificación, relacionados con el desarrollo de software dirigido por modelos y la separación de aspectos lógicos respecto a los detalles sobre las plataformas.

Como consecuencia de esta problemática se planteó la siguiente hipótesis de partida:

> ***La combinación de la especificación MDA con la idea del desarrollo ágil resultaría muy ventajosa de cara a un desarrollo de software flexible y asociado a los procesos del negocio.***

En el desarrollo de la tesis se han abordado todos los aspectos relacionados con esta hipótesis con el objetivo de comprobar la viabilidad de la misma. Concretamente se han estudiado y desarrollado los siguientes conceptos, incluidos en la hipótesis y considerados clave dentro de nuestra investigación:

- **La especificación MDA**. Representa uno de los pilares en los que se fundamenta la tesis. Por dicho motivo el capítulo 3 se ha dedicado a exponer los aspectos de éste estándar relacionados con el desarrollo de software, haciendo hincapié en los tipos de modelos asociados al negocio, clasificados como independientes de la computación ó CIM, y los modelos iniciales del software, clasificados como independientes de las plataformas ó PIM. Dado que la hipótesis de partida plantea una mejora en el desarrollo del software, dentro del mencionado capítulo se ha dado mucha importancia a la definición de un entorno de desarrollo de software basado en MDA.
- **Las metodologías ágiles**. Es otro de los aspectos fundamentales de la tesis. Ha sido expuesto en el capítulo 2 destacando de esta idea la combinación de una perspectiva de negocio, que persigue mejorar la competitividad de la empresa ante cambios en la demanda o en el mercado.
- **Desarrollo de software flexible y asociado a los procesos del negocio**. Con esta expresión nos referimos a un proceso que debe permitir construir o modificar software para poder reaccionar rápidamente ante los cambios, en este caso, ante cambios en las reglas de negocio.

La combinación de desarrollo ágil de software y MDA se ha materializado en los tres capítulos que componen las aportaciones principales de la tesis. En el capítulo 12 se describe desde una perspectiva muy general. En el capítulo 13 se exponen los pasos que se recomiendan para la creación de un prototipo basado en desarrollo de software ágil y MDA. Por último, el capítulo 14 se presentan los pasos para ejecutar un proceso de desarrollo de software que sea flexible y esté asociado a un proceso de negocio concreto.

Tras una descripción de cuáles han sido los aspectos clave que se han estudiado dentro de la investigación, en el siguiente apartado se expone el grado de cumplimiento de los objetivos planteados.

15.2. VERIFICACIÓN Y EVALUACIÓN DE LOS OBJETIVOS

En la introducción de la tesis se planteó el siguiente objetivo principal:

> ***Definir una metodología para el desarrollo de software que permita la creación de software a partir del modelado de procesos del negocio de acuerdo a la especificación MDA y el desarrollo de software ágil.***

Este objetivo principal se ha dividido en una serie de objetivos parciales, cuya verificación, evaluación y grado de cumplimiento se comentan a continuación.

- **Objetivo 1: Realizar un estudio de las áreas relacionadas con el objetivo principal, es decir, MDA, desarrollo ágil de software, patrones de diseño y estándares.**

En el capítulo 2 se ha llevado a cabo un estudio sobre el desarrollo de software ágil. Los resultados del estudio de la especicación MDA se recogen en el capítulo 3. Además, en el tema 4 se ha realizado un estudio de los patrones de diseño. Sobre los estándares se ha profundizado en los capítulos del 5 al 8, desarrollando los conceptos sobre UML, OCL, MOF y XMI.

- **Objetivo 2: Definir una metodología para el desarrollo de software ágil de acuerdo a la especificación MDA.**

Este es el objetivo más importante. Tras el estudio de las áreas ya mencionadas en el primer objetivo, los capítulos 12, 13 y 14 recogen los resultados del desarrollo de la investigación. Concretamente el capítulo 12 describe de forma general los aspectos en los que se basa la metodología, asociándolos a las áreas descritas en el estado del arte (capítulo 2 al 11). Los puntos clave en los que se basa esta metodología y que demuestran que el objetivo planteado se ha alcanzado, son los siguientes:

  - La recomendación propuesta es una interpretación válida de la especificación MDA.
  - Emplea principios básicos del desarrollo de software ágil para la definición y modelado de los procesos del negocio.

- Los modelos de tipo CIM están formados por los modelos de procesos del negocio.

- **Objetivo 3: Crear un prototipo que implemente la metodología.**

Debido a la multitud de pasos, actividades, roles y artefactos que incluye la metodología, se ha creado un prototipo que permite manejarla fácilmente.

- **Objetivo 4: Desarrollar una aplicación usando la metodología y utilizando el prototipo.**

En el capítulo 14 se recogen los resultados obtenidos tras la realización de una aplicación Web sobre Casa Rurales.

Una vez verificado el cumplimiento de los objetivos planteados, en el próximo apartado se describen las aportaciones originales de esta investigación.

15.3. APORTACIONES DE LA TESIS

El desarrollo de la investigación realizada ha permitido obtener una serie de aportaciones originales o novedosas que son descritas a continuación en diferentes categorías de cara a una mejor comprensión de las mismas.

15.3.1. Aportaciones en el ámbito de MDA

Como ya se ha comentado con anterioridad, MDA es uno de los pilares fundamentales en esta tesis. De hecho, la justificación de la investigación está relacionada con una serie de problemas y carencias de esta especificación y nuestras aportaciones representan propuestas para dar solución a dichos problemas. Concretamente:

- **Definición de un entorno de desarrollo basado en MDA.**

La especificación MDA no indica cómo las empresas deben adaptar su entorno de desarrollo para poder afrontar proyectos basados en MDA. Nos referimos a cómo deben cambiar elementos clave como los tipos de modelos a utilizar y lenguajes asociados, la arquitectura y transformaciones entre modelos, el proceso de desarrollo y las herramientas asociadas.

Obviamente cada empresa deberá definir su propio entorno en función a sus características, posibilidades, tipos de proyectos a realizar, etc. aunque siempre dentro de unas directrices básicas marcadas por la propia especificación MDA.

- **Definición de modelos CIM a partir de los procesos del negocio.**

Uno de los problemas presentados por la guía oficial de MDA [OMG] es que no define cómo deberían crearse los modelos de tipo CIM ni aporta detalles sobre cuál debería ser su contenido. En esta tesis se ha presentado una metodología y un prototipo que permiten la creación de modelos CIM a partir del modelado y especificación de procesos del negocio.

- **Directrices para la creación de modelos PIM a partir de modelos CIM.**

La especificación MDA no define cómo debe realizarse la transformación entre modelos CIM y PIM. La metodología aporta una propuesta para realizar esta transformación partiendo de una serie de restricciones, asociadas al las reglas de negocio.

- **Una metodología que mejora la conexión entre las reglas de negocio y las tecnologías MDA.**

La metodología enlaza los conceptos del dominio del negocio, representados y especificados en los modelos de procesos del negocio que forman el CIM, con los requisitos y elementos de análisis del software que forman los modelos iniciales del software, configurados como modelos PIM.

15.3.2. Aportaciones en el ámbito del desarrollo de software ágil

Otra de las áreas principales de esta tesis es la relacionada con la idea del desarrollo de software ágil. Nuestra investigación ha estudiado éstas áreas buscando su relación con MDA para alcanzar una mejora de la conexión del dominio empresarial con el tecnológico y, sobre todo, una mejora en los procesos de desarrollo de software, habitualmente no ligados a aspectos del negocio.

Nuestras principales aportaciones en este campo son las siguientes:

- **Una metodología para el desarrollo de software ágil que pueda adaptarse a las necesidades y características particulares de cada empresa.**

El éxito del desarrollo ágil de software depende de la forma en la que sea implantado en la empresa y de la cultura con la que sea utilizado. Dado que se ha estudiado las metodologías de desarrollo ágil de software dentro del ámbito de la lógica de Negocio, su implantación no debe ser afrontada como un proyecto meramente tecnológico o únicamente empresarial. Debe hacerse teniendo en cuenta ambos aspectos. Se propone un proceso basado en la aplicación de diversas iteraciones que permita a todo tipo de empresas alcanzar el escenario de implantación ajustado a sus necesidades y características particulares.

- **Una metodología de desarrollo de software que combina las ventajas del MDA y el desarrollo ágil.**

Esta es una de las principales aportaciones puesto que, como ya se ha mencionado en anteriores ocasiones, la mayoría de proyectos de desarrollo de software basados en MDA parten de modelos de tipo PIM y son considerados desde una perspectiva muy ligada al desarrollo del software, en la que no se tienen en cuenta los aspectos del negocio y las metodologías ágiles.

15.4. TRABAJOS DERIVADOS

Los siguientes trabajos están relacionados con la investigación realizada. Han sido presentados en diferentes congresos y jornadas internacionales con el objetivo de difundir ante la comunidad científica los resultados obtenidos.

[PC05] B. C. Pelayo García-Bustelo y J. M. Cueva Lovelle.
Evolución de las directrices de accesibilidad del W3C.
En Libro de Actas del III Simposio Internacional de Sistemas de Información en la Sociedad del Conocimiento. Instituto Tecnológico de las Américas (República Dominicana), 2005.

[PC06a] B. C. Pelayo García-Bustelo y J. M. Cueva Lovelle.
Arquitecturas dirigidas por modelos (MDA). El framework C3NET.
II Congreso Internacional de Ingeniería de Computación y Sistemas (IICIIS). Trujillo, PERÚ, 2006.

[PC06b] B. C. Pelayo García-Bustelo y J. M. Cueva Lovelle.
Usabilidad, accesibilidad y métricas de sitios Web.
II Congreso Internacional de Ingeniería de Computación y Sistemas

(IICIIS). Trujillo, PERÚ, 2006.

[PCJ06] B. C. Pelayo García-Bustelo, J. M. Cueva Lovelle, y A.A. Juan Fuente. *C3NET: Smart Environment For .NET Code Generation Using MDA.* En WORLDCOMP'06 Proceedings: The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing (composed of 28 Joint Conferences), Las Vegas, USA, 2006.

[PCS+05] B.C. Pelayo García-Bustelo, J.M. Cueva Lovelle, M.C. Suárez Torrente, y A.A. Juan Fuente. *C3NET: Framework para la construcción de MDA en la Plataforma .NET.* III Simposio Internacional de Sistemas de Información en la Sociedad del Conocimiento. Instituto Tecnológico de las Américas (República Dominicana), 2005.

15.5. LÍNEAS DE INVESTIGACIÓN FUTURAS

Esta tesis representa un primer paso en la definición de un proceso de desarrollo de software que combina MDA con desarrollo ágil de software. Debido a lo extenso del área que se ha tratado, se han dejado varias líneas abiertas para completar dicho proceso en un futuro. Principalmente nos referimos a las siguientes líneas de investigación:

- Aplicar de forma completa la metodología a un caso real.
- Ampliar la metodología expuesta en esta tesis, que se ha centrado en las etapas iniciales y finales del proceso de desarrollo. Se deberán definir los pasos y actividades para otras plataformas tecnológicas.
- Debido a la importancia que tienen las herramientas utilizadas durante el proceso de desarrollo, realizar un estudio de las herramientas actualmente disponibles para evaluar cuál es su nivel de utilidad dentro de la metodología propuesta.
- Debido a que la metodología presentada es muy amplia para poder ser empleada en todo tipo de proyectos, definir unas guías de aplicación a proyectos especificos, especialmente en el ámbito de la Ingeniería Web.

ANEXOS

Anexo A. CÓDIGOS FUENTES DEL PROTOTIPO

WebTALISMAN.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using NUML.Uml2;
using NUML.Xmi2;
using log4net;
using log4net.Config;

namespace WebTALISMAN
{
    /// <summary>
    /// Punto de inicio de la aplicación
    /// </summary>
    static class WebTALISMAN
    {
        private static readonly ILog log =
LogManager.GetLogger(typeof(WebTALISMAN));
        /// <summary>
        /// El punto de entrada de la aplicación
        /// </summary>
        [STAThread]
        static void Main()
        {
            XmlConfigurator.Configure(new System.IO.FileInfo("log4net.xml"));

            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            log.Info("**********************");
            log.Info("Entering application WebTALISMAN");
            Application.Run(new MainForm());
            log.Info("Exiting application WebTALISMAN");
        }
    }
}
```

NewProjectForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace WebTALISMAN
{
    /// <summary>
    /// Para crear nuevos proyectos
    /// </summary>
    public partial class NewProjectForm : Form
    {
        private string newFile;

        /// <summary>
        /// Constructor
        /// </summary>
        public NewProjectForm()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Confirmación de creación
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text != "")
            {
                newFile = textBox1.Text;
                Close();
            }
        }

        /// <summary>
        /// Devuelve el nuevo archivo del proyecto
        /// </summary>
        /// <returns></returns>
        public string getNewFile()
        {
            return newFile;
        }

        /// <summary>
        /// Evento que se procude al cargar el formulario
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void NewProjectForm_Load(object sender, EventArgs e)
        {

        }
    }
}
```

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.IO;
using WebTALISMAN.Models;
```

```
namespace WebTALISMAN
{
    /// <summary>
    /// Formulario principal de la aplicación
    /// </summary>
    public partial class MainForm : Form
    {
        private StringDictionary param; //Parámetros del documento
        private bool saved; //Si se ha guardado o no la última versión del documento
        private GeneratorASPNET2_0 generator; //Para generar páginas

        #region "Events"

        /// <summary>
        /// Cierre de la aplicación
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        /// <summary>
        /// Se abre un proyecto existente
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            bool execute = false;
            createPSMButton.Enabled = false;
            createCODEButton.Enabled = false;

            //Si hay un projecto nuevo que ha sufrido cambios se da opción a
guardarlos
            if (!this.saved)
            {
                DialogResult result = MessageBox.Show("¿Desea guardar los último
               cambios realizados en el proyecto que ya estaba abierto?",
               "WebTALISMAN",
                MessageBoxButtons.YesNoCancel, MessageBoxIcon.Exclamation);
                if (result == DialogResult.Yes)
                {
                    saveToolStripMenuItem_Click(sender, e);
                    execute = true;
                }
                else if (result == DialogResult.No) execute = true;
                //Si es cancel, no se hace nada
            }
            else execute = true;

            if (execute)
            {
                XmlDocument xml = new XmlDocument();
                openFileDialog1.ShowDialog();
                string fileName = openFileDialog1.FileName;

                //Se limpian los parámetros almacenados
                this.param.Clear();

                if (fileName != "")
                {
                    xml.Load(fileName);
                    XmlElement element = xml.DocumentElement;

                    //Se cargan parámetros del proyecto
                    this.param.Add("sgbd",
element.SelectSingleNode("sgbd").InnerText);
                      this.param.Add("platform",
               element.SelectSingleNode("platform").InnerText);
```

```
                    this.param.Add("projectName",
                element.SelectSingleNode("projectName").InnerText);
                    platformComboBox.SelectedItem =
                element.SelectSingleNode("platform").InnerText;
                    SGBDComboBox.SelectedItem =
element.SelectSingleNode("sgbd").InnerText;

                    //Se abre un projecto
                    this.ProjectIsOpened("Proyecto abierto");

                    this.Text = element.SelectSingleNode("projectName").InnerText +
                    " - WebTALISMAN";  //Título de la aplicación
                    this.titleLabel.Text = "Proyecto '" +
                    element.SelectSingleNode("projectName").InnerText + "'";
                    //Título de la parte de configuración
                }
            }
        }

        /// <summary>
        /// Se abre el formulario para crear un nuevo proyecto
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void newProjectToolStripMenuItem_Click(object sender, EventArgs e)
        {
            bool execute = false;
            createPSMButton.Enabled = false;
            createCODEButton.Enabled = false;

            //Si hay un projecto nuevo que ha sufrido cambios se da opción a
guardarlos
            if (!this.saved)
            {
                DialogResult result = MessageBox.Show("¿Desea guardar los último
               cambios realizados en el proyecto que se estaba abierto?",
               "WebTALISMAN",
                MessageBoxButtons.YesNoCancel, MessageBoxIcon.Exclamation);
                if (result == DialogResult.Yes)
                {
                    saveToolStripMenuItem_Click(sender, e);
                    execute = true;
                }
                else if (result == DialogResult.No) execute = true;
                //Si es cancel, no se hace nada
            }
            else execute = true;

            if (execute)
            {
                NewProjectForm form = new NewProjectForm();
                form.ShowDialog();
                string fileName = form.getNewFile();
                //Se limpian los parámetros almacenados
                this.param.Clear();

                //Se comprueba si el nombre del nuevo proyecto se ha introducido
                if (fileName == null) { }
                //Se comprueba que el nombre del nuevo projecto no esté repetido
                 else if (!File.Exists(Application.StartupPath + "\\Projects\\" +
               fileName + ".xml"))
                {
                    //XML
                   XmlTextWriter writer = new XmlTextWriter(Application.StartupPath
               + "\\Projects\\" + fileName + ".xml", System.Text.Encoding.UTF8);
                    //Usa indentación por legibilidad
                    writer.Formatting = Formatting.Indented;
                    //Escribe la declaración del XML
                    writer.WriteStartDocument();
                    //Escribe el elemento raiz
                    writer.WriteStartElement("WebTALISMAN");
                    //Escribe los elementos dentro de sus etiquetas
```

```
                writer.WriteElementString("sgbd", "Microsoft SQL Server 2005");
                writer.WriteElementString("platform", "ASP.NET 2.0");
                writer.WriteElementString("projectName", fileName);
                writer.WriteEndElement();
                writer.Flush();
                writer.Close();

                //Se cargan parámetros del proyecto
                this.param.Add("sgbd", "Microsoft SQL Server 2005");
                this.param.Add("platform", "ASP.NET 2.0");
                this.param.Add("projectName", fileName);

                //Se abre un projecto nuevo
                this.ProjectIsOpened("Proyecto creado");

                this.Text = fileName + " - WebTALISMAN";  //Título de la
aplicación
                  this.titleLabel.Text = "Proyecto '" + fileName + "'"; //Título
                  de la parte de configuración

                //Ahora se crean las carpetas necesarias para el trabajar con el
proyecto
            if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName);
                 if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Log\\"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Log\\");
                 if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Model\\"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Model\\");
                 if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Model\\PIM"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Model\\PIM");
                if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Model\\PIM\\Data"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Model\\PIM\\Data");
                 if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Model\\PIM\\Data\\Media"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Model\\PIM\\Data\\Media");
                 if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Model\\PIM\\Data\\XSLT"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Model\\PIM\\Data\\XSLT");
                 if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Model\\PSM"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Model\\PSM");
                 if (!Directory.Exists(Application.StartupPath + "\\Projects\\" +
            fileName + "\\Solution\\"))
                      Directory.CreateDirectory(Application.StartupPath +
            "\\Projects\\" + fileName + "\\Solution\\");

                 platformComboBox.SelectedItem = "ASP.NET 2.0";
                 SGBDComboBox.SelectedItem = "Microsoft SQL Server 2005";
              }
              else
              {
                  MessageBox.Show("El proyecto ya estaba creado", "Error creando
            el proyecto",
                      MessageBoxButtons.OK, MessageBoxIcon.Error);
                 toolStripStatusLabel1.Text = "Proyecto no creado";
                 toolStripStatusLabel1.Visible = true;
              }
          }
      }
```

