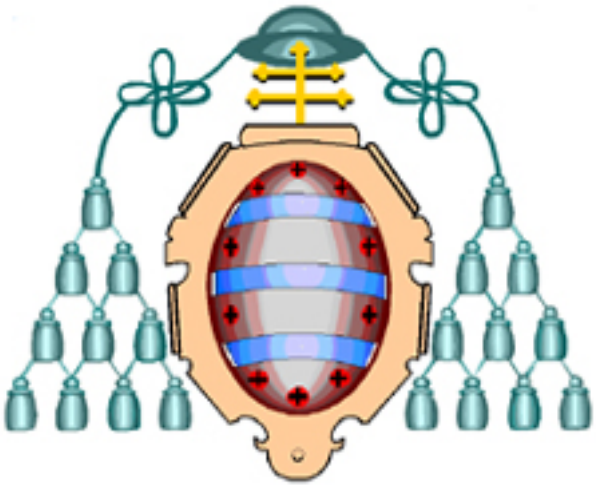
UNIVERSIDAD DE OVIEDO



DEPARTAMENTO DE INFORMÁTICA

Tesis Doctoral

MDCI: Model-Driven Continuous Integration

AUTOR: Vicente García Díaz
DIRECTORES: Dr. Juan Manuel Cueva Lovelle
Dra. Begoña Cristina Pelayo García-Bustelo

Oviedo, Junio de 2011

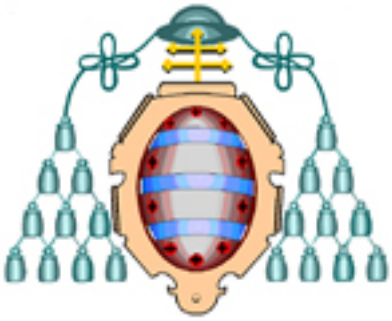
Universidad de Oviedo

MDCI:
Model-Driven Continuous Integration

Vicente García Díaz

Depósito Legal: AS. 03543-2012
http://hdl.handle.net/10803/80298

MDCI: Model-Driven Continuous Integration



UNIVERSIDAD DE OVIEDO

DEPARTAMENTO DE INFORMÁTICA

Tesis Doctoral

Presentada por

Vicente García Díaz

para la obtención del título de Doctor por la Universidad de Oviedo

Dirigida por

Dr. Juan Manuel Cueva Lovelle
Dra. Begoña Cristina Pelayo García-Bustelo

Agradecimientos

Esta Tesis ha sido llevada a cabo gracias a la ayuda de muchas personas que, directa o indirectamente, la han hecho posible.

Quiero agradecer a mis Directores, Dr. D. Juan Manuel Cueva Lovelle y Dra. Begoña Cristina Pelayo García-Bustelo, su disposición a orientarme y guiarme durante este trabajo.

Agradezco a mis compañeros del grupo del OOTLab sus ánimos y su colaboración para la realización de esta investigación.

Y por supuesto, a mi familia y a todas las personas que no he nombrado, pero que me han ayudado a superar los obstáculos encontrados durante este camino.

Resumen

El propósito de esta Tesis es *llevar a cabo un proceso en el que se aplique la práctica de la integración continua en un desarrollo de software dirigido por modelos de forma eficiente*, mediante el cual los desarrollos de software puedan beneficiarse conjuntamente de las mejoras y ventajas que proporcionan la aproximación de desarrollo de la ingeniería dirigida por modelos y la práctica de la integración continua.

La aproximación de la ingeniería dirigida por modelos es el último salto natural de la ingeniería del software en cuanto a la búsqueda de métodos de desarrollo que elevan el nivel de abstracción hasta el punto en el que los expertos de un dominio de conocimiento, ajenos al mundo informático, son capaces de guiar y cambiar la lógica de los sistemas informáticos.

La práctica de la integración continua es una recomendación de las principales metodologías de desarrollo, que tiene como objetivo la realización de integraciones automáticas del software en etapas tempranas del desarrollo, ofreciendo ventajas como la reducción del riesgo intrínseco que, dado su carácter temporal y único, tienen todos los proyectos.