```
        /// <summary>
        /// Se cierra un proyecto ha sido abierto previamente
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void closeProjectToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Si no se ha guardado la última versión se advierte de ello
            if (!this.saved)
            {
                DialogResult result = MessageBox.Show("¿Desea guardar los último
               cambios realizados en el proyecto?", "WebTALISMAN",
                MessageBoxButtons.YesNoCancel, MessageBoxIcon.Exclamation);
                if (result == DialogResult.Yes) saveToolStripMenuItem_Click(sender,
e);
                else if (result != DialogResult.Cancel)
                {
                    //Se limpian los parámetros almacenados
                    this.param.Clear();

                    //Se cierra el proyecto
                    this.ProjectIsClosed("Proyecto cerrado");
                    this.Text = "WebTALISMAN";
                }
                //Si se cancela no se hace nada
            }
            else
            {
                this.ProjectIsClosed("Proyecto cerrado"); //Si no hay que guardar,
               se cierra sin más
                this.Text = "WebTALISMAN";  //Título de la aplicación
            }
        }

        /// <summary>
        /// Se guarda un proyecto que está abierto
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Se guardan parámetros del proyecto
            XmlDocument xml = new XmlDocument();
               xml.Load(Application.StartupPath + "\\Projects\\" +
       this.param["projectName"] + ".xml");
            XmlElement element = xml.DocumentElement;
            element.SelectSingleNode("sgbd").InnerText = this.param["sgbd"];
            element.SelectSingleNode("platform").InnerText = this.param["platform"];
             xml.Save(Application.StartupPath + "\\Projects\\" +
       this.param["projectName"] + ".xml");
            this.Text = this.param["projectName"] + " - WebTALISMAN";  //Título de
la aplicación
            toolStripStatusLabel1.Text = "Proyecto guardado";
            toolStripStatusLabel1.Visible = true;
            this.saved = true;
        }

        /// <summary>
        /// Información sobre la aplicación
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {
               MessageBox.Show("Prototipo de TALISMAN", "Acerca de WebTALISMAN");
        }

        /// <summary>
        /// Evento de cambio en la selección de la plataforma
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
```

```
        private void platformComboBox_SelectedValueChanged(object sender, EventArgs
e)
        {
                this.param["platform"] = platformComboBox.Text;
                 this.Text = this.param["projectName"] + "* - WebTALISMAN";
               //Título de la aplicación
                this.saved = false;
        }

        /// <summary>
        /// Evento de cambio en la selección del SGBD
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void SGBDComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
                this.param["sgbd"] = SGBDComboBox.Text;
                 this.Text = this.param["projectName"] + "* - WebTALISMAN";
               //Título de la aplicación
                this.saved = false;
        }

        /// <summary>
         /// Evento que aparece al cerrar la aplicación permitiendo así guardar los
       cambios    realizados
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            //Si no se ha guardado la última versión se advierte de ello
            if (!this.saved)
            {
                DialogResult result = MessageBox.Show("¿Desea guardar los último
               cambios realizados en el proyecto?", "WebTALISMAN",
                MessageBoxButtons.YesNoCancel, MessageBoxIcon.Exclamation);
                if (result == DialogResult.Yes) saveToolStripMenuItem_Click(sender,
e);
                else if (result == DialogResult.Cancel) e.Cancel = true;
                //Si no, no se hace nada
            }
        }

        /// <summary>
        /// Genera el PSM a parir del PIM
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void createPSMButton_Click(object sender, EventArgs e)
        {
            this.generator.Execute();
            this.generator.ExecutePIMtoPIM();
            this.generator.ExecutePIMtoPSM();
            toolStripStatusLabel1.Text = "PSM generado";
            toolStripStatusLabel1.Visible = true;
            openModelsButton.Enabled = false;
            createPSMButton.Enabled = false;
            createCODEButton.Enabled = true;
        }

        /// <summary>
        /// Genera el código a partir del PSM
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void createCODEButton_Click(object sender, EventArgs e)
        {
            this.generator.ExecutePSMtoPSM();
            this.generator.ExecutePSMtoCODE();
            this.generator.Delete();
            toolStripStatusLabel1.Text = "Código generado";
            toolStripStatusLabel1.Visible = true;
            openModelsButton.Enabled = true;
```

```
        createPSMButton.Enabled = false;
        createCODEButton.Enabled = false;
           //Se ponen todos los checked sin marcar para quedar preparados para
           una nueva carga de modelos
        checkBoxClasses.Checked = false;
        checkBoxRelationalObjects.Checked = false;
        checkBoxFragments.Checked = false;
        checkBoxNavigation.Checked = false;
        checkBoxXMLSchemas.Checked = false;
        checkBoxXLink.Checked = false;
        checkBoxUsersPIM.Checked = false;
        checkBoxUsersPSM.Checked = false;
        checkBoxWebService.Checked = false;
        checkBoxWebServiceClientPIM.Checked = false;
        checkBoxWebServiceClientPSM.Checked = false;
        checkBoxWSDL.Checked = false;
    }

    /// <summary>
    /// Abre los modelos (PIM)
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void openModelsButton_Click(object sender, EventArgs e)
    {
        this.generator = new GeneratorASPNET2_0(this.param);
        if (this.generator.Parser() == true)
        {
            labelLoaded.Visible = false;
            String[] models;
            models = this.generator.OpenModels();
            foreach (String model in models)
            {
                switch (model)
                {
                    //CONTENIDO
                    case "classes":
                        checkBoxClasses.Checked = true;
                        break;
                    case "relationalObjects":
                        checkBoxRelationalObjects.Checked = true;
                        break;
                    //APLICACIÓN WEB
                    case "fragments":
                        checkBoxFragments.Checked = true;
                        break;
                    case "navigation":
                        checkBoxNavigation.Checked = true;
                        break;
                    case "usersPIM":
                        checkBoxUsersPIM.Checked = true;
                        break;
                    case "XMLSchemas":
                        checkBoxXMLSchemas.Checked = true;
                        break;
                    case "XLink":
                        checkBoxXLink.Checked = true;
                        break;
                    case "usersPSM":
                        checkBoxUsersPSM.Checked = true;
                        break;
                    //FUNCIONALIDAD
                    case "webService":
                        checkBoxWebService.Checked = true;
                        break;
                    case "webServiceClientsPIM":
                        checkBoxWebServiceClientPIM.Checked = true;
                        break;
                    case "webServiceClientsPSM":
                        checkBoxWebServiceClientPSM.Checked = true;
                        break;
                    case "WSDL":
```

```
                        checkBoxWSDL.Checked = true;
                        break;
                    }
                }
                toolStripStatusLabel1.Text = "Modelos abiertos";
                toolStripStatusLabel1.Visible = true;
                openModelsButton.Enabled = false;
                createPSMButton.Enabled = true;
                createCODEButton.Enabled = false;
            }
            else
            {
                labelLoaded.Visible = true;
            }
        }
        #endregion

        #region "Methods"

        /// <summary>
        /// Constructor del formulario principal
        /// </summary>
        public MainForm()
        {
            InitializeComponent();
            this.param = new StringDictionary();
            this.saved = true;
        }

        /// <summary>
        /// Cuando el proyecto está cerrado se inicializa el estado del formulario
        /// </summary>
        /// <param name="StatusLabelText">Texto para indicar en la barra de estado
       información</param>
        public void ProjectIsClosed(string StatusLabelText)
        {
            //Barra de estado
            toolStripStatusLabel1.Text = StatusLabelText;
            toolStripStatusLabel1.Visible = true;

            //Menús
            closeProjectToolStripMenuItem.Enabled = false;
            saveToolStripMenuItem.Enabled = false;
            tabControl1.Visible = false;
            this.saved = true;
        }

        /// <summary>
        /// Cuando el proyecto está abierto se cambia la información mostrada en la
       ventana del formulario
        /// </summary>
       /// <param name="StatusLabelText">Texto para indicar en la barra de estado
       información</param>
        public void ProjectIsOpened(string StatusLabelText)
        {
            //Barra de estado
            toolStripStatusLabel1.Text = StatusLabelText;
            toolStripStatusLabel1.Visible = true;

            //Menús
            closeProjectToolStripMenuItem.Enabled = true;
            saveToolStripMenuItem.Enabled = true;
            tabControl1.Visible = true;

            this.saved = true;
        }
        #endregion

    }
}
```

Model.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.Windows.Forms;
using System.IO;

namespace WebTALISMAN.Models
{
    /// <summary>
    /// Parcialmente implementada: será la clase que se encargue de convertir los
XMI de   entrada en un XML legible para la aplicación
    /// </summary>
    class Model
    {
        /// <summary>
        /// Carga los modelos
        /// </summary>
        /// <param name="fileName">Archivo de entrada</param>
        /// <returns></returns>
        public bool LoadModels(string fileName)
        {
            bool ok = false;
            XmlDocument xml = new XmlDocument();
            xml.Load(fileName);
            XmlElement root = xml.DocumentElement;
            string version = root.GetAttribute("xmi.version");
            switch (version)
            {
                case "1.0":
                    ok = this.XMI_1_0(xml);
                    break;
                case "1.2":
                    ok = this.XMI_1_2(xml);
                    break;
                default:
                        MessageBox.Show("La version del archivo XMI no se reconoce.
              " + version, "Error abriendo el modelo",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                    break;
            }
            return ok;
        }

        /// <summary>
        /// Determina si un documento XML es XMI 1.0 válido y lo carga
        /// </summary>
        /// <param name="xml">Documento XMI</param>
        /// <returns></returns>
        public bool XMI_1_0(XmlDocument xml)
        {
            bool ok = true;

            MessageBox.Show("CARGADO 1.0");
            return ok;
        }

        /// <summary>
        /// Determina si un documento XML es XMI 1.2 válido y lo carga
        /// </summary>
        /// <param name="xml">Documento XMI</param>
        /// <returns></returns>
        public bool XMI_1_2(XmlDocument xml)
        {
            bool ok = true;

            MessageBox.Show("CARGADO 1.2");
            return ok;
        }    }}
```

Utils.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace WebTALISMAN
{
    /// <summary>
    /// Utilidades que se utilizan
    /// </summary>
    class Utils
    {
        /// <summary>
       /// Copia recursiva de todas los archivos y subdirectorios desde un origen
       hasta un destino
        /// </summary>
        /// <param name="strSourceDir">Directorio origen</param>
        /// <param name="strDestDir">Directorio destino</param>
        /// <param name="bolRecursive">True si se utiliza recursividad</param>
       public static void RecursiveCopyFiles(String strSourceDir, String strDestDir,
       Boolean bolRecursive)
        {
            try
            {
                //Add trailing separators to the supplied paths if they don't exist.
                if
(!strSourceDir.EndsWith(System.IO.Path.DirectorySeparatorChar.ToString()))
                {
                    strSourceDir +=
Convert.ToString(System.IO.Path.DirectorySeparatorChar);
                }

                if
(!strDestDir.EndsWith(System.IO.Path.DirectorySeparatorChar.ToString()))
                {
                    strDestDir +=
Convert.ToString(System.IO.Path.DirectorySeparatorChar);
                }

                //Recursive switch to continue drilling down into dir structure.
                Int32 intposSep;
                Int32 i;
                if (bolRecursive)
                {
                    //Get a list of directories from the current parent.
                    String[] astrDirs;
                    astrDirs = System.IO.Directory.GetDirectories(strSourceDir);
                    for (i = 0; i < astrDirs.Length ; i++)
                    {
                        //Get the position of the last separator in the current
path.
                        intposSep = astrDirs[i].LastIndexOf("\\");
                        //Get the path of the source directory.
                        String strDir;
                        strDir = astrDirs[i].Substring((intposSep + 1),
                      astrDirs[i].Length - (intposSep + 1));
                        //Create the new directory in the destination directory.
                        System.IO.Directory.CreateDirectory(strDestDir + strDir);
                        //Since we are in recursive mode, copy the children also
                        RecursiveCopyFiles(astrDirs[i], (strDestDir + strDir),
bolRecursive);
                    }
                }
                //Get the files from the current parent.
                String[] astrFiles;
                astrFiles = System.IO.Directory.GetFiles(strSourceDir);
                for (i = 0; i != astrFiles.Length; i++)
                {
                    //Get the position of the trailing separator.
                    intposSep = astrFiles[i].LastIndexOf("\\");
                    //Get the full path of the source file.
```

```
                    String strFile;
                      strFile = astrFiles[i].Substring((intposSep + 1),
                astrFiles[i].Length - (intposSep + 1));
                    //Copy the file.
                    System.IO.File.Copy(astrFiles[i], (strDestDir + strFile));
                  }
            }
            catch (Exception ex)
            {
                System.Windows.Forms.MessageBox.Show(ex.Message);
            }
        }

    } //Class
}
```

Platforms/ASPNET2_0/Templates/Associations.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class Associations : Template
    {
        private static string[] _phs
            = new string[] { "$Target!", "$Multiplicity!", "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public Associations()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\Associations.tpl"), _phs)
        {
        }

        public string Target;
        public string Multiplicity;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Target!"] = Target;
            if (Multiplicity != "1") base["$Multiplicity!"] = "[]";
            base["$Name!"] = "Id_" + Target;
        }
    }
}
```

Platforms/ASPNET2_0/Templates/Class.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class Class : Template
    {
        private static string[] _phs
               = new string[] { "$Visibility!", "$Inherit!", "$Final!",
               "$Abstract!", "$Name!", "$Properties*", "$Associations*",
               "$GetSetProperties*", "$Methods*", "$StringSlices*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public Class()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\Class.tpl"), _phs)
        {
            Properties = new TemplateCollection("\n");
            Properties.FirstLineTabs = 0;
            Properties.Tabs = 2;
            Associations = new TemplateCollection("\n");
            Associations.FirstLineTabs = 0;
            Associations.Tabs = 2;
            GetSetProperties = new TemplateCollection("\n");
            GetSetProperties.FirstLineTabs = 0;
            GetSetProperties.Tabs = 2;
            Methods = new TemplateCollection("\n");
            Methods.FirstLineTabs = 0;
            Methods.Tabs = 2;
            StringSlices = new TemplateCollection("\n");
            StringSlices.FirstLineTabs = 0;
            StringSlices.Tabs = 0;
        }

        public string Visibility;
        public string Inherit;
        public string Final;
        public string Abstract;
        public string Name;
        public readonly TemplateCollection Properties;
        public readonly TemplateCollection Associations;
        public readonly TemplateCollection GetSetProperties;
        public readonly TemplateCollection Methods;
        public readonly TemplateCollection StringSlices;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Visibility!"] = Visibility;
            if (Inherit != "") base["$Inherit!"] = ": " + Inherit;
            if (Final == "true") base["$Final!"] = " static";
            if (Abstract == "true") base["$Abstract!"] = " abstract";
            base["$Name!"] = Name;
            base["$Properties*"] = Properties.ToString();
            base["$Associations*"] = Associations.ToString();
            base["$GetSetProperties*"] = GetSetProperties.ToString();
            base["$Methods*"] = Methods.ToString();
            base["$StringSlices*"] = StringSlices.ToString();
        }    }}
```

Class.tpl

```
/// CSharpClass File generated by WebTALISMAN.
using System;

namespace Solution.Business
{
        /// <summary>
        /// Summary description for $Name!
        /// </summary>
        $Visibility!$Abstract!$Final! class $Name! $Inherit!
        {
                //Properties
                $Properties*

                //Properties because of associations
                $Associations*

                /// <summary>
                /// Constructor
                /// </summary>
                public $Name!()
                {
                }

                $GetSetProperties*
                $Methods*
         /// <summary>
         /// Convert object to string
         /// </summary>
         /// <returns>Object serialized</returns>
         public override String ToString()
                {
             String strOutPut = "";
              $StringSlices*
             return strOutPut;
                }
        }
}
```

Platforms/ASPNET2_0/Templates/ClasssBF.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class ClassBF : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public ClassBF()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\ClassBF.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
```

```
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

ClasssBF.tpl

```
/// CSharpClass File generated by WebTALISMAN.
using System;
using System.Collections;
using Solution.Business;
using Solution.DataAccess;

namespace Solution.BusinessFacade
{
        /// <summary>
        /// Summary description for $Name!BF
        /// </summary>
    public class $Name!BF : BusinessFacadeBase
    {
        /// <summary>
        /// Constructor
        /// </summary>
        public $Name!BF()
        {
        }

        /// <summary>
        /// Save a $Name! object
        /// </summary>
        /// <param name="pobj$Name!">Object to save</param>
        public Int32 Save($Name! pobj$Name!)
        {
            VerifyIsLoggedIn();
            DataAccessBase objDA = new DataAccessBase();
            return objDA.Save(pobj$Name!);
        }

        /// <summary>
        /// Get a $Name! object list
        /// </summary>
        /// <returns>A $Name! object list</returns>
        public IList Get$Name!()
        {
            VerifyIsLoggedIn();
            DataAccessBase objDA = new DataAccessBase();
            return objDA.Get(typeof($Name!));
        }

        /// <summary>
        /// Get a $Name! object
        /// </summary>
        /// <param name="pintId">Id of the object</param>
        /// <returns>A $Name! object</returns>
        public $Name! Get$Name!(Int32 pintId)
        {
            VerifyIsLoggedIn();
            DataAccessBase objDA = new DataAccessBase();
            return objDA.Get(typeof($Name!), pintId) as $Name!;
        }

        /// <summary>
        /// Delete a $Name! object
        /// </summary>
        /// <param name="pobj$Name!">Object to delete</param>
        public void Delete($Name! pobj$Name!)
```

```
        {
            VerifyIsLoggedIn();
            DataAccessBase objDA = new DataAccessBase();
            objDA.Delete(pobj$Name!);
        }
    }
}
```

Platforms/ASPNET2_0/Templates/ClassMultipleAssociation.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class ClassMultipleAssociation : Template
    {
        private static string[] _phs
            = new string[] { "$Name1!", "$Name2!", "$TypeId!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public ClassMultipleAssociation()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\ClassMultipleAssociation.tpl"),
               _phs)
        {
        }

        public string Name1;
        public string Name2;
        public string TypeId;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name1!"] = Name1;
            base["$Name2!"] = Name2;
            base["$TypeId!"] = TypeId;
        }
    }
}
```

ClassMultipleAssociation.tpl

```
/// CSharpClass File generated by WebTALISMAN.
using System;

namespace Solution.Business
{
        /// <summary>
        /// Summary description for $Name!
        /// </summary>
        public class $Name1!_$Name2!
        {
                //Properties
                private $TypeId! id;

                //Properties because of associations
                private $Name1! Id_$Name1!;
```

```
                private $Name2! Id_$Name2!;

                /// <summary>
                /// Constructor
                /// </summary>
                public $Name1!_$Name2!()
                {
                }


                /// <summary>
                /// Get or Set the Id property
                /// </summary>
                public $TypeId! pId
                {
                    get { return id; }
                    set { id = value; }
                }

                /// <summary>
                /// Get or Set the Id_$Name1! property
                /// </summary>
                public $Name1! pId_$Name1!
                {
                    get { return Id_$Name1!; }
                    set { Id_$Name1! = value; }
                }

                /// <summary>
                /// Get or Set the Id_$Name2! property
                /// </summary>
                public $Name2! pId_$Name2!
                {
                    get { return Id_$Name2!; }
                    set { Id_$Name2! = value; }
                }


        }
}
```

Platforms/ASPNET2_0/Templates/ClassTest.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class ClassTest : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$ClassTestInput*", "$ClassTestInputModify*"
};

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public ClassTest()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\ClassTest.tpl"), _phs)
        {
            ClassTestInput = new TemplateCollection("\n");
            ClassTestInput.FirstLineTabs = 0;
            ClassTestInput.Tabs = 0;
            ClassTestInputModify = new TemplateCollection("\n");
            ClassTestInputModify.FirstLineTabs = 0;
```

```
            ClassTestInputModify.Tabs = 0;
        }

        public string Name;
        public readonly TemplateCollection ClassTestInput;
        public readonly TemplateCollection ClassTestInputModify;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$ClassTestInput*"] = ClassTestInput.ToString();
            base["$ClassTestInputModify*"] = ClassTestInputModify.ToString();
        }
    }
}
```

ClassTest.tpl

```
using System;
using System.Reflection;
using NUnit.Framework;
using Solution.Business;
using Solution.BusinessFacade;


namespace Solution.UnitTests
{
    [TestFixture]
    public class $Name!Test
    {
        private int mintId;

        [TestFixtureSetUp]
        public void OneTimeSetUp()
        {
            Console.Out.WriteLine("Running " + MethodBase.GetCurrentMethod());
        }

        [TestFixtureTearDown]
        public void OneTimeTearDown()
        {
            Console.Out.WriteLine("Running " + MethodBase.GetCurrentMethod());
        }

        /// <summary>
        /// Save a $Name!
        /// </summary>
        [Test]
        public void Save()
        {
            Console.Out.WriteLine("Running " + MethodBase.GetCurrentMethod());
            $Name!BF obj$Name!BF = new $Name!BF();
            $Name! obj$Name! = new $Name!();
             $ClassTestInput*
            mintId = obj$Name!BF.Save(obj$Name!);
            Console.Out.WriteLine(obj$Name!.ToString());
        }

        /// <summary>
        /// Get a $Name!
        /// </summary>
        [Test]
        public void Get()
        {
            Console.Out.WriteLine("Running " + MethodBase.GetCurrentMethod());
            $Name!BF obj$Name!BF = new $Name!BF();
```

```
            $Name! obj$Name! = obj$Name!BF.Get$Name!(Convert.ToInt32(mintId));
            Assert.IsNotNull(obj$Name!);
            Console.Out.WriteLine(obj$Name!.ToString());
        }

        /// <summary>
        /// Modify a $Name!
        /// </summary>
        [Test]
        public void Modify()
        {
            Console.Out.WriteLine("Running " + MethodBase.GetCurrentMethod());
            $Name!BF obj$Name!BF = new $Name!BF();
            $Name! obj$Name! = new $Name!();
            obj$Name!.pId = mintId;
            $ClassTestInputModify*
            obj$Name!BF.Save(obj$Name!);
            Get();
        }

        /// <summary>
        /// Delete a $Name!
        /// </summary>
        [Test]
        public void Delete()
        {
            Console.Out.WriteLine("Running " + MethodBase.GetCurrentMethod());
            $Name!BF obj$Name!BF = new $Name!BF();
            $Name! obj$Name! = new $Name!();
            obj$Name!.pId = mintId;
            obj$Name!BF.Delete(obj$Name!);
        }

    }
}
```

Platforms/ASPNET2_0/Templates/ClassTestInput.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class ClassTestInput : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Attribute!", "$Value!", "$Type!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public ClassTestInput()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\ClassTestInput.tpl"), _phs)
        {
        }

        public string Name, Attribute, Type;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            string uAttribute, first;
            uAttribute = "";
```

```
            first = Attribute[0].ToString();
            first = first.ToUpper(); //Covierto a mayúscula

            for (int i = 1; i < Attribute.Length; i++) //en este ciclo se coge la
            palabra menos la 1ra letra
            {
                uAttribute = uAttribute + Attribute[i];
            }

            uAttribute = first + uAttribute; //Se forma la palabra que es igual a la
            original sólo que con la primera letra en mayúsculas
            base["$Name!"] = Name;
            base["$Attribute!"] = uAttribute;
            switch (Type)
            {
                case "int":
                    base["$Value!"] = Convert.ToString(10);
                    break;
                case "string":
                    base["$Value!"] = "\"TEXT\"";
                    break;
            }
        }
    }
}
```

ClassTestInput.tpl

```
obj$Name!.p$Attribute! = $Value!;
```

Platforms/ASPNET2_0/Templates/ClassTestInputModify.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class ClassTestInputModify : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Attribute!", "$Value!", "$Type!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public ClassTestInputModify()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\ClassTestInput.tpl"), _phs)
        {
        }

        public string Name, Attribute, Type;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            string uAttribute, first;
            uAttribute = "";
            first = Attribute[0].ToString();
            first = first.ToUpper(); //Covierto a mayúscula
```

```
            for (int i = 1; i < Attribute.Length; i++) //en este ciclo se coge la
            palabra menos la 1ra letra
            {
                uAttribute = uAttribute + Attribute[i];
            }

            uAttribute = first + uAttribute; //Se forma la palabra que es igual a la
            original sólo que con la primera letra en mayúsculas
            base["$Name!"] = Name;
            base["$Attribute!"] = uAttribute;
            switch (Type)
            {
                case "int":
                    base["$Value!"] = Convert.ToString(20);
                    break;
                case "string":
                    base["$Value!"] = "\"NEW TEXT\"";
                    break;
            }
        }
    }
}
```

Platforms/ASPNET2_0/Templates/Compiles.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class Compiles : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public Compiles()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\Compiles.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

Compiles.tpl

```
<Compile Include="$Name!.cs" />
```

Platforms/ASPNET2_0/Templates/DataAccessBase.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class DataAccessBase : Template
    {
        private static string[] _phs
               = new string[] { "$hibernate.connection.provider!",
               "$hibernate.dialect!", "$hibernate.connection.driver_class!",
               "$hibernate.connection.connection_string!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public DataAccessBase()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\DataAccessBase.tpl"), _phs)
        {
        }

        public string Provider;
        public string Dialect;
        public string Driver;
        public string ConnectionString;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$hibernate.connection.provider!"] = Provider;
            base["$hibernate.dialect!"] = Dialect;
            base["$hibernate.connection.driver_class!"] = Driver;
            base["$hibernate.connection.connection_string!"] = ConnectionString;
        }
    }
}
```

DataAccessBase.tpl

```
using System;
using System.Collections;
using NHibernate;
using NHibernate.Cfg;
using NHibernate.Expression;
using log4net;

namespace Solution.DataAccess
{
    public class DataAccessBase
    {
        protected Configuration cfg;
        protected ISessionFactory factory;
        protected ISession session;
        protected ITransaction transaction;
        protected ILog log;

        /// <summary>
        /// Constructor
        /// </summary>
        public DataAccessBase()
```

```
        {
            log = log4net.LogManager.GetLogger("DataAccess.DataAccessBase");
            cfg = new Configuration();
            cfg.SetProperty("hibernate.connection.provider",
"$hibernate.connection.provider!");
            cfg.SetProperty("hibernate.dialect", "$hibernate.dialect!");
            cfg.SetProperty("hibernate.connection.driver_class",
"$hibernate.connection.driver_class!");
            cfg.SetProperty("hibernate.connection.connection_string",
"$hibernate.connection.connection_string!");
            cfg.AddAssembly("Solution.Business");
            factory = cfg.BuildSessionFactory();
        }

        /// <summary>
        /// Save a object
        /// </summary>
        /// <param name="pobj">Object to be saved</param>
        public Int32 Save(object pobj)
        {
            try
            {
                Int32 id;
                session = factory.OpenSession();
                transaction = session.BeginTransaction();
                session.SaveOrUpdate(pobj); //Tell NHibernate that this object
should be saved
                id = Convert.ToInt32(session.GetIdentifier(pobj));
                transaction.Commit(); //Commit all of the changes to the DB and
       close the ISession
                session.Close();
                log.Info("Object " + pobj.GetType().Name + " saved");
                return id;
            }
            catch (Exception ex)
            {
                log.Error(ex.Message);
                throw;
            }
        }

        /// <summary>
        /// Returns a list of items matching the type supplied
        /// </summary>
        /// <param name="type">Type of the object</param>
        /// <returns>List of the objects</returns>
        public IList Get(Type type)
        {
            return GetByType(type);
        }

        /// <summary>
        /// Get a object by id
        /// </summary>
        /// <param name="type">Type of the object</param>
        /// <param name="pintId">Id of the object</param>
        /// <returns>Object</returns>
        public object Get(Type type, Int64 pintId)
        {
            try
            {
                session = factory.OpenSession();
                transaction = session.BeginTransaction();
                ICriteria crit = session.CreateCriteria(type);
                crit.Add(Expression.Eq("id", pintId));
                IList list = crit.List();
                transaction.Commit();
                session.Close();
                if (list == null || list.Count < 1)
                {
                    return null;
                }
                else
```

```
                {
                    return list[0]; //The last element
                }
            }
            catch (Exception ex)
            {
                log.Error(ex.Message);
                throw;
            }
        }

        /// <summary>
        /// Get objects by a type
        /// </summary>
        /// <param name="type">Type of the objects</param>
        /// <returns>List of the objects</returns>
        private IList GetByType(Type type)
        {
            IList items = null;
            try
            {
                session = factory.OpenSession();
                transaction = session.BeginTransaction();
                items = session.CreateCriteria(type).List();
                 transaction.Commit(); //Commit all of the changes to the DB and
               close the ISession
                session.Close();
                return items;
            }
            catch (Exception ex)
            {
                log.Error(ex.Message);
                throw;
            }
        }

        /// <summary>
        /// Delete a object
        /// </summary>
        /// <param name="pobj">Object to delete</param>
        public void Delete(object pobj)
        {
            try
            {
                session = factory.OpenSession();
                transaction = session.BeginTransaction();
                session.Delete(pobj);
                transaction.Commit(); //Commit all of the changes to the DB and
               close the ISession
                session.Close();
            }
            catch (Exception ex)
            {
                log.Error(ex.Message);
            }
        }

    }
}
```

Platforms/ASPNET2_0/Templates/EmbeddedResources.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class EmbeddedResources : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public EmbeddedResources()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\EmbeddedResources.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

Platforms/ASPNET2_0/Templates/GetSetProperties.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class GetSetProperties : Template
    {
        private static string[] _phs
            = new string[] { "$Visibility!", "$Type!", "$Name!", "$UName!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public GetSetProperties()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\GetSetProperties.tpl"), _phs)
        {
        }

        public string Visibility;
        public string Type;
        public string Name;
```

```
        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            string uName, first;
            uName = "";
            first = Name[0].ToString();
            first = first.ToUpper(); //Covierto a mayúscula

            for (int i = 1; i < Name.Length; i++) //en este ciclo se coge la palabra
            menos la 1ra letra
            {
                uName = uName + Name[i];
            }
                 uName = first + uName; //Se forma la palabra que es igual a la
               original sólo que con la primera letra en mayúsculas

            base["$Visibility!"] = Visibility;
            base["$Type!"] = " " + Type;
            base["$Name!"] = Name;
            base["$UName!"] = uName;
        }
    }
}
```

GetSetProperties.tpl

```
/// <summary>
/// Get or Set the $UName! property
/// </summary>
$Visibility!$Type! p$UName!
{
       get { return $Name!; }
       set { $Name! = value; }
}
```

Platforms/ASPNET2_0/Templates/HBM.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class HBM : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$TypeId!", "$HBMProperties*",
"$HBMAssociations*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public HBM()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\HBM.tpl"), _phs)
        {
            HBMProperties = new TemplateCollection("\n");
            HBMProperties.FirstLineTabs = 0;
            HBMProperties.Tabs = 2;
            HBMAssociations = new TemplateCollection("\n");
            HBMAssociations.FirstLineTabs = 0;
            HBMAssociations.Tabs = 2;
```

```
        }

        public string Name;
        public string TypeId;
        public readonly TemplateCollection HBMProperties;
        public readonly TemplateCollection HBMAssociations;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$TypeId!"] = TypeId;
            base["$HBMProperties*"] = HBMProperties.ToString();
            base["$HBMAssociations*"] = HBMAssociations.ToString();
        }
    }
}
```

HBM.tpl

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  <class name="Solution.Business.$Name!, Solution.Business" table="$Name!">
    <id name="pId" column="Id" type="$TypeId!" unsaved-value="0">
      <generator class="identity"/>
    </id>
    $HBMProperties*
    $HBMAssociations*
  </class>
</hibernate-mapping>
```

Platforms/ASPNET2_0/Templates/HBMAssociations.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class HBMAssociations : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$UName!", "$More!", "$Relation!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public HBMAssociations()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\HBMAssociations.tpl"), _phs)
        {
        }

        public string Name;
        public string Relation;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            string uName, first;
            uName = "";
```

```
            first = Name[0].ToString();
            first = first.ToUpper(); //Covierto a mayúscula

            for (int i = 1; i < Name.Length; i++) //en este ciclo se coge la palabra
            menos la 1ra letra
            {
                uName = uName + Name[i];
            }
                uName = first + uName; //Se forma la palabra que es igual a la
               original sólo que con la primera letra en mayúsculas

            base["$Name!"] = Name;
            base["$UName!"] = uName;
            switch (Relation)
            {
                case "one-to-one":
                    base["$More!"] = "constrained=\"true\"";
                    break;
                case "one-to-many":
                    base["$More!"] = "column=\"" + Name + "\" cascade=\"all\"";
                    break;
                case "many-to-one":
                    base["$More!"] = "column=\"" + Name + "\" cascade=\"all\"";
                    break;
                case "many-to-many":
                    base["$More!"] = "column=\"" + Name + "\" cascade=\"all\"";
                    break;
            }
            base["$Relation!"] = Relation;
        }
    }
}
```

HBMAssociations.tpl

```
<$Relation! name="p$UName!" $More! />
```

Platforms/ASPNET2_0/Templates/HBMMultipleAssociations.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class HBMMultipleAssociation : Template
    {
        private static string[] _phs
            = new string[] { "$Name1!", "$Name2!", "$TypeId!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public HBMMultipleAssociation()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\HBMMultipleAssociation.tpl"),
               _phs)
        {
        }

        public string Name1;
        public string Name2;
        public string TypeId;
```

```
        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name1!"] = Name1;
            base["$Name2!"] = Name2;
            base["$TypeId!"] = TypeId;
        }
    }
}
```

HBMMultipleAssociations.tpl

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  <class name="Solution.Business.$Name1!_$Name2!, Solution.Business"
table="$Name1!_$Name2!">
    <id name="pId" column="Id" type="Int32" unsaved-value="0">
      <generator class="identity"/>
    </id>
    <many-to-one name="pId_$Name1!" column="Id_$Name1!" cascade="all"/>
    <many-to-one name="pId_$Name2!" column="Id_$Name2!" cascade="all"/>
  </class>
</hibernate-mapping>
```

Platforms/ASPNET2_0/Templates/HBMProperties.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class HBMProperties : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$UName!", "$Type!", "$Length!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public HBMProperties()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\HBMProperties.tpl"), _phs)
        {
        }

        public string Name;
        public string UName;
        public string Type;
        public string Length;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            string uName, first;
            uName = "";
            first = Name[0].ToString();
            first = first.ToUpper(); //Covierto a mayúscula

            for (int i = 1; i < Name.Length; i++) //en este ciclo se coge la palabra
            menos la 1ra letra
```

```
            {
                uName = uName + Name[i];
            }
                uName = first + uName; //Se forma la palabra que es igual a la
               original sólo que con la primera letra en mayúsculas

            base["$Name!"] = Name;
            base["$UName!"] = uName;
            base["$Type!"] = Type;
            base["$Length!"] = "";
        }
    }
}
```

HBMProperties.tpl

```
<property name="p$UName!" column="$Name!" type="$Type!" $Length! />
```

Platforms/ASPNET2_0/Templates/Links.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class Links : Template
    {
        private static string[] _phs
            = new string[] { "$LinksPages*", "$LinksMethods*"};

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public Links()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\Links.tpl"), _phs)
        {
            Pages = new TemplateCollection("\n");
            Pages.FirstLineTabs = 3;
            Pages.Tabs = 3;
            Methods = new TemplateCollection("\n");
            Methods.FirstLineTabs = 1;
            Methods.Tabs = 1;
        }

        public readonly TemplateCollection Pages;
        public readonly TemplateCollection Methods;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$LinksPages*"] = Pages.ToString();
            base["$LinksMethods*"] = Methods.ToString();
        }
    }
}
```

Links.tpl

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections;
using Solution.Business;
using Solution.BusinessFacade;

/// <summary>
/// Summary description for Links
/// </summary>
public class Links
{
    Hashtable mobjQuery;
    public Links()
    {
        //
        // TODO: Add constructor logic here
        //
    }

    public String Transform(String pstrPage, String pstrWebPage, Hashtable
pobjQuery)
    {
        mobjQuery = pobjQuery;
        String strResultPage = "";
        switch (pstrWebPage)
        {
            $LinksPages*
            default:
                strResultPage = pstrPage; //There is no transform
                break;
        }
        if (strResultPage.Contains("menus />")) //Headers
        {
            String strMenu = "<link href=\"Hypertext/XSLT/StyleSheetTALISMAN.css\"
rel=\"stylesheet\" type=\"text/css\" />";
            strMenu += "<script type=\"text/javascript\"
src=\"Hypertext/XSLT/JavaScriptTALISMAN.js\"></script>";
            strResultPage = strResultPage.Replace("<menus />", strMenu);
        }
        return strResultPage;
    }

$LinksMethods*
}
```

Platforms/ASPNET2_0/Templates/LinksMethod1Multiple.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class LinksMethod1Multiple : Template
    {
        private static string[] _phs
            = new string[] { "$From!", "$To!", "$Title!", "$Num!" };
```

```
        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public LinksMethod1Multiple()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\LinksMethod1Multiple.tpl"), _phs)
        {
        }

        public string From;
        public string To;
        public string Title;
        public string Num;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$From!"] = From;
            base["$To!"] = To;
            base["$Title!"] = Title;
            base["$Num!"] = Num;
        }
    }
}
```

LinksMethod1Multiple.tpl

```
if (pstrPage.Contains("<$From!.$To! />"))
{
    if (Convert.ToString(mobjQuery["$To!"]) != "")
    {
            String strMenu = "<div class=\"suckerdiv\">";
        strMenu += "<ul id=\"TALISMANmenu$Num!\">";
        strMenu += "<li><a href='$To!.aspx?$To!=" +
       Convert.ToString(mobjQuery["Id"]) + "&Id=" +
       Convert.ToString(mobjQuery["$To!"]) + "'>$Title!</a></li>";
        strMenu += "</li>";
        strMenu += "</ul>";
        strMenu += "</div>";
        strResultPage = strResultPage.Replace("<$From!.$To! />", strMenu);
       }
}
```

Platforms/ASPNET2_0/Templates/LinksMethodFoundNull.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class LinksMethodFoundNull : Template
    {
        private static string[] _phs
            = new string[] { "$From!", "$To!", "$Title!", "$Num!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public LinksMethodFoundNull()
```

```
                : base(new System.IO.FileInfo(Application.StartupPath +
                "\\Platforms\\ASPNET2_0\\Templates\\LinksMethodFoundNull.tpl"), _phs)
        {
        }

        public string From;
        public string To;
        public string Title;
        public string Num;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$From!"] = From;
            base["$To!"] = To;
            base["$Title!"] = Title;
            base["$Num!"] = Num;
        }
    }
}
```

LinksMethodFoundNull.tpl