Con la unión de la ingeniería dirigida por modelos y de la práctica de la integración continua se busca ofrecer, a los equipos de desarrollo que trabajan utilizando algún tipo de iniciativa de la ingeniería dirigida por modelos, la posibilidad de integrar de forma continua y distribuida sus desarrollos. Al mismo tiempo, los clientes, verdaderos expertos del dominio de conocimiento en su ámbito de negocio, se benefician del aumento del nivel de abstracción de las técnicas de desarrollo para que ellos mismos, y de forma transparente, sean capaces de modificar su propio sistema informático sin la ayuda de personal técnico ajeno a su negocio, ahorrando así tiempo y costes.

Para cumplir con el objetivo de esta Tesis Doctoral se construye un prototipo que salva los impedimentos actuales que no permiten la unión entre estos dos nuevos activos de la ingeniería del software. Los principales problemas encontrados están relacionados con la selección de una iniciativa de desarrollo apropiada, los siste-

mas de control de versiones especialmente adaptados para trabajar con modelos, la generación incremental de artefactos a partir de modelos y la adaptación a las herramientas actuales de integración continua de forma optimizada. La separación del trabajo realizado en diferentes bloques permite ofrecer soluciones de forma tanto aislada como en conjunto, dando lugar a un trabajo iterativo e incremental de comienzo a fin.

Para analizar las ventajas que ofrece la propuesta de este trabajo frente a otras posibilidades de desarrollo, se realiza una evaluación mediante la creación de diferentes casos de prueba en los que la medición de diferentes parámetros ofrecen una estimación numérica de las ventajas reales obtenidas. El análisis descriptivo, el contraste de hipótesis y las técnicas de regresión permiten una mejor interpretación de los resultados.

Finalmente, se define el proceso, objetivo último de este trabajo, mediante la respuesta a diferentes preguntas planteadas, que facilitan su comprensión y entendimiento.

Palabras reservadas

Ingeniería dirigida por modelos, integración continua, modelo, metamodelo, comparación de modelos, sistema de control de versiones, generador de artefactos, generación incremental, lenguaje de dominio específico, Model-Driven Architecture, Meta-Object Facility, Ecore.

Abstract

The purpose of this Thesis is to create a *process in which the continuous integration practice can be applied to a model-driven software development in an effective way*, through which software developments can benefit jointly and simultaneously from the improvements and advantages provided by the model-driven engineering development approach and the continuous integration practice.

The model-driven engineering approach is the last natural step of software engineering in the search for development approaches that raise the level of abstraction to the point that experts in a domain of knowledge, outside the computer world, are able to guide and change the logic of computer systems.

The continuous integration practice is a recommendation of the most widely accepted development methodologies that aims to carry out automatic software integrations in early stages of development, offering benefits such as reducing the inherent risk that, given its unique nature, every project has.

By merging the model-driven engineering and the continuous integration practice, the aim is to provide to development teams that work using some kind of model-driven engineering initiative, the possibility to integrate their developments in a continuous and distributed way. At the same time, customers, the real experts in the domain of knowledge in their field of business, can benefit from the increased level of abstraction in developing techniques. Thus, they, in a transparent manner, are able to modify their own computer system without the help of external technical staff, so saving time and costs.

To meet the objective of this Thesis, a prototype which saves all the current constraints that do not allow the union between these two new tools of software engineering is build. The main problems found were related to the selection of an appropriate development initiative, the version control systems specially adapted to working with models, the incremental generation of artifacts from models, and the optimized adaptation to existing continuous integration tools. The separation of work in different blocks can provide solutions, both in isolation or in conjunction, resulting in an iterative and incremental work from beginning to end.

To analyze the benefits of the proposal in this work compared to other development possibilities, an evaluation is performed by creating different test cases in which the measurement of different parameters can give a numerical estimate of the real benefits obtained. The descriptive analysis, the hypothesis testing, and regression techniques allow a better interpretation of results.

Finally, the process, the main objective of this work, is defined by answering various questions posed to facilitate its comprehension and understanding.

Keywords

Model-driven engineering, continuous integration, model, metamodel, model comparison, version control system, generator of artifacts, incremental generation, domain-specific language, Model-Driven Architecture, Meta-Object Facility, Ecore.