```
if (pstrPage.Contains("<$From!.$To! />"))
{
        String strMenu = "<div class=\"suckerdiv\">";
    strMenu += "<ul id=\"TALISMANmenu$Num!\">";
    strMenu += "<li><a href=\"$To!.aspx?$From!=" + Convert.ToString(mobjQuery["Id"])
+   "\">$Title!</a>";
    strMenu += "</li>";
    strMenu += "</ul>";
    strMenu += "</div>";
    strResultPage = strResultPage.Replace("<$From!.$To! />", strMenu);
}
```

Platforms/ASPNET2_0/Templates/LinksMethodMultiple1.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class LinksMethodMultiple1 : Template
    {
        private static string[] _phs
            = new string[] { "$From!", "$To!", "$Title!", "$Index!", "$Num!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public LinksMethodMultiple1()
                : base(new System.IO.FileInfo(Application.StartupPath +
                "\\Platforms\\ASPNET2_0\\Templates\\LinksMethodMultiple1.tpl"), _phs)
        {
        }

        public string From;
        public string To;
        public string Title;
        public string Index;
        public string Num;
```

```
        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$From!"] = From;
            base["$To!"] = To;
            base["$Title!"] = Title;
            base["$Index!"] = Index;
            base["$Num!"] = Num;
        }
    }
}
```

LinksMethodMultiple1.tpl

```
if (pstrPage.Contains("<$From!.$To! />"))
{
        String strMenu = "<div class=\"suckerdiv\">";
    strMenu += "<ul id=\"TALISMANmenu$Num!\">";
    strMenu += "<li><a href=\"$To!.aspx\">$Title!</a>";
    strMenu += "<ul>";
    $To!BF obj$To!BF = new $To!BF();
    IList obj$To!List = obj$To!BF.Get$To!();
    int i = 0;
    foreach ($To! obj$To! in obj$To!List)
    {
        if (obj$To!.pId_$From!.pId == Convert.ToInt32(mobjQuery["Id"]))
        {
               strMenu += "<li><a href='$To!.aspx?$From!=" +
               Convert.ToString(mobjQuery["Id"]) + "&Id=" + obj$To!.pId + "'>" +
               obj$To!.p$Index! + "</a></li>";
               i++;
        }
    }
    strMenu += "</ul>";
    strMenu += "</li>";
    strMenu += "</ul>";
    strMenu += "</div>";
    if (i == 0) strMenu = "";
    strResultPage = strResultPage.Replace("<$From!.$To! />", strMenu);
}
```

Platforms/ASPNET2_0/Templates/LinksMethodMultipleMultiple.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class LinksMethodMultipleMultiple : Template
    {
        private static string[] _phs
            = new string[] { "$From!", "$To!", "$Title!", "$Index!", "$Num!",
"$Table!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public LinksMethodMultipleMultiple()
```

```
            : base(new System.IO.FileInfo(Application.StartupPath +
            "\\Platforms\\ASPNET2_0\\Templates\\LinksMethodMultipleMultiple.tpl")
            , _phs)
        {
        }

        public string From;
        public string To;
        public string Title;
        public string Index;
        public string Num;
        public string Table;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$From!"] = From;
            base["$To!"] = To;
            base["$Title!"] = Title;
            base["$Index!"] = Index;
            base["$Num!"] = Num;
            base["$Table!"] = Table;
        }
    }
}
```

LinksMethodMultipleMultiple.tpl

```
if (pstrPage.Contains("<$From!.$To! />"))
{
    String strMenu = "<div class=\"suckerdiv\">";
    strMenu += "<ul id=\"TALISMANmenu$Num!\">";
    strMenu += "<li><a href=\"$To!.aspx\">$Title!</a>";
    strMenu += "<ul>";
    $Table!BF obj$Table!BF = new $Table!BF();
    IList obj$Table!List = obj$Table!BF.Get$Table!();
    int i = 0;
    foreach ($Table! obj$Table! in obj$Table!List)
    {
        if (obj$Table!.pId_$From!.pId == Convert.ToInt32(mobjQuery["Id"]))
        {
            strMenu += "<li><a href='$To!.aspx?$From!=" +
            Convert.ToString(mobjQuery["Id"]) + "&Id=" + obj$Table!.pId_$To!.pId
            + "'>" + obj$Table!.pId_$To!.p$Index! + "</a></li>";
             i++;
        }
    }
    strMenu += "</ul>";
    strMenu += "</li>";
    strMenu += "</ul>";
    strMenu += "</div>";
    if (i == 0) strMenu = "";
    strResultPage = strResultPage.Replace("<$From!.$To! />", strMenu);
}
```

Platforms/ASPNET2_0/Templates/LinksMethodNullFound.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
```

```
    public class LinksMethodNullFound : Template
    {
        private static string[] _phs
            = new string[] { "$From!", "$To!", "$Title!", "$Index!", "$Num!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public LinksMethodNullFound()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\LinksMethodNullFound.tpl"), _phs)
        {
        }

        public string From;
        public string To;
        public string Title;
        public string Index;
        public string Num;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$From!"] = From;
            base["$To!"] = To;
            base["$Title!"] = Title;
            base["$Index!"] = Index;
            base["$Num!"] = Num;
        }
    }
}
```

LinksMethodNullFound.tpl

```
if (pstrPage.Contains("<$From!.$To! />"))
{
    String strMenu = "<div class=\"suckerdiv\">";
    strMenu += "<ul id=\"TALISMANmenu$Num!\">";
    strMenu += "<li><a href=\"$To!.aspx\">$Title!</a>";
    strMenu += "<ul>";
    $To!BF obj$To!BF = new $To!BF();
    IList obj$To!List = obj$To!BF.Get$To!();
    int i = 0;
    foreach ($To! obj$To! in obj$To!List)
    {
       strMenu += "<li><a href='$To!.aspx?$From!="+
       Convert.ToString(mobjQuery["Id"]) +"&Id=" + obj$To!.pId + "'>" +
       obj$To!.p$Index! + "</a></li>";
       i++;
    }
    strMenu += "</ul>";
    strMenu += "</li>";
    strMenu += "</ul>";
    strMenu += "</div>";
    if (i == 0) strMenu = "";
    strResultPage = strResultPage.Replace("<$From!.$To! />", strMenu);
}
```

Platforms/ASPNET2_0/Templates/LinksMethods.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class LinksMethods : Template
    {
        private static string[] _phs
               = new string[] { "$Name!", "$LinksMethodNullFound*",
               "$LinksMethodFoundNull*", "$LinksMethodMultiple1*",
               "$LinksMethod1Multiple*", "$LinksMethodMultipleMultiple*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public LinksMethods()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\LinksMethods.tpl"), _phs)
        {
            LinksMethodNullFound = new TemplateCollection("\n");
            LinksMethodNullFound.FirstLineTabs = 1;
            LinksMethodNullFound.Tabs = 2;
            LinksMethodFoundNull = new TemplateCollection("\n");
            LinksMethodFoundNull.FirstLineTabs = 1;
            LinksMethodFoundNull.Tabs = 2;
            LinksMethodMultiple1 = new TemplateCollection("\n");
            LinksMethodMultiple1.FirstLineTabs = 1;
            LinksMethodMultiple1.Tabs = 2;
            LinksMethod1Multiple = new TemplateCollection("\n");
            LinksMethod1Multiple.FirstLineTabs = 1;
            LinksMethod1Multiple.Tabs = 2;
            LinksMethodMultipleMultiple = new TemplateCollection("\n");
            LinksMethodMultipleMultiple.FirstLineTabs = 1;
            LinksMethodMultipleMultiple.Tabs = 2;
        }

        public string Name;
        public readonly TemplateCollection LinksMethodNullFound;
        public readonly TemplateCollection LinksMethodFoundNull;
        public readonly TemplateCollection LinksMethodMultiple1;
        public readonly TemplateCollection LinksMethod1Multiple;
        public readonly TemplateCollection LinksMethodMultipleMultiple;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$LinksMethodNullFound*"] = LinksMethodNullFound.ToString();
            base["$LinksMethodFoundNull*"] = LinksMethodFoundNull.ToString();
            base["$LinksMethodMultiple1*"] = LinksMethodMultiple1.ToString();
            base["$LinksMethod1Multiple*"] = LinksMethod1Multiple.ToString();
            base["$LinksMethodMultipleMultiple*"] =
LinksMethodMultipleMultiple.ToString();
        }
    }
}
```

LinksMethods.tpl

```
private String Transform$Name!(String pstrPage)
{
        String strResultPage = pstrPage;
        $LinksMethodNullFound*
        $LinksMethodFoundNull*
        $LinksMethodMultiple1*
        $LinksMethod1Multiple*
        $LinksMethodMultipleMultiple*
        return strResultPage;
}
```

Platforms/ASPNET2_0/Templates/LinksPages.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class LinksPages : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public LinksPages()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\LinksPages.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

LinksPages.tpl

```
case "$Name!":
        strResultPage = Transform$Name!(pstrPage);
        break;
```

Platforms/ASPNET2_0/Templates/Methods.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class Methods : Template
    {
        private static string[] _phs
               = new string[] { "$WebMethod!", "$Visibility!", "$Abstract!",
               "$Return!", "$ReturnType!", "$Name!", "$Parameters*",
               "$ParametersComment!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public Methods()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\Methods.tpl"), _phs)
        {
        }

        public string WebMethod;
        public string Visibility;
        public string Abstract;
        public string ReturnType;
        public string Name;
        public readonly ObjectCollection Parameters = new ObjectCollection(", ");
        public string ParametersComment;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            if (WebMethod == "true") base["$WebMethod!"] = "[WebMethod] ";
            else base["$WebMethod!"] = null;
            base["$Visibility!"] = Visibility;
            if (Abstract == "true") base["$Abstract!"] = " abstract";
            base["$ReturnType!"] = " " + ReturnType;
            if (ReturnType != "void") base["$Return!"] = "return null;";
            base["$Name!"] = Name;
            base["$Parameters*"] = Parameters.ToString();
            base["$ParametersComment!"] = ParametersComment;
        }
    }
}
```

Methods.tpl

```
/// Method $Name!$ParametersComment!
$WebMethod!$Visibility!$Abstract!$ReturnType! $Name!($Parameters*)
{
       $Return!
}
```

Platforms/ASPNET2_0/Templates/Page.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class Page : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public Page()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\Page.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

Page.tpl

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="$Name!.aspx.cs"
Inherits="_$Name!" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Platforms/ASPNET2_0/Templates/PageCs.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class PageCs : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$PageCsDataAccess*", "$PageCsInfoNode*" };

        /// <summary>
        /// Constructor de la clase
```

```
        /// </summary>
        public PageCs()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\PageCs.tpl"), _phs)
        {
            PageCsDataAccess = new TemplateCollection("\n");
            PageCsDataAccess.FirstLineTabs = 0;
            PageCsDataAccess.Tabs = 0;
            PageCsInfoNode = new TemplateCollection("\n");
            PageCsInfoNode.FirstLineTabs = 5;
            PageCsInfoNode.Tabs = 5;
        }

        public string Name;
        public readonly TemplateCollection PageCsDataAccess;
        public readonly TemplateCollection PageCsInfoNode;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$PageCsDataAccess*"] = PageCsDataAccess.ToString();
            base["$PageCsInfoNode*"] = PageCsInfoNode.ToString();
        }
    }
}
```

PageCs.tpl

```
/// CSharpPageClass File generated by WebTALISMAN.

using System;
using System.Web.UI;
using System.Xml;
using System.Xml.Xsl;
using System.IO;
using System.Collections;
using log4net;
using log4net.Config;
using Solution.Business;
using Solution.BusinessFacade;

public partial class _$Name! : System.Web.UI.Page
{
    protected ILog log = LogManager.GetLogger(typeof(_$Name!));

    protected void Page_Load(object sender, EventArgs e)
    {
        XmlConfigurator.Configure(new
FileInfo(AppDomain.CurrentDomain.SetupInformation.ConfigurationFile));
        Hashtable objQuery = new Hashtable();
        System.Collections.Specialized.NameObjectCollectionBase.KeysCollection
objKeys =   Request.QueryString.Keys;
        foreach (String strKey in objKeys)
        {
            objQuery.Add(strKey, Request.QueryString.Get(strKey));
        }
        String strResult;
            try
            {
                //1: Load the date from the data base
                $PageCsDataAccess*

                //2: Write the XML document
                XmlDocument objXmlDocument = new XmlDocument();
                objXmlDocument.Load("Solution.WebUI/Hypertext/$Name!.xml");
                XmlNode objNodes;
                objNodes = objXmlDocument.DocumentElement;
```

```
                foreach (XmlNode objNode in objNodes.ChildNodes)
                {
                    switch (objNode.Name)
                    {
                         $PageCsInfoNode*
                    }
                }
                objXmlDocument.Save("Solution.WebUI/Hypertext/$Name!.xml");

                //3: Displays the page
                XmlDocument xdoc = new XmlDocument(); //XML Doc
                xdoc.Load("Solution.WebUI/Hypertext/$Name!.xml");
                XslCompiledTransform xsl = new XslCompiledTransform(); //XSLT Sheet
                xsl.Load("Solution.WebUI/Hypertext/XSLT/$Name!.xslt");
                StringWriter sw = new StringWriter();
                xsl.Transform(xdoc, null, sw);
                strResult = sw.ToString();

                //4: The menus
               Links objLinks = new Links();
                strResult = objLinks.Transform(strResult, "$Name!", objQuery);
            }
            catch (Exception ex) //There is no result
            {
                log.Error("ERROR: " + ex.Message);
                //Displays an error page
                XmlDocument xdoc = new XmlDocument(); //XML Doc
                xdoc.Load("Solution.WebUI/Hypertext/Error.xml");
                XslCompiledTransform xsl = new XslCompiledTransform(); //XSLT Sheet
                xsl.Load("Solution.WebUI/Hypertext/XSLT/Error.xslt");
                StringWriter sw = new StringWriter();
                XsltArgumentList xslarg = new XsltArgumentList();
                xslarg.AddParam("APage", "", "$Name!");
                xsl.Transform(xdoc, xslarg, sw);
                strResult = sw.ToString();
            }
            //Parse the controls and add it to the page
            Control ctrl = Page.ParseControl(strResult);
            Page.Controls.Add(ctrl);
        }
}
```

Platforms/ASPNET2_0/Templates/PageCsDataAccess.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class PageCsDataAccess : Template
    {
        private static string[] _phs
            = new string[] { "$Name!"};

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public PageCsDataAccess()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\PageCsDataAccess.tpl"), _phs)
        {
        }

        public string Name;
```

```
        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

PageCsDataAccess.tpl

```
                $Name!BF obj$Name!BF = new $Name!BF();
                 $Name! obj$Name! =
               obj$Name!BF.Get$Name!(Convert.ToInt32(objQuery["Id"]));
                log.Info("$Name! loaded");
```

Platforms/ASPNET2_0/Templates/PageCsInfoNode.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class PageCsInfoNode : Template
    {
        private static string[] _phs
            = new string[] { "$NodeName!", "$Page!", "$PropertyName!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public PageCsInfoNode()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\PageCsInfoNode.tpl"), _phs)
        {
        }

        public string NodeName;
        public string Page;
        public string PropertyName;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$NodeName!"] = NodeName;
            base["$Page!"] = Page;
            base["$PropertyName!"] = PropertyName;
        }
    }
}
```

PageCsInfoNode.tpl

```
case "$NodeName!":
objNode.InnerText = Convert.ToString(obj$Page!.$PropertyName!);
break;
```

Platforms/ASPNET2_0/Templates/Properties.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class Propertiess : Template
    {
        private static string[] _phs
               = new string[] { "$Visibility!", "$Changeability!", "$Static!",
               "$Type!", "$Name!", "$Initial!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public Propertiess()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\Properties.tpl"), _phs)
        {
        }

        public string Visibility;
        public string Changeability;
        public string Static;
        public string Type;
        public string Name;
        public string Initial;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Visibility!"] = Visibility;
            if (Changeability == "frozen") base["$Changeability!"] = " const";
            if (Static == "true") base["$Static!"] = " static";
            base["$Type!"] = " " + Type;
            base["$Name!"] = Name;
            if (Initial != "") base["$Initial!"] = " = " + Initial;
        }
    }
}
```

Properties.tpl

```
$Visibility!$Static!$Changeability!$Type! $Name!$Initial!;
```

Platforms/ASPNET2_0/Templates/SolutionBusinessCSPROJ.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SolutionBusinessCSPROJ : Template
    {
        private static string[] _phs
```

```
            = new string[] { "$EmbeddedResources*", "$Compiles*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SolutionBusinessCSPROJ()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SolutionBusinessCSPROJ.tpl"),
               _phs)
        {
            EmbeddedResources = new TemplateCollection("\n");
            EmbeddedResources.FirstLineTabs = 0;
            EmbeddedResources.Tabs = 2;
            Compiles = new TemplateCollection("\n");
            Compiles.FirstLineTabs = 0;
            Compiles.Tabs = 2;
        }

        public readonly TemplateCollection EmbeddedResources;
        public readonly TemplateCollection Compiles;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$EmbeddedResources*"] = EmbeddedResources.ToString();
            base["$Compiles*"] = Compiles.ToString();
        }
    }
}
```

SolutionBusinessCSPROJ.tpl

```
<Project DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>8.0.50727</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{6263EF00-6B01-46F8-A663-5CFFE22F5C9A}</ProjectGuid>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>Solution.Business</RootNamespace>
    <AssemblyName>Solution.Business</AssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Data" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
```

```
    <Compile Include="Properties\AssemblyInfo.cs" />
    $Compiles*
  </ItemGroup>
  <ItemGroup>
    $EmbeddedResources*
  </ItemGroup>
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the targets below
and uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Platforms/ASPNET2_0/Templates/SolutionBusinessFacadeCSPROJ.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SolutionBusinessFacadeCSPROJ : Template
    {
        private static string[] _phs
            = new string[] { "$Compiles*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SolutionBusinessFacadeCSPROJ()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SolutionBusinessFacadeCSPROJ.tpl"
               ), _phs)
        {
            Compiles = new TemplateCollection("\n");
            Compiles.FirstLineTabs = 0;
            Compiles.Tabs = 2;
        }

        public readonly TemplateCollection Compiles;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Compiles*"] = Compiles.ToString();
        }
    }
}
```

SolutionBusinessFacadeCSPROJ.tpl

```
<Project DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>8.0.50727</ProductVersion>
```

```
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{9DAC9EA1-1D84-4647-AE2A-2E1D5B73ECB9}</ProjectGuid>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>Solution.BusinessFacade</RootNamespace>
    <AssemblyName>Solution.BusinessFacade</AssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Data" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="BusinessFacadeBase.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
    $Compiles*
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\Solution.Business\Solution.Business.csproj">
      <Project>{6263EF00-6B01-46F8-A663-5CFFE22F5C9A}</Project>
      <Name>Solution.Business</Name>
    </ProjectReference>
    <ProjectReference Include="..\Solution.DataAccess\Solution.DataAccess.csproj">
      <Project>{A57FDC49-6404-47E8-AADD-DCDF967D6BAC}</Project>
      <Name>Solution.DataAccess</Name>
    </ProjectReference>
  </ItemGroup>
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the targets below
and uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Platforms/ASPNET2_0/Templates/SolutionDataAccessCSPROJ.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SolutionDataAccessCSPROJ : Template
    {
        private static string[] _phs
            = new string[] {  };
```

```
        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SolutionDataAccessCSPROJ()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SolutionDataAccessCSPROJ.tpl"),
               _phs)
        {
        }

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
        }
    }
}
```