Índice general resumido

Índice general

Índice de figuras

Índice de tablas

Índice de archivos de código fuente

Bloque I

Planteamiento del problema

Índice del bloque

CAPÍTULO 1

Introducción

En este capítulo se presentan las bases para el desarrollo de esta Tesis Doctoral, que será posteriormente demostrada mediante la creación y utilización de un prototipo para la evaluación de diferentes casos de uso. Se establece la motivación, una hipótesis de partida, el objetivo a cumplir y las principales actividades del resto del trabajo.

⁎ ⁎ ⁎ ⁎

1.1. Motivación

El aumento de la complejidad del desarrollo de software es un problema cada vez más importante, dado que los clientes piden software cada vez más sofisticado, con menos errores y con mayores capacidades [Groth, 2004]. Adicionalmente, debido a su rápida expansión, los sistemas software se han hecho necesarios en prácticamente todos los dominios o áreas profesionales que existen en la actualidad. Este hecho, genera ciertos problemas, por ejemplo equipos de desarrollo expertos en un área concreta que pasado un tiempo tienen que realizar otro proyecto en otro área diferente con el consiguiente periodo de adaptación necesario. Además, debido a la existencia de muchas plataformas tecnológicas, se puede pensar que el desarrollo podría estar mucho más optimizado si pudiéramos reutilizar no sólo parte del código que se genera a diario para las diferentes plataformas, sino que también pudiéramos reutilizar la experiencia en un dominio concreto, y no sólo nuestra experiencia, también la experiencia que otros han adquirido en el dominio con el paso del tiempo [Caldiera and Basili, 1991].

Así, para realizar los cada día más complejos desarrollos de software, se puede incrementar el número de desarrolladores. Sin embargo, mucho más óptimo que aumentar el volumen de desarrolladores, sería aumentar nuestra capacidad de producción mediante la industrialización del software [Mcilroy, 1968], al igual que lo han hecho otros muchos sectores. Por ejemplo, el sector del automóvil, que ha pasado de producir los automóviles de manera artesanal a hacerlo de manera automatizada gracias a personas como Henry Ford y su famoso Ford Model T que data de 1908.

La ingeniería del software continuamente ofrece nuevas herramientas que, adecuadamente empleadas, pueden ayudarnos en la difícil tarea de desarrollar software cumpliendo con la clásica triple restricción de la gestión de proyectos (alcance, tiempo y coste) citada en numerosas fuentes como [Frame, 2002] o [PMI, 2005]. Así, en los últimos años, ha aparecido una nueva aproximación de desarrollo de software denominada MDE (Model-Driven Engineering o Ingeniería Dirigida por Modelos) [Kent, 2002], que eleva el nivel de abstracción de los lenguajes tradicionales mediante el empleo de modelos permitiendo, de ese modo, utilizar conceptos cercanos a los dominios de los problemas. También está en auge la práctica CI (Continuous Integration o Integración Continua) [Herbsleb and Grinter, 1999], la cual permite tener un mayor control del estado del proyecto y detectar problemas y errores en etapas iniciales del ciclo de desarrollo.

La principal motivación de este trabajo tiene su origen en [García-Díaz et al., 2008], un caso real de aplicación de MDE en el que se minimizan

los tiempos de desarrollo de sistemas de trazabilidad alimentaria adaptados a las necesidades específicas de cada uno de los posibles clientes. Sin embargo, durante la realización de dicho trabajo, se detectó la ausencia de tecnología adecuada para permitir utilizar satisfactoriamente la práctica de la integración continua en un proceso de desarrollo mediante iniciativas dirigidas por modelos. Así, el equipo de desarrollo no ha podido utilizar plenamente la práctica de la integración continua debido a que el desarrollo estaba *dirigido* por modelos y no *basado* por modelos, como suele ser habitual[1].

1.2. Hipótesis

Trabajos recientes como [Völter and Stahl, 2006] o [Duvall et al., 2007] han demostrado que la aproximación de desarrollo de la ingeniería dirigida por modelos y la práctica de la integración continua, por separado, mejoran la productividad y la calidad del desarrollo de software. Sin embargo, el proceso de integración continua no puede utilizarse adecuadamente con el paradigma de desarrollo dirigido por modelos debido a que las tecnologías actuales presentan deficiencias.

Cuando se aplica CI en un desarrollo de software tradicional, los diferentes miembros del equipo de desarrollo obtienen una serie de ventajas como reducción de riesgo, eliminación de *bugs*, mejores relaciones con los clientes, aumento de la moral del equipo, estimaciones más acertadas, disponibilidad inmediata de la última versión del código, mejora de las capacidades colaborativas, reducción de costes, realización automática de pruebas o informes, etc. [Fowler, 2009a]. Un factor clave es que las herramientas de CI permiten construir y desplegar el código de forma automática y transparente en el momento en el que un desarrollador realiza un cambio (véase sección 17.1).

Por otro lado, cuando se utiliza la aproximación de desarrollo MDE, una de las principales ventajas es que son los expertos en un dominio concreto de conocimiento los que pueden realizar programas (o una parte de ellos), a través de una herramienta de modelado [Völter, 2003a]. Sin embargo, necesitan la ayuda de personal técnico, típicamente de un equipo de desarrollo, para generar artefactos a partir de los modelos, integrar los artefactos generados con el resto del código fuente de una aplicación, compilar cada cambio, realizar pruebas, realizar modificaciones en bases de datos, generar informes de rendimiento o, por ejemplo, volver a desplegar

[1]La diferencia entre *dirigido* y *basado* radica en que el término *dirigido* indica que los modelos son artefactos de primera mano en el proceso de desarrollo. Por otra parte, el término *basado* sugiere que los modelos son únicamente la documentación a partir de la cual se comienza el desarrollo de un sistema software

la aplicación.

A partir de las dos ideas, integración continua e ingeniería dirigida por modelos, se plantea la siguiente hipótesis de partida:

> Es posible que los *expertos en un dominio de conocimiento* sean capaces de crear, desplegar y cambiar satisfactoriamente aplicaciones software sin necesidad de ser ayudados por *expertos tecnológicos*.

1.3. Objetivo

De la hipótesis citada se deriva el objetivo general de esta Tesis, que se cita a continuación:

> Llevar a cabo un proceso en el que se aplique la práctica de la integración continua en un desarrollo de software dirigido por modelos de forma eficiente. Así se conseguirán dos **sub-objetivos** principales:
>
> 1. ***Permitir que los expertos tecnológicos apliquen integración continua durante el desarrollo***. Que los diferentes miembros del equipo de desarrollo puedan integrar de manera continua y distribuida sus desarrollos dirigidos por modelos (al igual que lo pueden hacer mediante la programación tradicional).
>
> 2. ***Permitir que expertos en un dominio de conocimiento modifiquen una aplicación***. Que diferentes expertos en un dominio modifiquen de manera distribuida modelos de dicho dominio, provocando que las aplicaciones se modifiquen dinámica y transparentemente para ellos. Así, se permitiría que sean los expertos en el dominio de conocimiento los que creen, despliegen o cambien las aplicaciones en función de las necesidades del negocio del cliente. *Este es el objetivo que, una vez cumplido, validaría la hipótesis de partida.*

1.3.1. Apoyo de la comunidad científica

Existen varias opiniones de revisores de revistas como Journal of Systems and Software y Science of Computer Programming, indexadas en el JCR (Journal Citation Reports) del Thomson Reuters ISI (Institute for Scientific Information)[2], que han apoyado la hipótesis y el objetivo general de esta Tesis. Algunas de las opiniones recibidas de los expertos en el área han sido:

[2]http://www.isinet.com/

"The problem is relevant and as it is formulated by the authors addresses a real need in the development of software based on the idea of considering models as first-class entities which are usually subject to a constant and persistent evolution."

"The work presented in the manuscript is interesting and valuable...the work is an excellent example of good software engineering work."

"The objectives are interesting, challenging, and of interest for the software and MDE community...thus two basic ingredients are required as correctly suggested by the authors, i.e., an effective model versioning control and an incremental generation of artifacts."

"The research topic is very interesting and relevant in the area of software maintenance and model-driven engineering."

"The paper discusses a real challenge: how to improve MDE to use it in a software industrial process."

"This paper is very original, combining MDE and CI is a very interesting idea."

"The idea of bridging the gap between MDE and CI is interesting."