SolutionDataAccessCSPROJ.tpl

```
<Project DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>8.0.50727</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{A57FDC49-6404-47E8-AADD-DCDF967D6BAC}</ProjectGuid>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>Solution.DataAccess</RootNamespace>
    <AssemblyName>Solution.DataAccess</AssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="HashCodeProvider, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=154fdcb44c4484fc">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\UTILS\Lib\HashCodeProvider.dll</HintPath>
    </Reference>
    <Reference Include="Iesi.Collections, Version=1.0.0.1, Culture=neutral,
PublicKeyToken=154fdcb44c4484fc">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\UTILS\Lib\Iesi.Collections.dll</HintPath>
    </Reference>
    <Reference Include="log4net, Version=1.2.9.0, Culture=neutral,
PublicKeyToken=b32731d11ce58905">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\UTILS\Lib\log4net.dll</HintPath>
    </Reference>
```

```
    <Reference Include="NHibernate, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=154fdcb44c4484fc">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\UTILS\Lib\NHibernate.dll</HintPath>
    </Reference>
    <Reference Include="System" />
    <Reference Include="System.Data" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="DataAccessBase.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\Solution.Business\Solution.Business.csproj">
      <Project>{6263EF00-6B01-46F8-A663-5CFFE22F5C9A}</Project>
      <Name>Solution.Business</Name>
    </ProjectReference>
  </ItemGroup>
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the targets below
and uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Platforms/ASPNET2_0/Templates/SolutionUnitTestCSPROJ.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SolutionUnitTestCSPROJ : Template
    {
        private static string[] _phs
            = new string[] { "$Compiles*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SolutionUnitTestCSPROJ()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SolutionUnitTestCSPROJ.tpl"),
               _phs)
        {
            Compiles = new TemplateCollection("\n");
            Compiles.FirstLineTabs = 0;
            Compiles.Tabs = 2;
        }

        public readonly TemplateCollection Compiles;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Compiles*"] = Compiles.ToString();
        }
    }
}
```

SolutionUnitTestCSPROJ.tpl

```
<Project DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>8.0.50727</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{E16AFE9D-34FB-4F01-9CDD-A625F5C66ACC}</ProjectGuid>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>Solution.UnitTest</RootNamespace>
    <AssemblyName>Solution.UnitTest</AssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="nunit.framework, Version=2.2.9.0, Culture=neutral,
PublicKeyToken=96d09a1eb7f44a77, processorArchitecture=MSIL">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\UTILS\Lib\nunit.framework.dll</HintPath>
    </Reference>
    <Reference Include="System" />
    <Reference Include="System.Data" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Properties\AssemblyInfo.cs" />
    $Compiles*
  </ItemGroup>
  <ItemGroup>
    <ProjectReference
Include="..\Solution.BusinessFacade\Solution.BusinessFacade.csproj">
      <Project>{9DAC9EA1-1D84-4647-AE2A-2E1D5B73ECB9}</Project>
      <Name>Solution.BusinessFacade</Name>
    </ProjectReference>
    <ProjectReference Include="..\Solution.Business\Solution.Business.csproj">
      <Project>{6263EF00-6B01-46F8-A663-5CFFE22F5C9A}</Project>
      <Name>Solution.Business</Name>
    </ProjectReference>
  </ItemGroup>
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the targets below
and uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Platforms/ASPNET2_0/Templates/SQLServer2005.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SQLServer2005 : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Tables*", "$TablesMultipleAssociation*",
"$Foreigns*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SQLServer2005()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SQLServer2005.tpl"), _phs)
        {
            Tables = new TemplateCollection("\n");
            Tables.FirstLineTabs = 0;
            Tables.Tabs = 0;
            TablesMultipleAssociation = new TemplateCollection("\n");
            TablesMultipleAssociation.FirstLineTabs = 0;
            TablesMultipleAssociation.Tabs = 0;
            Foreigns = new TemplateCollection("\n");
            Foreigns.FirstLineTabs = 0;
            Foreigns.Tabs = 0;
        }

        public string Name;
        public readonly TemplateCollection Tables;
        public readonly TemplateCollection TablesMultipleAssociation;
        public readonly TemplateCollection Foreigns;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$Tables*"] = Tables.ToString();
            base["$TablesMultipleAssociation*"] =
TablesMultipleAssociation.ToString();
            base["$Foreigns*"] = Foreigns.ToString();
        }
    }
}
```

SQLServer2005.tpl

```
use $Name!
go

$Tables*
$TablesMultipleAssociation*
/*Foreign Keys*/
$Foreigns*

go
```

Platforms/ASPNET2_0/Templates/SQLServer2005Foreign.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SQLServer2005Foreign : Template
    {
        private static string[] _phs
            = new string[] { "$Table!", "$Name!", "$Reference!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SQLServer2005Foreign()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SQLServer2005Foreign.tpl"), _phs)
        {
        }

        public string Table;
        public string Name;
        public string Reference;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Table!"] = Table;
            base["$Name!"] = Name;
            base["$Reference!"] = Reference;
        }
    }
}
```

SQLServer2005Foreign.tpl

```
ALTER TABLE $Table! ADD FOREIGN KEY ($Name!) REFERENCES $Reference!(Id) ON DELETE
CASCADE ON UPDATE CASCADE
```

Platforms/ASPNET2_0/Templates/SQLServer2005Property.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SQLServer2005Property : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Type!", "$Default!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SQLServer2005Property()
```

```
                : base(new System.IO.FileInfo(Application.StartupPath +
                "\\Platforms\\ASPNET2_0\\Templates\\SQLServer2005Property.tpl"),
                _phs)
        {
        }

        public string Name;
        public string Type;
        public string Default;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$Type!"] = Type;
            base["$Default!"] = Default;
        }
    }
}
```

SQLServer2005Property.tpl

```
$Name! $Type! default $Default!,
```

Platforms/ASPNET2_0/Templates/SQLServer2005Table.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SQLServer2005Table : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$TypeId!", "$Properties*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SQLServer2005Table()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SQLServer2005Table.tpl"), _phs)
        {
            Properties = new TemplateCollection("\n");
            Properties.FirstLineTabs = 0;
            Properties.Tabs = 1;
        }

        public string Name;
        public string TypeId;
        public readonly TemplateCollection Properties;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$TypeId!"] = TypeId;
            base["$Properties*"] = Properties.ToString();
        }
    }
```

```
}
```

SQLServer2005Table.tpl

```
CREATE TABLE $Name! (
        Id $TypeId! IDENTITY (1, 1) NOT NULL,
        $Properties*
        PRIMARY KEY (Id)
)
```

Platforms/ASPNET2_0/Templates/SQLServer2005TableMultipleAssociation.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class SQLServer2005TableMultipleAssociation : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$TypeId!", "$Name1!", "$Name2!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public SQLServer2005TableMultipleAssociation()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\SQLServer2005TableMultipleAssocia
               tion.tpl"), _phs)
        {
        }

        public string Name;
        public string TypeId;
        public string Name1;
        public string Name2;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$TypeId!"] = TypeId;
            base["$Name1!"] = Name1;
            base["$Name2!"] = Name2;
        }
    }
}
```

SQLServer2005TableMultipleAssociation.tpl

```
CREATE TABLE $Name! (
        Id $TypeId! IDENTITY (1, 1) NOT NULL,
        Id_$Name1! $TypeId! NOT NULL,
        Id_$Name2! $TypeId! NOT NULL,
        FOREIGN KEY (Id_$Name1!) REFERENCES $Name1!(Id) ON DELETE CASCADE ON UPDATE
CASCADE,
```

```
        FOREIGN KEY (Id_$Name2!) REFERENCES $Name2!(Id) ON DELETE CASCADE ON UPDATE
CASCADE,
        PRIMARY KEY (Id_$Name1!, Id_$Name2!)
)
```

Platforms/ASPNET2_0/Templates/StringSlices.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class StringSlices : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public StringSlices()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\StringSlices.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            string uName, first;
            uName = "";
            first = Name[0].ToString();
            first = first.ToUpper(); //Covierto a mayúscula

            for (int i = 1; i < Name.Length; i++) //en este ciclo se coge la palabra
            menos la 1ra letra
            {
                uName = uName + Name[i];
            }
               uName = first + uName; //Se forma la palabra que es igual a la
              original sólo que con la primera letra en mayúsculas
            base["$Name!"] = uName;
        }
    }
}
```

StringSlices.tpl

```
strOutPut += "$Name!: " + Convert.ToString(p$Name!) + "\r\n";
```

Platforms/ASPNET2_0/Templates/WebConfig.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
```

```
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class WebConfig : Template
    {
        private static string[] _phs
               = new string[] { "$hibernate.connection.provider!",
               "$hibernate.dialect!", "$hibernate.connection.driver_class!",
               "$hibernate.connection.connection_string!", "$Authorization*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public WebConfig()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\WebConfig.tpl"), _phs)
        {
            Authorization = new TemplateCollection("\n");
            Authorization.FirstLineTabs = 0;
            Authorization.Tabs = 1;
        }

        public string Provider;
        public string Dialect;
        public string Driver;
        public string ConnectionString;
        public readonly TemplateCollection Authorization;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$hibernate.connection.provider!"] = Provider;
            base["$hibernate.dialect!"] = Dialect;
            base["$hibernate.connection.driver_class!"] = Driver;
            base["$hibernate.connection.connection_string!"] = ConnectionString;
            base["$Authorization*"] = Authorization.ToString();
        }
    }
}
```

WebConfig.tpl

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">

  <configSections>
    <section name="nhibernate" type="System.Configuration.NameValueSectionHandler,
System, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,
log4net" />
  </configSections>

  <system.web>
    <roleManager enabled="true" />
    <authentication mode="Forms">
      <forms name="Login" loginUrl="Login.aspx" protection="All" timeout="45" >
      </forms>
    </authentication>
    <compilation debug="true"/>
  </system.web>

  $Authorization*

  <nhibernate>
    <add key="hibernate.connection.provider" value="$hibernate.connection.provider!"
/>
```

```
    <add key="hibernate.dialect" value="$hibernate.dialect!" />
    <add key="hibernate.connection.driver_class"
value="$hibernate.connection.driver_class!" />
    <add key="hibernate.connection.connection_string"
value="$hibernate.connection.connection_string!" />
  </nhibernate>

  <log4net>
    <appender name="GeneralLog" type="log4net.Appender.RollingFileAppender">
      <file value="Logs/general.log" />
      <appendToFile value="true" />
      <rollingStyle value="Size" />
      <maxSizeRollBackups value="10" />
      <maximumFileSize value="100KB" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%d{ddMMMyyyy-HH:mm:ss.fff} %-5p %c - %m%n" />
      </layout>
    </appender>
    <appender name="NHibernateFileLog" type="log4net.Appender.RollingFileAppender">
      <file value="Logs/nhibernate.log" />
      <appendToFile value="true" />
      <rollingStyle value="Size" />
      <maxSizeRollBackups value="10" />
      <maximumFileSize value="100KB" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%d{dd MMM yyyy HH:mm:ss.fff} %-5p %c - %m%n"  />
      </layout>
    </appender>
    <!-- levels: DEBUG, INFO, WARN, ERROR, FATAL -->
    <root>
      <level value="DEBUG" />
      <appender-ref ref="GeneralLog" />
    </root>
    <logger name="NHibernate" additivity="false">
      <level value="DEBUG"/>
      <appender-ref ref="NHibernateFileLog"/>
    </logger>
  </log4net>
</configuration>
```

Platforms/ASPNET2_0/Templates/WebConfigAuthorization.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class WebConfigAuthorization : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Value!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public WebConfigAuthorization()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\WebConfigAuthorization.tpl"),
               _phs)
        {
        }

        public string Name;
        public string Value;

        /// <summary>
```

```
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$Value!"] = Value;
        }
    }
}
```

WebConfigAuthorization.tpl

```
<location path="$Name!.aspx">
  <system.web>
    <authorization>
      $Value!
    </authorization>
  </system.web>
</location>
```

Platforms/ASPNET2_0/Templates/WebService.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class WebService : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public WebService()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\WebService.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name + "WS";
        }
    }
}
```

WebService.tpl

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/$Name!.cs" Class="$Name!" %>
```

Platforms/ASPNET2_0/Templates/WebServiceImpl.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class WebServiceImpl : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Properties*", "$GetSetProperties*",
"$Methods*"};

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public WebServiceImpl()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\WebServiceImpl.tpl"), _phs)
        {
            Properties = new TemplateCollection("\n");
            Properties.FirstLineTabs = 0;
            Properties.Tabs = 1;
            GetSetProperties = new TemplateCollection("\n");
            GetSetProperties.FirstLineTabs = 0;
            GetSetProperties.Tabs = 1;
            Methods = new TemplateCollection("\n");
            Methods.FirstLineTabs = 1;
            Methods.Tabs = 2;
        }

        public string Name;
        public readonly TemplateCollection Properties;
        public readonly TemplateCollection GetSetProperties;
        public readonly TemplateCollection Methods;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name + "WS";
            base["$Properties*"] = Properties.ToString();
            base["$GetSetProperties*"] = GetSetProperties.ToString();
            base["$Methods*"] = Methods.ToString();
        }
    }
}
```

WebServiceImpl.tpl

```
/// CSharpClass File generated by WebTALISMAN.
using System;
using System.Collections;
using System.Web.Services;
using Solution.BusinessFacade;
using Solution.Business;

/// <summary>
/// Summary description for $Name!
/// </summary>
[WebService(Namespace = "http://tempuri.org/$Name!")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class $Name! : System.Web.Services.WebService
{
```

```
        public $Name!()
        {
                //Uncomment the following line if using designed components
                //InitializeComponent();
        }

        //Properties
        $Properties*

        $GetSetProperties*
        $Methods*
}
```

Platforms/ASPNET2_0/Templates/XMLSchema.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class XMLSchema : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Elements*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public XMLSchema()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\XMLSchema.tpl"), _phs)
        {
            Elements = new TemplateCollection("\n");
            Elements.FirstLineTabs = 4;
            Elements.Tabs = 4;
        }

        public string Name;
        public readonly TemplateCollection Elements;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$Elements*"] = Elements.ToString();
        }
    }
}
```

XMLSchema.tpl

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="$Name!">
    <xsd:complexType>
      <xsd:sequence>
        $Elements*
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Platforms/ASPNET2_0/Templates/XMLSchemaElement.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class XMLSchemaElement : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Type!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public XMLSchemaElement()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\XMLSchemaElement.tpl"), _phs)
        {
        }

        public string Name;
        public string Type;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$Type!"] = Type;
        }
    }
}
```

XMLSchemaElement.tpl

```
<xsd:element name="$Name!" type="xsd:$Type!" />
```

Platforms/ASPNET2_0/Templates/XMLSchemaEmpty.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class XMLSchemaEmpty : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public XMLSchemaEmpty()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\XMLSchemaEmpty.tpl"), _phs)
        {
        }
```

```
        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

XMLSchemaEmpty.tpl

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="$Name!">
  </xsd:element>
</xsd:schema>
```

Platforms/ASPNET2_0/Templates/XMLSlice.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class XMLSlice : Template
    {
        private static string[] _phs
            = new string[] { "$Name!", "$Elements*" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public XMLSlice()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\XMLSlice.tpl"), _phs)
        {
            Elements = new TemplateCollection("\n");
            Elements.FirstLineTabs = 1;
            Elements.Tabs = 1;
        }

        public string Name;
        public readonly TemplateCollection Elements;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
            base["$Elements*"] = Elements.ToString();
        }
    }
}
```

XMLSlice.tpl

```
<?xml version="1.0" encoding="utf-8"?>
<$Name! xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="$Name!.xsd">
$Elements*
</$Name!>
```

Platforms/ASPNET2_0/Templates/XMLSliceElement.cs

```
using System;
using System.Windows.Forms;
using ExpertCoder.Templates;

namespace WebTALISMAN.Platforms.ASPNET2_0.Templates
{
    /// <summary>
    /// Para trabajar con la plantilla correspondiente a la clase
    /// </summary>
    public class XMLSliceElement : Template
    {
        private static string[] _phs
            = new string[] { "$Name!" };

        /// <summary>
        /// Constructor de la clase
        /// </summary>
        public XMLSliceElement()
               : base(new System.IO.FileInfo(Application.StartupPath +
               "\\Platforms\\ASPNET2_0\\Templates\\XMLSliceElement.tpl"), _phs)
        {
        }

        public string Name;

        /// <summary>
        /// Introducción de la información en la plantilla de destino
        /// </summary>
        protected override void SetPlaceHoldersValues()
        {
            base["$Name!"] = Name;
        }
    }
}
```

Anexo B. APLICACIÓN WEB "*CASAS RURALES*"

Archivo "pim.xml"

```
<?xml version="1.0" encoding="utf-8"?>
<PIM>
  <classes>
    <class name="Casa" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="nombreCasa" visibility="public" type="string"
initial="&quot;NOMBRE_DE_LA_CASA&quot;" changeability="changeable" isStatic="true"
/>
        <property name="direccion" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="telefono" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="precioDiaFest" visibility="public" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="precioDiaLab" visibility="public" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="comoLlegar" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="estado" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="tipoCasa" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="capacidad" visibility="protected" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="foto" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="logo" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="personaC" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="cuentaPersonaC" visibility="protected" type="string"
initial="" changeability="changeable" isStatic="false" />
        <property name="tlfPersonaC" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="ActPropia" multiplicity="multiple" />
        <association target="Zona" multiplicity="1" dependent="false" />
        <association target="Reserva" multiplicity="multiple" />
        <association target="Habitacion" multiplicity="multiple" />
      </associations>
      <methods>
```

```
        <method name="UnMetodo" type="normal" visibility="public"
returnType="string" isAbstract="false">
          <parameters>
            <param name="param1" type="string" />
            <param name="param2" type="int" />
          </parameters>
        </method>
      </methods>
    </class>
    <class name="Habitacion" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="nHabitacion" visibility="protected" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="capacidad" visibility="protected" type="int" initial=""
changeability="changeable" isStatic="false" />
        <property name="estado" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="Casa" multiplicity="1" dependent="false" />
      </associations>
      <methods>
      </methods>
    </class>
    <class name="Reserva" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="fechaInicio" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="fechaFinal" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="Casa" multiplicity="1" dependent="false" />
        <association target="Cliente" multiplicity="1" dependent="false" />
      </associations>
      <methods>
      </methods>
    </class>
    <class name="Cliente" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="DNIC" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="nombreC" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="apellidosC" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="direccionC" visibility="protected" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="telefonoC" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="Reserva" multiplicity="multiple" />
      </associations>
      <methods>
      </methods>
    </class>
    <class name="Zona" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="nombreZ" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="descripcion" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="foto" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="Casa" multiplicity="multiple" />
```

```
        <association target="Actividad" multiplicity="multiple" />
      </associations>
      <methods>
      </methods>
    </class>
    <class name="Actividad" visibility="public" inherit="" isFinal="false"
isAbstract="false">
      <properties>
        <property name="nombreA" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="descripcion" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="Zona" multiplicity="multiple" />
      </associations>
      <methods>
      </methods>
    </class>
    <class name="ActPropia" visibility="public" inherit="Actividad" isFinal="false"
isAbstract="false">
      <properties>
        <property name="nombreA" visibility="inherit" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="descripcion" visibility="inherit" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="horario" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
        <property name="precioAct" visibility="public" type="string" initial=""
changeability="changeable" isStatic="false" />
      </properties>
      <associations>
        <association target="Casa" multiplicity="1" dependent="false" />
      </associations>
      <methods>
      </methods>
    </class>
  </classes>
  <fragments>
    <fragment name="Default">
      <properties>
      </properties>
      <associations>
        <association target="Zona" multiplicity="multiple" />
      </associations>
    </fragment>
    <fragment name="Zona">
      <properties>
        <property name="nombreZ" />
        <property name="descripcion" />
        <property name="foto" />
      </properties>
      <associations>
        <association target="Default" multiplicity="1" />
        <association target="Casa" multiplicity="multiple" />
        <association target="Actividad" multiplicity="multipleZero" />
      </associations>
    </fragment>
    <fragment name="Casa">
      <properties>
        <property name="nombreCasa" />
        <property name="direccion" />
        <property name="telefono" />
        <property name="comoLlegar" />
        <property name="precioDiaFest" />
        <property name="precioDiaLab" />
        <property name="foto" />
        <property name="logo" />
      </properties>
      <associations>
        <association target="Zona" multiplicity="1" />
        <association target="Reserva" multiplicity="multipleZero" />
        <association target="ActPropia" multiplicity="multiple" />
```

```
      </associations>
    </fragment>
    <fragment name="Reserva">
      <properties>
        <property name="fechaInicio" />
        <property name="fechaFinal" />
        <property name="Cliente.DNIC" />
        <property name="Cliente.nombreC" />
        <property name="Cliente.apellidosC" />
        <property name="Cliente.telefonoC" />
      </properties>
      <associations>
        <association target="Casa" multiplicity="1" />
      </associations>
    </fragment>
    <fragment name="Actividad">
      <properties>
        <property name="nombreA" />
        <property name="descripcion" />
      </properties>
      <associations>
        <association target="Zona" multiplicity="multiple" />
      </associations>
    </fragment>
    <fragment name="ActPropia">
      <properties>
        <property name="nombreA" />
        <property name="descripcion" />
        <property name="horario" />
        <property name="precioAct" />
      </properties>
      <associations>
        <association target="Casa" multiplicity="1" />
      </associations>
    </fragment>
  </fragments>
  <navigation>
    <Default type="resource" label="sliceDefault"></Default>
    <Zona type="resource" label="sliceZona"></Zona>
    <Casa type="resource" label="sliceCasa"></Casa>
    <Actividad type="resource" label="sliceActividad"></Actividad>
    <ActPropia type="resource" label="sliceActPropia"></ActPropia>
    <Reserva type="resource" label="sliceReserva"></Reserva>
    <Index1 type="resource" label="sliceIndex1"></Index1>
    <Index2 type="resource" label="sliceIndex2"></Index2>
    <Index3 type="resource" label="sliceIndex3"></Index3>
    <Index4 type="resource" label="sliceIndex4"></Index4>
    <Index5 type="resource" label="sliceIndex5"></Index5>
    <Index6 type="resource" label="sliceIndex6"></Index6>
    <link1 type="arc" from="sliceDefault" to="sliceIndex1" show="replace"
actuate="onRequest" title="" index=""></link1>
    <link2 type="arc" from="sliceIndex1" to="sliceZona" show="replace"
actuate="onRequest" title="Zonas" index="NombreZ"></link2>
    <link3 type="arc" from="sliceZona" to="sliceDefault" show="replace"
actuate="onRequest" title="Inicio" index=""></link3>
    <link4 type="arc" from="sliceZona" to="sliceIndex2" show="replace"
actuate="onRequest" title="" index=""></link4>
    <link14 type="arc" from="sliceZona" to="sliceIndex5" show="replace"
actuate="onRequest" title="" index=""></link14>
    <link5 type="arc" from="sliceIndex2" to="sliceCasa" show="replace"
actuate="onRequest" title="Casas" index="NombreCasa"></link5>
    <link6 type="arc" from="sliceCasa" to="sliceZona" show="replace"
actuate="onRequest" title="Volver a zona" index=""></link6>
    <link9 type="arc" from="sliceCasa" to="sliceIndex3" show="replace"
actuate="onRequest" title="" index=""></link9>
    <link11 type="arc" from="sliceCasa" to="sliceIndex4" show="replace"
actuate="onRequest" title="" index=""></link11>
    <link7 type="arc" from="sliceActividad" to="sliceIndex6" show="replace"
actuate="onRequest" title="" index=""></link7>
    <link16 type="arc" from="sliceIndex6" to="sliceZona" show="replace"
actuate="onRequest" title="Zonas" index="NombreZ"></link16>
```

```
    <link8 type="arc" from="sliceActPropia" to="sliceCasa" show="replace"
actuate="onRequest" title="Volver a casa" index=""></link8>
    <link10 type="arc" from="sliceIndex3" to="sliceReserva" show="replace"
actuate="onRequest" title="Reservas" index="FechaInicio"></link10>
    <link12 type="arc" from="sliceIndex4" to="sliceActPropia" show="replace"
actuate="onRequest" title="Actividades" index="NombreA"></link12>
    <link13 type="arc" from="sliceReserva" to="sliceCasa" show="replace"
actuate="onRequest" title="Volver a casa" index=""></link13>
    <link15 type="arc" from="sliceIndex5" to="sliceActividad" show="replace"
actuate="onRequest" title="Actividades" index="NombreA"></link15>
  </navigation>
  <users>
    <page name="Reserva">
      <allow roles='Admin'/>
      <deny users='?'/>
    </page>
  </users>
  <webServices>
    <webService name="ObtenerInfoCasasRurales">
      <properties>
        <property name="strNombre" visibility="public" type="string"
initial="&amp;quot;CASAS_RURALES&amp;quot;" changeability="changeable" isStatic="true" />
      </properties>
      <methods>
        <method name="GetZonas" type="webService" visibility="public"
returnType="Zona[]" >
          <parameters>
          </parameters>
        </method>
        <method name="GetCasas" type="webService" visibility="public"
returnType="Casa[]" >
          <parameters>
            <param name="idZona" type="int" />
          </parameters>
        </method>
        <method name="GetActPropias" type="webService" visibility="public"
returnType="ActPropia[]" >
          <parameters>
            <param name="idCasa" type="int" />
          </parameters>
        </method>
      </methods>
    </webService>
    <webService name="ObtenerInfoActividades">
      <properties>
      </properties>
      <methods>
        <method name="GetActividades" type="webService" visibility="public"
returnType="Actividad[]" >
          <parameters>
          </parameters>
        </method>
      </methods>
    </webService>
  </webServices>
  <webServicesClients>
    <webServicesClient name="ContinentalEstadoWSC"
url="http://webservices.continental.com/flightstatus/flightstatus.asmx?WSDL" />
    <webServicesClient name="ContinentalCalendarioWSC"
url="http://webservices.continental.com/schedule/schedule.asmx?wsdl" />
  </webServicesClients>
</PIM>
```

Actividad.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" >
      <head runat="server">
        <title>
          Actividad - <xsl:value-of select="Actividad/nombreA" />
        </title>
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <menus />
      </head>
      <body>
        <form id="form1" runat="server">
          <div>
              <div class='info'>
              <h1>Descripción de la actividad</h1>
                <table summary="Actividad">
                  <tr>
                    <td>
                      <span class='title'>Nombre: </span><span
class='titleInfo'><xsl:value-of select="Actividad/nombreA" /></span><br/><br/>
                      <span class='titleInfo'><xsl:value-of
select="Actividad/descripcion" /></span><br/><br/>
                    </td>
                    <td class="menuTitle" valign="top"><Actividad.Zona /></td>
                  </tr>
                </table>
            </div>
            <p>
              <a href="http://validator.w3.org/check?uri=referer"><img
src="Hypertext/Media/valid-xhtml10.png" alt="Valid XHTML 1.0 Transitional"
height="31" width="88" /></a>
              <a href="http://jigsaw.w3.org/css-validator/#validate-by-input"><img
src="Hypertext/Media/vcss.gif" alt="Valid CSS 2.1" height="31" width="88" /></a>
              <a href="http://www.w3.org/WAI/WCAG1-Conformance"><img
src="Hypertext/Media/vaa.gif" alt="Valid WAI-AA" height="31" width="88" /></a>
            </p>
          </div>
        </form>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

ActPropia.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" >
      <head runat="server">
        <title>
          Actividad Propia - <xsl:value-of select="ActPropia/nombreA" />
        </title>
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <menus />
```

```
      </head>
      <body>
        <form id="form1" runat="server">
          <div>
              <div class='info'>
              <h1>Descripción de la actividad de la casa</h1>
                <table summary="Actividad">
                  <tr>
                    <td>
                      <span class='title'>Nombre: </span><span
class='titleInfo'><xsl:value-of select="ActPropia/nombreA" /></span><br/><br/>
                      <span class='title'>Horario: </span><span
class='titleInfo'><xsl:value-of select="ActPropia/horario" /></span><br/><br/>
                      <span class='title'>Precio: </span><span
class='titleInfo'><xsl:value-of select="ActPropia/precioAct" />€</span><br/><br/>
                      <span class='titleInfo'><xsl:value-of
select="ActPropia/descripcion" /></span><br/><br/>
                    </td>
                    <td class="menuTitle" valign="top"><ActPropia.Casa /></td>
                  </tr>
                </table>
            </div>
            <p>
              <a href="http://validator.w3.org/check?uri=referer"><img
src="Hypertext/Media/valid-xhtml10.png" alt="Valid XHTML 1.0 Transitional"
height="31" width="88" /></a>
              <a href="http://jigsaw.w3.org/css-validator/#validate-by-input"><img
src="Hypertext/Media/vcss.gif" alt="Valid CSS 2.1" height="31" width="88" /></a>
              <a href="http://www.w3.org/WAI/WCAG1-Conformance"><img
src="Hypertext/Media/vaa.gif" alt="Valid WAI-AA" height="31" width="88" /></a>
            </p>
          </div>
        </form>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Casa.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" >
      <head runat="server">
        <title>
          Casa - <xsl:value-of select="Casa/nombreCasa" />
        </title>
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <menus />
      </head>
      <body>
        <form id="form1" runat="server">
          <div>
              <div class='info'>
              <h1>Casa</h1>
                <table summary="Casa">
                  <tr>
                    <td>
                      <span class='title'>Nombre: </span><span
class='titleInfo'><xsl:value-of select="Casa/nombreCasa" /></span><br/><br/>
                      <span class='title'>Dirección: </span><span
class='titleInfo'><xsl:value-of select="Casa/direccion" /></span><br/><br/>
                      <span class='title'>Teléfono: </span><span
class='titleInfo'><xsl:value-of select="Casa/telefono" /></span><br/><br/>
```

```
                    <span class='title'>Como llegar: </span><span
class='titleInfo'><xsl:value-of select="Casa/comoLlegar" /></span><br/><br/>
                    <span class='title'>Precio día festivo: </span><span
class='titleInfo'><xsl:value-of select="Casa/precioDiaFest" />€</span><br/><br/>
                    <span class='title'>Precio día laborable: </span><span
class='titleInfo'><xsl:value-of select="Casa/precioDiaLab" />€</span><br/><br/>
                  </td>
                  <td class="menuTitle" valign="top"><Casa.Zona /></td>
                  <td class="menuTitle" valign="top"><Casa.ActPropia /></td>
                  <td class="menuTitle" valign="top"><Casa.Reserva /></td>
                </tr>
              </table>
              <p style="text-position:center">
                <img style="border:none;text-align:center" alt="Foto de la
casa"><xsl:attribute name="src">Hypertext/Media<xsl:value-of select="Casa/foto"
/></xsl:attribute></img>
              </p>
            </div>
            <p>
              <a href="http://validator.w3.org/check?uri=referer"><img
src="Hypertext/Media/valid-xhtml10.png" alt="Valid XHTML 1.0 Transitional"
height="31" width="88" /></a>
              <a href="http://jigsaw.w3.org/css-validator/#validate-by-input"><img
src="Hypertext/Media/vcss.gif" alt="Valid CSS 2.1" height="31" width="88" /></a>
              <a href="http://www.w3.org/WAI/WCAG1-Conformance"><img
src="Hypertext/Media/vaa.gif" alt="Valid WAI-AA" height="31" width="88" /></a>
            </p>
          </div>
        </form>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Default.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" >
      <head runat="server">
        <title>
          Casas Rurales en Asturias
        </title>
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <menus />
      </head>
      <body>
        <form id="form1" runat="server">
          <div>
            <h1 class="title">Casas Rurales en Asturias</h1>
            <table summary="Portada principal">
              <tr valign="middle">
                <td>
                  <img src="Hypertext/Media/Default/imagen1.jpg" alt="Logo Casas
Rurales de Asturias" />
                </td>
                <td class="menuTitle">
                  <img src="Hypertext/Media/Default/ecoturismo.jpg" alt="Logo
ecoturismo" />
                  <br/>
                  <p>Seleccione una Zona:</p>
                  <Default.Zona />
                </td>
              </tr>
```

```
            </table>
            <p>
              <a href="http://validator.w3.org/check?uri=referer"><img
src="Hypertext/Media/valid-xhtml10.png" style="border:none" alt="Valid XHTML 1.0
Transitional" height="31" width="88" /></a>
              <a href="http://jigsaw.w3.org/css-validator/#validate-by-input"><img
src="Hypertext/Media/vcss.gif" style="border:none" alt="Valid CSS 2.1" height="31"
width="88" /></a>
              <a href="http://www.w3.org/WAI/WCAG1-Conformance"><img
src="Hypertext/Media/vaa.gif" style="border:none" alt="Valid WAI-AA" height="31"
width="88" /></a>
            </p>
          </div>
        </form>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Error.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:param name="APage"/>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" >
      <head runat="server">
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <title>
          <xsl:value-of select="$APage" />
        </title>
      </head>
      <body>
        <form id="form1" runat="server">
          <div class="error">
            Se ha producido un error en <xsl:value-of select="$APage" /> debido a
que no se ha podido obtener la información necesaria para generar la página. Por
favor, revise que ha cargado la página correctamente.
          </div>
        </form>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Reserva.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" >
      <head runat="server">
        <title>
          Reserva - <xsl:value-of select="Reserva/nombreC" />
        </title>
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <menus />
      </head>
      <body>
```

```
        <form id="form1" runat="server">
          <div>
              <div class='info'>
              <h1>Descripción de la reserva</h1>
                <table summary="Reserva">
                  <tr>
                    <td>
                      <span class='titleInfo'>Nombre: <xsl:value-of
select="Reserva/Cliente.nombreC" /></span><br/><br/>
                      <span class='titleInfo'>Apellidos: <xsl:value-of
select="Reserva/Cliente.apellidosC" /></span><br/><br/>
                      <span class='titleInfo'>DNI: <xsl:value-of
select="Reserva/Cliente.DNIC" /></span><br/><br/>
                      <span class='titleInfo'>Teléfono: <xsl:value-of
select="Reserva/Cliente.telefonoC" /></span><br/><br/>
                      <span class='titleInfo'>Fecha inicial: <xsl:value-of
select="Reserva/fechaInicio" /></span><br/><br/>
                      <span class='titleInfo'>Fecha final: <xsl:value-of
select="Reserva/fechaFinal" /></span><br/><br/>
                    </td>
                  </tr>
                  <tr><td><Reserva.Casa /></td></tr>
                </table>
            </div>
            <p>
              <a href="http://validator.w3.org/check?uri=referer"><img
src="Hypertext/Media/valid-xhtml10.png" alt="Valid XHTML 1.0 Transitional"
height="31" width="88" /></a>
              <a href="http://jigsaw.w3.org/css-validator/#validate-by-input"><img
src="Hypertext/Media/vcss.gif" alt="Valid CSS 2.1" height="31" width="88" /></a>
              <a href="http://www.w3.org/WAI/WCAG1-Conformance"><img
src="Hypertext/Media/vaa.gif" alt="Valid WAI-AA" height="31" width="88" /></a>
            </p>
          </div>
        </form>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

StyleSheet.css

```
body
{
        background-color:#CCFFCC;
}

h1
{
        font-family:"Comic Sans MS";
        font-style:italic;
        font-variant:small-caps;
        text-align:center;
        color:#663300;
}

img
{
        border-bottom-style:none;
        border-top-style:none;
        border-right-style:none;
        border-left-style:none;
}

.info
{
        background-color:#CCFFCC;
        border-top-color:black;
        border-top-style:solid;
```

```
        border-top-width:thin;
        border-bottom-color:black;
        border-bottom-style:solid;
        border-bottom-width:thin;
        border-left-color:black;
        border-left-style:solid;
        border-left-width:thin;
        border-right-color:black;
        border-right-style:solid;
        border-right-width:thin;
}

.title
{
        padding-left:30px;
        font-style:italic;
        font-variant:small-caps;
        text-align:center;
        font-family:"Comic Sans MS";
        color:blue;
}

.menuTitle
{
        padding-left:25px;
        font-family:Verdana;
        color:blue;
        font-variant:small-caps;
}

.titleInfo
{
        font-family:Verdana;
}


/*OPTIONAL: To error pages*/
.error
{
        font-family:Verdana;
        font-size:larger;
        color:red;
        background-color:white;
        text-align:center;
        margin-top:100px;
}

/*OPTIONAL: To login page*/
.login
{
}
```

Zona.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="yes">
  </xsl:output>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" >
      <head runat="server">
        <title>
          Zona - <xsl:value-of select="Zona/nombreZ" />
        </title>
        <link href="Hypertext/XSLT/StyleSheet.css" rel="stylesheet" type="text/css"
/>
        <menus />
      </head>
      <body>
```

```
        <form id="form1" runat="server">
          <div>
              <div class='info'>
              <h1>Descripción de la zona</h1>
                <table summary="Zona">
                  <tr>
                    <td>
                      <span class='title'>Nombre: </span><span
class='titleInfo'><xsl:value-of select="Zona/nombreZ" /></span><br/><br/>
                      <span class='titleInfo'><xsl:value-of
select="Zona/descripcion" /></span><br/><br/>
                      <img alt="Foto de la zona"><xsl:attribute
name="src">Hypertext/Media<xsl:value-of select="Zona/foto" /></xsl:attribute></img>
                    </td>
                    <td class="menuTitle" valign="top"><Zona.Casa /></td>
                    <td class="menuTitle" valign="top"><Zona.Actividad /></td>
                    <td class="menuTitle" valign="top"><Zona.Default /></td>
                  </tr>
                </table>
            </div>
            <p>
              <a href="http://validator.w3.org/check?uri=referer"><img
src="Hypertext/Media/valid-xhtml10.png" alt="Valid XHTML 1.0 Transitional"
height="31" width="88" /></a>
              <a href="http://jigsaw.w3.org/css-validator/#validate-by-input"><img
src="Hypertext/Media/vcss.gif" alt="Valid CSS 2.1" height="31" width="88" /></a>
              <a href="http://www.w3.org/WAI/WCAG1-Conformance"><img
src="Hypertext/Media/vaa.gif" alt="Valid WAI-AA" height="31" width="88" /></a>
            </p>
          </div>
        </form>
      </body>
    </html>
  </xsl:template>

      </xsl:stylesheet>
```

Listado de Archivos generados

Solution.sln
Solution.WebUI\Web.config
Solution.WebUI\Actividad.aspx
Solution.WebUI\Actividad.aspx.cs
Solution.WebUI\ActPropia.aspx
Solution.WebUI\ActPropia.aspx.cs
Solution.WebUI\Casa.aspx
Solution.WebUI\Casa.aspx.cs
Solution.WebUI\Default.aspx
Solution.WebUI\Default.aspx.cs
Solution.WebUI\Login.aspx
Solution.WebUI\Login.aspx.cs
Solution.WebUI\Reserva.aspx
Solution.WebUI\Reserva.aspx.cs
Solution.WebUI\Zona.aspx
Solution.WebUI\Zona.aspx.cs
Solution.WebUI\App_Code\ContinentalCalendarioWSC.cs
Solution.WebUI\App_Code\ContinentalEstadoWSC.cs
Solution.WebUI\App_Code\Links.cs
Solution.WebUI\App_Code\ObtenerInfoActividadesWS.cs

Solution.WebUI\App_Code\ObtenerInfoCasasRurarlesWS.cs
Solution.WebUI\WebServices\ObtenerInfoActividades.asmx
Solution.WebUI\WebServices\ObtenerInfoCasasRurarles.asmx
Solution.Business\Solution.Business.csproj
Solution.Business\Actividad.cs
Solution.Business\Actividad.hbm.xml
Solution.Business\ActPropia.cs
Solution.Business\ActPropia.hbm.xml
Solution.Business\Casa.cs
Solution.Business\Casa.hbm.xml
Solution.Business\Cliente.cs
Solution.Business\Cliente.hbm.xml
Solution.Business\Habitacion.cs
Solution.Business\Habitacion.hbm.xml
Solution.Business\Reserva.cs
Solution.Business\Reserva.hbm.xml
Solution.Business\Zona.cs
Solution.Business\Zona.hbm.xml
Solution.Business\Zona_Actividad.cs
Solution.Business\Zona_Actividad.hbm.xml
Solution.BusinessFacade\Solution.BusinessFacade.csproj
Solution.BusinessFacade\ActividadBF.cs
Solution.BusinessFacade\ActPropiaBF.cs
Solution.BusinessFacade\BusinessFacadeBase.cs
Solution.BusinessFacade\CasaBF.cs
Solution.BusinessFacade\ClienteBF.cs
Solution.BusinessFacade\HabitacionBF.cs
Solution.BusinessFacade\ReservaBF.cs
Solution.BusinessFacade\Zona_ActividadBF.cs
Solution.BusinessFacade\ZonaBF.cs
Solution.DataAccess\Solution.DataAccess.csproj
Solution.DataAccess\DataAccessBase.cs
Solution.UnitTest\Solution.UnitTest.csproj
Solution.UnitTest\ActividadTest.cs
Solution.UnitTest\ActPropiaTest.cs
Solution.UnitTest\CasaTest.cs
Solution.UnitTest\ClienteTest.cs
Solution.UnitTest\HabitacionTest.cs
Solution.UnitTest\ReservaTest.cs
Solution.UnitTest\ZonaTest.cs
UTILS\DataBase\DataBase.sql

LISTA DE ACRÓNIMOS

ANSI	American Nacional Standards Institute
ASL	Action Specification Language
ASM	Abstract State Machine
ASP	Active Server Pages
AST	Abstract Syntax Tree
ATC	Atomic Transformation Code
ATL	ATLAS Transformation Language
BPEL	Business Process Execution Language
CASE	Computer-Aided Software Engineering
CMOF	Complete MOF
CORBA	Common Object Request Broker Architecture
CRUD	Create, read, update and delete
CWM	Common Warehouse Metamodel
DAO	Data Access Object
DDL	Data Definition Language
DPC	Dynamic Proxy Classes.
DSL	Domain Specific Language
DTD	Document Type Definition

EBNF	Extended Backus Naur Form
EMF	Eclipse Modeling Framework
EMOF	Essential MOF
GMT	Generative Modeling Technologies
GMT	Generative Model Transformer
HUTN	Human-Usable Textual Notation
IBM	International Business Machines
INCITS	International Committee for Information Technology Standards
INRIA	Institut national de recherche en informatique et en automatique
ISO	International organization for standardization
Java EE	Java Platform, Enterprise Edition
JCP	Java Community Process
JDO	Java Data Objects Specification.
JET	Java Emitter Templates
JMI	Java Metadata Interface
JSF	Java Server Faces
JSP	Java Server Pages
MCC	Model Component Compiler
MDA	Model Driven Architecture
MDD	Model Driven Development
MDR	Metadata Repository
MOF	Meta Object Facility
MOMENT	MOdel manageMENT
MTOM	M2MModel-to-Model Transformation
MTOT	M2TModel-to-Text Transformation
OCL	Object Constraint Language

OMG	Object Management Group
OMT	Object Modeling Technique
OOPSLA	Object-Oriented Programming, Systems, Languages Applications
OOSE	Object-oriented Software Engineering
PIM	Platform Independent Model
PSI	Platform Specific Implementation
PSL	Process Specification Language
PSM	Platform Specific Model
QVT	Query/View/Transformation
RFP	Request for Proposal
RUP	Rational Unified Proccess
SDL	Specification and Design Language
SQL	Structured Query Language
TDD	Test-Driven Development
UML	Unified Modeling Language
UMT	Model Transformation Tool.
UP	Unified Proccess
VIATRA	VIsual Automated model TRAnsformations
VPM	Visual and Precise Metamodeling
VTCL	Viatra Textual Command Language
VTML	Viatra Textual (Meta)Modeling Language
VTTL	Viatra Textual Template Language
WfMC	Workflow Management Coalition
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XP	Extreme Progamming

XPDL XML Process Definition Language

XSD XML Schema Definition

XSLT Extensible Stylesheet Language Transformations

BIBLIOGRAFÍA Y REFERENCIAS WEB

[ACJ01] D. Alur, J. Crupi y D. Malks.
Core J2EE Patterns - Best Practices and Design Strategies.
Pearson Education, 2001.

[AJ02] S.W.Ambler y R. Jeffries
Agile Modeling: Effective Practices for Extreme Programming and the Unified Process.
Wiley, 1st edition. 2002

[Ale77] C. Alexander.
A Pattern Language: Towns, Buildings, Construction.
Oxford University Press, 1977.

[AM95] M. P. Atkinson y R. Morrison.
Orthogonally persistent object systems.
The VLDB Journal, 4(3):115–120, Julio 1995.

[Amb04] S.W. Ambler
Agile Model Driven Development with UML 2.
Cambridge University Press, 3rd edition, 2004.

[AMP86] M. P. Atkinson, R. Morrison, y G. D. Pratten.
Designing a Persistent Information Space Architecture.
Proceedings of 10th IFIP World Congress, p. 115–120. Dublin, 1986.

[And] AndroMDA.
Sitio Web oficial: http://www.andromda.org/.

[And06a] *Andromda cartridges documentation*.
http://galaxy.andromda.org/docs/andromda-cartridges/index.html.

[And06b] *Andromda metafacades.*
http://galaxy.andromda.org/docs/andromda-metafacades/index.html.

[And06c] *Andromda BPM for struts cartridge.*
http://galaxy.andromda.org/docs/andromda-bpm4struts-cartridge/index.html.

[ANS] ANSI. American nacional standards institute.
Sitio Web oficial: http://www.ansi.org/.

[Ant] *Antlr Parser Generator Home.*
http://www.antlr.org

[Aok91] P. M. Aoki.
Implementation of extended indexes in postgres.
SIGIR Forum, 25(1):2--9, 1991.

[Apa06] *Apache Velocity Home Page.*
http://jakarta.apache.org/velocity/.

[Arc] ArcStyler.
Sitio Web oficial: http://www.io-software.com/.

[ATL] ATL. Atlas Transformation Language.
Sitio Web oficial: http://www.sciences.univ-nantes.fr/lina/atl/.

[BBI+04] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh y B. Selic
An MDA Manifesto
The MDA Journal, páginas 133-144, Diciembre 2004

[BC89] K. Beck y W. Cunningham.
A Laboratory for Teaching Object-Oriented Thinking.
En OOPSLA, pages 1--6, 1989.

[BD06] M. Belaunde y G.Dupe
SmartQVT Home Page.
http://smartqvt.elibel.tm.fr/index.html

[Bec01] K. Beck.
Manifesto for agile software development.
Technical Report, 2001.
http://www.agilemanifesto.org/.

[Bec02] K. Beck.
Test Driven Development: By Example.
Addison- Wesley Professional, 1st edition. 2002

[Bec99a] K. Beck.
Extreme Programming Explained: Embrace Change.
Addison Wesley, 1999.

[Bec99b] K. Beck.
Embracing change with extreme programming.
IEEE Computer, 32(10):70--77, 1999.

[Beu06] C. Beust
Agile people still don't get it.
Otaku, Cedric's Weblog. 2006
Disponible en http://beust.com/weblog/archives/000392.html.

[BF01] K. Beck y M. Fowler.
Planning Extreme Programming.
Addison Wesley, 2001.

[BF95] E. Bertino y P. Foscoli.
Index organizations for object-oriented database systems.
IEEE Trans. Knowl. Data Eng., 7(2):193--209, 1995.

[BK04] C. Bauer y G. King.
Hibernate in action.
Manning publications, 2004.

[BK04] C. Bauer, y G. King
Hibernate in Action.
Manning Publications, 1st edition, 2004

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, P. Sommerlad, y M. Stal.
Pattern-Oriented Software Architecture: A System of Patterns.
John Wiley & Sons, Agosto 1996.

[BO99] E. Bertino y B.C. Ooi.
The indispensability of dispensable indexes.
IEEE Trans. Knowl. Data Eng., 11(1):17--27, 1999.

[Boc02] J. Bock.
CIL Programming: Under the Hood of .NET.
Apress, Junio 2002.

[Boe88] B. Boehm
A spiral model of software development and enhancement.
Computer, 21(5):61–72, 1988

[Boh02] M. Bohlen
Enterprise Java Beans Ge-Packt.
Mitp-Verlag,2002

[Boh06] M. Bohlen
The AndroMDA Architecture, 2006
http://galaxy.andromda.org/index.php?option=com_
content&task=blogsection&id=10&Itemid=78

[Boo93] G. Booch
Object-Oriented Analysis and Design with Applications.
Addison-Wesley Professional, 2nd edition. 1993

[Boo94] G. Booch.
Análisis y diseño orientado a objetos con aplicaciones.
Addison Wesley / Díaz de Santos, 1994.

[BS03] E. Börger y R. Stärk
Abstract state machines. A method for high-level system design and analysis.
Springer-Verlag, 2003

[BSM+03] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick y T.J. Grose
Eclipse Modeling Framework: A Developer's Guide.
Addison Wesley, 2003

[BTA06] *Borland Together Architect 2006*
http://www.borland.com/.

[Cat96] R. Cattell.
The Object Database Standard: ODMG-93, Release 1.2.
Morgan Kaufmann, 1996.

[CB74] D.D. Chamberlin y R.F. Boyce.
SEQUEL: A Structured English Query Language.
Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Michigan, May 1-3, 1974, 2 Volumes, pages 249--264. ACM, 1974.

[CDE+03] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer y C. Talcott
The maude 2.0 system.
Rewriting Techniques and Applications (RTA 2003), number 2706 in Lecture Notes in Computer Science, pages 76–87. Springer-Verlag, 2003

[CDF+86] M.J. Carey, D.J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. E. Richardson, y E.J. Shekita.
The Architecture of the EXODUS Extensible DBMS.
Proceedings of OODBS: 1986 International Workshop on Object-Oriented Database Systems, September 23-26, 1986, Asilomar Conference Center, Pacific Grove, California, USA, , pages 52--65. IEEE Computer Society, 1986.

[CM06] P. Cáceres y E. Marcos
Proyecto MIDAS
Grupo de Investigación Kybele, 2006
Universidad Rey Juan Carlos, Madrid

[Coc00] A. Cockburn.
Reexamining the cost of change curve
En Internet, 2000
http://www.xprogramming.com/xpmag/cost-of-change.htm/

[Coc01] A. Cockburn
Agile Software Development.
Addison-Wesley Professional, 1st edition. 2001

[Coc04] A. Cockburn
Crystal Clear : A Human-Powered Methodology for Small Teams.
Addison-Wesley Professional, 1st edition. 2004

[Cod] Codagen architect.
Sitio Web oficial: http://www.codagen.com/

[Cod70] E. F. Codd.
A relational model of data for large shared data banks.
Communications of ACM, 13(6):377--387, 1970.

[Coo04] S. Cook
Domain-Specific Modeling and Model Driven Architecture.
The MDA Journal, pages 80–94. 2004

[COR04] *Common Object Request Broker Architecture (CORBA/IIOP). Version 3.0.3.*
Object Management Group,2004

[CSH+98] G. Clossman, P. Shaw, M. Hapner, J. Klein, R. Pledereder, y B. Becker.
Java and Relational Databases: SQLJ (Tutorial).
Proceedings ACM SIGMOD Internacional Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA., page 500.
ACM Press, 1998.

[Cue01] J.M. Cueva Lovelle
Lenguajes, gramáticas y autómatas.
Editorial Servitec, 2001

[Cue98] J.M. Cueva Lovelle
Conceptos Básicos de Procesadores de Lenguaje
Cuaderno Didáctico número 10. Editorial Servitec, 1998

[CWM] Common Warehouse Metamodel
CWM Specification. Version 1.1.
Object Management Group, 2003

[Dev97] R. Devis Botella.
C++. STL-Plantillas-Excepciones-Roles y Objetos.
Paraninfo, 1997.

[DPC99] DPC. Dynamic Proxy Classes.
Sun Microsystems, Inc. Technical report, 1999
http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html.

[DW99] S.C. Drye y W.C. Wake.
Java Foundation Classes Swing Reference.
Manning Publications Co., 1999.

[Ecl06a] Eclipse Home Page
http://www.eclipse.org

[Ecl06b] Eclipse GMT home page.
http://www.eclipse.org/gmt/.

[EEK+99] H. Ehrig, G. Engels, H.J. Kreowski y G. Rozenberg
Handbook on graph grammars and computing by graph transformation
Applications, Languages and Tools. World Scientific, 1999

[Eff06a] S. Efftinge
OpenArchitectureWare 4.1 Check – Validation Language.
http://www.openarchitectureware.org/

[Eff06b] S. Efftinge
OpenArchitectureWare 4.1 Expressions Framework Reference.
http://www.openarchitectureware.org/

[Eff06c] S. Efftinge
OpenArchitectureWare 4.1 Extend Language Reference.
http://www.openarchitectureware.org/

[Eff06d] S. Efftinge
OpenArchitectureWare 4.1 Xpand2 Language Referente.
http://www.openarchitectureware.org/

[Ega88] D. E. Egan.
Individual differences in human-computer interaction.
Handbook of Human-Computer Interaction.
Elsevier Science Publishers, 1988.

[EHM+05] B. Evjen, S. Hanselman, F. Muhammad, S.S. Sivakumar, y D. Rader
Professional ASP.NET 2.0.
Wrox, 2005.

[EM99] A. Eisenberg y J. Melton.
Sql: 1999, formerly known as SQL3.
SIGMOD Record, 28(1):131--138, 1999.

[EV06a] S. Efftinge y M. Voelter
OpenArchitectureWare4.1 Workflow Engine Reference
http://www.openarchitectureware.org/

[EV06b] S. Efftinge y M. Voelter
OpenArchitectureWare 4.1 Xtext Reference Documentation
http://www.openarchitectureware.org/

[FEB03] J. Fisher, M. Ellis, y J. Bruce.
JDBC Tutorial and Reference, Third Edition.
Addison Wesley, 2003.

[Fer02] C. Fernández Acebal.
Lenguaje para el desarrollo Ágil de software. Un enfoque basado en patrones de diseño.
Trabajo de investigación, Universidad de Oviedo, Junio 2002.

[Fow00] M. Fowler.
Acronym pojo, 2000
http://www.martinfowler.com/bliki/POJO.html.

[Fow03] M. Fowler
UML Distilled: A Brief Guide to the Standard
Object Modeling Language. Addison Wesley, third edition. 2003

[Fow04] M. Fowler.
Inversion of control containers and the dependency injection pattern.
Martin Fowler's Bliki 2oo4
http://martinfowler.com/articles/injection.html.

[Fow05] M. Fowler.
The new methodology.
Technical report, 2005.
http://www.martinfowler.com/articles/newMethodology.html.

[Fow96] M. Fowler.
Analysis Patterns: Reusable Object Models.
Addison-Wesley Professional, 1st edition. 1996

[Fow99] M. Fowler.
Refactoring: Improving the Design of Existing Code.
Addison Wesley, 1999.

[Fra03] D.S. Frankel.
Model Driven Architecture. Applying MDA to Enterprise Computing.
OMG Press, 2003.

[Fre06] FreeMarker
Freemarker home page.
http://freemarker.sourceforge.net/.

[Fus97] M.L. Fussel.
Foundations of object relational mapping.
White paper, ChiMu Corporation, 1997.
http://www.chimu.com/publications/objectRelational.

[GF94] O. Gotel y A. Finkelstein
An analysis of the requirements traceability problem.
Proceedings of the IEEE International Conference on Requirements Engineering, pages 94–102. 1994

[GHJ+95] E. Gamma, R. Helm, R. Johnson, y J. Vlisides.
Design Patterns: Elements of Reusable Object Oriented Software.
Addison Wesley, 1995.

[GJS04] J. Gosling, B. Joy y G. Seele.
The Java Language Specification. Third Edition.
Addison Wesley, 2004.

[GMT] GMT. Generative Model Transformer
Sitio Web oficial: http://www.eclipse.org/gmt.

[Gou01] J. Gough.
Compiling for the .NET Common Language Runtime (CLR)
Prentice Hall, 2001.

[GSC+04] J. Greenfield, K. Short, S. Cook, S. Kent, y J. Crupi,
Software Factories: Assembling Applications with Patterns, models, Frameworks, and Tools.
Wiley, 1st edition, 2004

[HC01] G.T. Heineman y W. T. Councill
Component Based Software Engineering: Putting the Pieces Together.
Addison- Wesley Professional, 1st edition. 2001

[Heb05] M. Hebach
MDA with QVT.
Borland Together 2006 Presentation.

[Her03] J. Herrington
Code Generation in Action.
Manning Publications,2003

[Hib06] Hibernate. Hibernate Reference Documentation.
JBoss, Inc., 2006.
Sitio Web oficial: http://www.hibernate.org/

[How02] D. Howe
The Free Online Dictionary. "Virtual Machine", 2002
www.foldoc.org.

[IBM] IBM. International Business Machines.
Sitio Web oficial: http://www.ibm.com.

[INC] INCITS. International Committee for Information Technology Standards.
Sitio Web oficial: http://www.incits.org/

[INR05] INRIA A. Nantes
ATL: Atlas Transformation Language.
Specification of the ATL Virtual Machine v 0.1.,2005

[INR06] INRIA A. Nantes
ATL: Atlas Transformation Language.
ATL User Manual. Version 0.7., 2006

[ISO] *ISO. International organization for standarization.*
Sitio Web oficial: http://www.iso.org/.

[ISO00] *ISO 9000:2000, Quality management systems – Fundamentals and vocabulary.*
International Organization for Standardization.

[Jav03] *JavaBeans. Enterprise Java Beans 2.1 specification.*
Sun Microsystems, Inc. Technical report, 2003
http://java.sun.com/products/ejb/2.0.html.

[Jav05] *JavaBeans. JSR 220: Enterprise JavaBeansTM, version 3.0.*
Sun Microsystems, Inc. Technical report, 2005.
http://java.sun.com/products/ejb/docs.html.

[Jav05] *JSR-000244 JavaTM Platform, Enterprise Edition 5 Specification*.
Sun Microsystems,2005

[Jav96] *JavaBeans. JavaBeans 1.0 API Specification*.
Sun Microsystems, Inc. Technical report, 1996.
http://java.sun.com/products/javabeans/docs/spec.htm

[JBR99] I. Jacobson, G.Booch y J. Rumbaugh
The Unified Software Development Process.
Addison-Wesley. 1999

[JCR97] *Java Core Reflection. API and Specification.*
Sun Microsystems, Inc, 1997.
http://java.sun.com/j2se/1.3/docs/guide/reflection/.

[JDO03] *JDO. Java Data Objects Specification.*
Sun Microsystems, Inc. Technical report, 2003.
http://java.sun.com/products/jdo/.

[JF02] L. Joyanes Aguilar y M. Fernandez Azuela
C# manual de programacion
Mcgraw-hill / interamericana de España, 2002.

[JHA+05] R. Johnson, J. Hoeller, A. Arendsen, T. Risberg, y C Sampaleanu.
Professional Java Development with the Spring Framework.
Wrox, 2005

[Jor04] M. Jordan.
A comparative study of persistente mechanisms for the java platform.
SML technical report series, Sun Microsystems, Inc, 2004.
http://research.sun.com/techrep/2004/abstract-136.html.

[JOS03] *Java Object Serialization Specification.*
Sun Microsystems, Inc, 2003.
http://java.sun.com/.

[JSF] *JSF. Java Server Faces*
Sun Microsystems, Inc
http://java.sun.com/javaee/javaserverfaces/index.jsp.

[JSP] *JSP. Java Server Pages*.
Sun Microsystems, Inc
http://java.sun.com/products/jsp/index.jsp#specs.

[JUn] *JUnit.*
Sitio Web oficial: http://www.junit.org/index.htm.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.M Loingtier y J. Irwin
Aspect oriented programming.
Proceedings of ECOOP'97 Conference. Finland. 1997

[KP88] G. E. Krasner y S. T. Pope.
A cookbook for using the model-view controller user interface paradigm in SMALLTALK-80.
J. Object Oriented Program., 1(3):26--49, 1988.

[Kra96] D. Kramer.
The java platform. A white paper.
White paper, Sun Microsystems, Inc, Mayo 1996.
http://java.sun.com/docs/white/platform/javaplatformTOC.doc.html.

[Kru00] P. Kruchten
The Rational Unified Process. An Introduction.
Addison Wesley, 2nd edition.2000

[Kru01] S. Krug
Don't make me think! A common sense approach to web usability
Pearson Educación, 2001.

[KWB03] A. Kleppe, J. Warmer y W. Bast
MDA Explained. The Model Driven Architecture: Practice and Promise.
Addison-Wesley. 2003.

[Lar03] C. Larman
Agile and Iterative Development: A Manager'sGuide.
Addison-Wesley Professional, 1st edition. 2003

[Lar04] C. Larman
Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.
Addison Wesley Professional, third edition. 2004

[Lid02] S Lidin.
Inside Microsoft .NET IL Assembler.
Microsoft Press, 2002.

[LMP87] B.G. Lindsay, J. McPherson, y H. Pirahesh.
A Data Management Extension Architecture.
SIGMOD Conference: Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987, pages 220--226. ACM Press, 1987.

[Lop06] B. López Pérez
Adaptación dinámica de persistencia de objetos mediante reflectividad computacional.
Tesis Doctoral (inédita), Universidad de Oviedo, 2006.

[Mai89] D. Maier.
Why Isn't There an Object-Oriented Data Model?
En IFIP Congress, pages 793--798, 1989.

[Man05] K. Mann
JavaServer Faces in Action.
Manning Publications, 2005

[Man06] J. Manrubia Díez
Desarrollo de Software dirigido por modelos
Memoria de investigación subvencionada por FICYT y dirigida por J. M. Cueva Lovelle.
Inédita, 2006

[Mar01] A.B. Martinez Prieto.
Un Sistema de Gestión de Bases de Datos Orientadas a Objetos sobre una Maquina Abstracta Persistente.
Tesis Doctoral, Universidad de Oviedo, 2001.

[MB02] S. J. Mellor y M. J. Balcer
Executable UML: A Foundation for Model-Driven Architecture.
Addison Wesley, 2002

[MC99] A.B. Martínez Prieto y J.M. Cueva Lovelle.
Técnicas de indexación para las bases de datos orientadas a objetos.
Novática, Monográfico Bases de Datos Avanzadas (140), 1999.

[MCC] MCC. Model Component Compiler
Sitio Web oficial:
http://www.inferdata.com/products/mcc/mdac.html.

[MDR] Metadata Repository (MDR).
NetBeans
http:// mdr.netbeans.org/.

[Mel04] S. J. Mellor
Agile MDA.
The MDA Journal, pages 144–160, 2004

[Mid] Middlegen
Sitio Web oficial: http://boss.bekk.no/boss/middlegen/index.html.

[MM03] J. Miller y J. Mukerji
MDA Guide Version 1.0.1.
Object Management Group, 2003

[Mod] ModFact.
Sitio Web oficial: http://modfact.lip6.fr/ModFactWeb/index.jsp.

[MOF05] *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final Adopted Specification.*
Object Management Group, 2005

[MOF06] *MOF. Meta-Object Facility 2.0.*
Object Management Group. Septiembre 2006.
http://www.omg.org/cgi-bin/doc?formal/2006-01-01.

[MRM03] J.S. Miller, S. Ragsdale, y J. Miller.
The Common Language Infrastructure Annotated Standard.
Addison-Wesley, Octubre 2003.

[MRW77] J. McCall, P. Richards y G. Walters
Factors in software quality.
Rome Air Development Center, RADC TR-77-369, 1977.

[MS93] J. Melton y A. Simon.
Understanding the New SQL: A Complete Guide.
Morgan Kaufmann, 1993.

[MSB+04] G.J. Myers, C. Sandler, T. Badgett y T.M. Thomas
The Art of Software Testing.
John Wiley & Sons, 2nd edition. 2004

[MSC] Microsoft Corporation.
Sitio Web oficial: http://www.microsoft.com/.

[MSU+04] S.J. Mellor, K. Scott, A. Uhl, y D. Weise.
MDA Distilled. Principles of Model-Driven Architecture.
Object Technology. Addison-Wesley, 2004.

[MTL] MTL Engine.
Sitio Web oficial: http://modelware.inria.fr/

[MVB] Microsoft Visual Basic. Centro de desarrollo de Visual Basic.
Sitio Web oficial: http://msdn.microsoft.com/vbasic/.

[MVC#] Microsoft Visual C#. Centro de desarrollo de C#.
Sitio Web oficial: http://msdn.microsoft.com/vcsharp/.

[MVC+] Microsoft Visual C++. Centro de desarrollo de C++.
Sitio Web oficial: http://msdn.microsoft.com/visualc/.

[MVJ] Microsoft Visual J#. Centro de desarrollo de J#.
Sitio Web oficial: http://msdn.microsoft.com/vjsharp/

[MW03] P. Milne y K. Walrath.
Long-term persistence for javabeans.
Technical report, Sun Microsystems, Inc, 2003.
http://java.sun.com/products/jfc/tsc/articles/persistence.

[MZ96] E. May y B. Zimmer
The evolutionary development model for software.
Hewlett-Packard Journal, 47(4):39–45, 1996

[Net] NetBeans IDE.
Home Page
http://www.netbeans.org/.

[Nie01] J. Nielsen.
Usabilidad, Diseño De Sitios Web.
Prentice Hall, 2001.

[oaW06] *openArchitectureWare (oAW)*
Home Site.
http://www.openarchitectureware.org/.

[OGL+99] L. Olsina, D. Godoy, G. Lafuente, y G. Rossi.
Assessing the quality of academic websites: A case study.
The New Review of Hypermedia and Multimedia, 5:81--103, 1999.

[OMA+99] F. Ortín Soler, A.B. Martínez Prieto, D. Álvarez Gutiérrez y J.M. Cueva Lovelle.
Diseño de un Sistema de Persistencia Implícita Mediante Reflectividad Computacional.
JISBD: IV Jornadas de Ingeniería del Software y Bases de Datos, Cáceres, del 24 al 26 de noviembre de 1999, pages 39--50. Universidad de Extremadura, 1999.

[OME] *OMELET*.
Sitio Web oficial: http://www.eclipse.org/omelet.

[OMG] *OMG. Object management group*.
Sitio Web oficial:http://www.omg.org/

[Opea] *OpenMDX*.
Sitio Web oficial: http://www.openmdx.org/index.html.

[Opeb] *Openmodel*.
Sitio Web oficial: http://openmodel.tigris.org/.

[Opt] *OptimalJ*.
Sitio Web oficial:
http://www.compuware.com/products/optimalj/default.htm.

[ORA] *Oracle Corporation*.
Sitio Web oficial: http://www.oracle.com.

[Ora99] *Oracle 8iTM Objects and Extensibility Option. Features Overview.* Oracle Corporation, 6585 Merchant Place, Suite 100, Warrenton, VA 20187, USA, 1999. http://www.oracle.com/.

[Ort01] F. Ortín Soler *Sistema Computacional de Programación Flexible Diseñado sobre una Máquina Abstracta Reflectiva no Restrictiva.* Tesis doctoral inédita, Universidad de Oviedo,2001

[PC05] B. C. Pelayo García-Bustelo y J. M. Cueva Lovelle. *Evolución de las directrices de accesibilidad del W3C.* En Libro de Actas del III Simposio Internacional de Sistemas de Información en la Sociedad del Conocimiento. Instituto Tecnológico de las Américas (República Dominicana), 2005.

[PC06a] B. C. Pelayo García-Bustelo y J. M. Cueva Lovelle. *Arquitecturas dirigidas por modelos (MDA). El framework C3NET.* II Congreso Internacional de Ingeniería de Computación y Sistemas (IICIIS). Trujillo, PERÚ, 2006.

[PC06b] B. C. Pelayo García-Bustelo y J. M. Cueva Lovelle. Usabilidad, accesibilidad y métricas de sitios Web. II Congreso Internacional de Ingeniería de Computación y Sistemas (IICIIS). Trujillo, PERÚ, 2006.

[PCJ06] B. C. Pelayo García-Bustelo, J. M. Cueva Lovelle, y A.A. Juan Fuente. *C3NET: Smart Environment For .NET Code Generation Using MDA.* En WORLDCOMP'06 Proceedings: The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing (composed of 28 Joint Conferences), Las Vegas, USA, 2006.

[PCS+05] B.C. Pelayo García-Bustelo, J.M. Cueva Lovelle, M.C. Suárez Torrente, y A.A. Juan Fuente. *C3NET: Framework para la construcción de MDA en la Plataforma .NET.* III Simposio Internacional de Sistemas de Información en la Sociedad del Conocimiento. Instituto Tecnológico de las Américas (República Dominicana), 2005.

[Pel04] B. C. Pelayo García-Bustelo. *C3NET: Herramienta para el desarrollo de lenguajes en la Plataforma .NET.* Proyecto de Fin de Carrera de Ingeniería Informática, Escuela Politécnica Superior de Ingeniería de Gijón, 2004

[PGR+04] J. Padrón Lorenzo, A. Estévez García, J.L Roda García y F. García López *BOA, un framework MDA de alta productividad.* I Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones DSDM, 2004

[PGS+05] J. Padrón Lorenzo, J. García Luna, E. Sánchez Rebull, y A. Estévez García
Implementación de un motor de transformaciones con soporte MOF 2.0 QVT.
II Taller sobre Desarrollo Dirigido por Modelos. MDA y Aplicaciones. DSDM, 2005

[Pla03] D.S. Platt.
Introducing Microsoft .Net, Third Edition.
Microsoft Press, 2003.

[Pop03] R. Popma
JET Tutorial.
Eclipse Corner,2003
http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html.

[QHB+06] P. Queralt , Pascual, L. Hoyos, A. Boronat, J. Carsí y I. Ramos
Un motor de transformación de modelos con soporte para el lenguaje QVT Relations
Desarrollo de Software Dirigido por Modelos, DSDM.
Sitges, Spain, 2006

[QVT05] *MOF QVT. Final Adopted Specification. ptc/05-11-01.*
Object Mangement Group, 2005

[RAC+02] S. Robinson, K.S. Allen, O. Cornes, J. Glynn, Z. Greenvoss, B. Harvey, C. Nagel, M. Skinner y K. Watson.
Professional C#. 2nd Edition.
Wrox, Marzo 2002.

[Ree] Trygve Reenskaug.
Sitio Web oficial: http://folk.uio.no/trygver/.

[Ree00] G. Reese.
Database Programming with JDBC and Java, Second Edition.
O'Reilly & Associates, 2000.

[Ree92] J. M. Reeves
What is software design?
C++ Journal. 1992
Disponible en :
http://www.bleading-edge.com/Publications/C++Journal/Cpjour2.htm.

[RFW+04] C. Raistrick, P. Francis, J. Wright, C. Carter y I. Wilkie.
Model Driven Architecture with Executable UML.
Cambridge University Press, 2004.

[Roo03] R. M. Roos
Java Data Objects.
Addison Wesley, 2003.

[Roy70] W. W. Royce
Managing the Development of Large Software
Systems. IEEE Wescon, pages 1–9, 1970

[RSB05] E. Roman, R. Patel Sriganesh, y G. Brose.
Mastering Enterprise JavaBeans Third Edition.
Wiley Publishing, 2005.

[San06] M.A. Sánchez Vidales
Una recomendación para el desarrollo de software en un contexto de negocio bajo demanda de acuerdo a la especificación MDA (Model Driven Architecture) y la arquitectura SOA (Service Oreinted arquitectura)
Tesis Doctoral (inédita), Universidad Pontificia de Salamanca, 2006.

[San98] R. E. Sanders.
ODBC 3.5 Developers Guide.
McGraw-Hill Companies, 1998.

[SB01] K. Schwaber y M. Beedle
Agile Software Development with SCRUM.
Prentice Hall, 1st edition. 2001

[SFP04] R. Soley, D.S. Frankel y J. Parodi.
The MDA Journal: Model Driven Architecture Straight From The Masters.
Meghan Kiffer Pr, 2004.

[SFJ05] M. A. Sánchez Vidales, A. Fermoso García y L. Joyanes Aguilar
From the platform independent model (PIM) to the final code model (FCM) according to the model driven architecture (MDA).
Proceedings de IADIS International Conference on Applied Computing, Algarve, Portugal, 2005. Páginas 417-420

[SM88] S. Shlaer y S. J. Mellor
Object-oriented Systems Analysis: Modeling the World in Data.
Prentice Hall,1988

[SM91] S. Shlaer y S. J. Mellor
Object Lifecycles: Modeling the World in States.
Yourdon Press Computing Series, 1st edition, 1991

[Sma] *SmallTalk*.
Sitio Web oficial: http://www.smalltalk.org/main/

[Sol00] R. Soley and OMG Staff Strategy Group
Model Driven Architecture. White Paper. Draft 3.2.
Object Management Group, 2000.

[Som01] I. Sommerville
Software Engineering.
Addison-Wesley, 6th edition. 2001

[Sos] SosyInc.
Sitio Web oficial: http://www.sosyinc.com/.

[SPR] SPRING.
Sitio Web oficial: http://www.springframework.com/.

[Ste04] L. S. Stepanian
Solving the Requirements Traceability Problem: A Comparison of Two Pre-Requirements Specification Traceability Enablers.
Computer Systems Research Group. Department of Computer Science.
University of Toronto. 2004 Disponible en
http://www.cs.toronto.edu/~levon/projects/RTSurvey/RT.pdf.

[Sto86] M.l Stonebraker.
Inclusion of New Types in Relational Data Bases Systems.
Proceedings of the Second International Conference on Data Engineering, February 5-7, 1986, Los Angeles, California, USA, pages 262--269. IEEE Computer Society, 1986.

[STR] STRUTS. Apache struts project.
Sitio Web oficial: http://struts.apache.org/.

[Str98] B. Stroustrup.
The C++ Programming Language. Third Edition.
Addison Wesley, Octubre 1998.

[SUN] Sun Microsystems Inc.
Sitio Web oficial: http://www.sun.com.

[TFR02] D. Turk, R. France y B. Rumpe
Limitations of Agile Software Processes.
Proceedings of Third International Conference on eXtreme Programming and Agile Processes in Software Engineering. 2002

[TL02] T. L. Thai y H. Lam
.NET Framework Essentials.
O'Reilly, second edition, 2002

[TLM06] K. Tatroe, R. Lerdorf y P. MacIntyre
Programming PHP.
O'Reilly Media, 2nd edition, 2006

[Tyr01] S. Tyrrell
The many dimensions of the software process.
ACM Crossroads. 2001

[UD] Universal Design. The center for universal design.
Sitio Web oficial: http://www.design.ncsu.edu/cud/.

[UML] UML. Unified Modeling Language.
Object Management Group.
Sitio Web oficial: http://www.uml.org

[UMT] UMT. Model Transformation Tool.
Sitio Web oficial: http://umt-qvt.sourceforge.net/

[Van05] G. Vanderburg, G.
A simple model of agile software processes or extreme programming annealed.
Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications, 40(10):539–545. 2005

[VIA06] *The VIATRA2 Model Transformation Framework*.
http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html.

[Vis] VisualWADE.
Sitio Web oficial: http://www.visualwade.com/.

[VP03] D. Varro y A. Pataricza
VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and uml.
Journal of Software and Systems Modeling, pages 187–210, 2003

[W3C] W3C. World Wide Web Consortium.
Sitio Web oficial: http://www.w3.org/.

[WK03] J. Warmer y A. Kleppe.
The Object Constraint Language. Second Edition. Getting your models ready for MDA.
Object Technology. Addison-Wesley, 2003.

[WM99] I. T. Wilkie y S. J. Mellor
A Mapping from Shlaer-Mellor to UML.
Kennedy Carter Ltd/Project Technology, 1999

[WR03] C. Walls y N. Richards.
XDoclet in Action.
Manning Publications, 2003.

[XDo] XDoclet.
Sitio Web oficial: http://xdoclet.sourceforge.net/xdoclet/index.html.

[XFo04] XForms.
W3C Working Draft, 2004
Sitio Web oficial: http://www.w3.org/MarkUp/Forms/

[XMI05] *XMI. XML Metadata Interchange, v2.1.*
Object Management Group., Septiembre 2005.
http://www.omg.org/cgi-bin/doc?formal/2005-09-01.

[XML04] *XML. Extensible Markup Language (XML) 1.0*
World Wide Web Consortium, 2004.
http://www.w3.org/XML/.