1.4. Actividades principales

Para alcanzar el objetivo y validar la hipótesis de partida, se plantean una serie de actividades, las cuales permitirán tener una mejor visión del trabajo realizado y de los beneficios obtenidos. Dichas actividades, que se describen a continuación, están basadas en las carencias detectadas en [García-Díaz, 2009], que impiden una correcta unión entre la práctica CI y la aproximación MDE.

1.4.1. Elección de una iniciativa MDE

Es posible desarrollar aplicaciones utilizando el paradigma MDE mediante el empleo de diferentes iniciativas que, aunque parecidas, tienen diferencias a nivel conceptual y tecnológico. Por ejemplo, MDA (Model-Driven Architecture) [Miller et al., 2003] se basa en el uso de estándares. Otras iniciativas, como las Software Factories [Greenfield and Short, 2003], son más productivas debido al beneficio de

una mayor integración con la plataforma para la cual se quiera desarrollar el software y la posibilidad de utilizar un mayor nivel de abstracción [Tolvanen, 2008], aunque generalmente no emplean estándares.

El problema es que las herramientas existentes para trabajar con modelos están enfocadas a trabajar con tecnologías muy diversas, lo que generalmente conlleva a tener que elegir entre trabajar con estándares o trabajar con herramientas orientadas a una determinada plataforma. Idealmente, habría que llegar a una solución de compromiso que favorezca el uso de estándares pero que permita emplear, al mismo tiempo, tecnologías y herramientas orientadas a una determinada plataforma. Para lograrlo, se podrían realizar puentes o mapeos entre metamodelos como por ejemplo el evaluado en [Tolosa, 2009] entre las DSL Tools y KM3 (Kernel Meta Meta Model).

El beneficio de solventar el problema de cara a cumplir el objetivo de este trabajo sería que se podrían utilizar la mayoría de tecnologías y herramientas disponibles para trabajar con modelos, estén o no basadas en estándares. De ese modo, se lograría una independencia tecnológica para el resto de actividades de este trabajo.

1.4.2. Detección de nuevas versiones en los VCSs

Los VCSs (Version Control Systems o Sistemas de Control de Versiones) son herramientas básicas para realizar la práctica de la integración continua, dado que el proceso de integración continua suele comenzar una vez que el sistema de control de versiones detecte que hay una nueva versión de los activos de desarrollo en el repositorio gestionado por él.

El problema es que los sistemas de control de versiones utilizados por las herramientas de CI no son adecuados para trabajar con modelos porque no identifican correctamente las diferencias entre las diferentes versiones de un modelo [Oliveira et al., 2005, Murta et al., 2008, Altmanninger et al., 2008]. Habría que adaptar la herramienta de CI para trabajar con un sistema de control de versiones adaptado para modelos. El principal inconveniente es que aún no existen ni productos finales ni prototipos funcionales completamente implementados para tal fin.

El beneficio de solventar el problema de cara a cumplir el objetivo de este trabajo sería que se reduciría el número de ocasiones que la herramienta de CI identifica nuevas versiones de forma errónea (o no las identifica correctamente cuando realmente las hay) de los modelos de entrada manipulados por los usuarios de las herramientas de modelado. Por tanto, el número de veces que la herramienta de integración continua ejecuta los *scripts* de construcción del software estaría más cercano a las necesidades reales.

1.4.3. Integración de MDE con herramientas de CI

La práctica y las herramientas de CI han evolucionado mucho con el paso del tiempo [Duvall et al., 2007], y suelen tener un mecanismo para ofrecer retroalimentación a los usuarios de las mismas. Generalmente, las herramientas tienen soporte nativo para muchas utilidades como pruebas unitarias, generación de informes o análisis de código.

Una gran ayuda de las herramientas de CI es que ofrecen una interfaz homogénea para mostrar la retroalimentación que se va produciendo a lo largo del proceso de integración continua, independientemente de las herramientas que se estén utilizando subyacentemente.

El problema es que las herramientas de CI actuales no están preparadas de forma nativa para trabajar directamente con ningún generador de artefactos dirigido por modelos. Se podrían utilizar generadores que funcionaran por línea de comandos, dado que la mayoría de herramientas de CI permiten ejecutar aplicaciones por línea de comandos, pero no se podría interpretar la salida del generador de forma homogénea respecto al resto de actividades realizadas por la herramienta de CI y, por tanto, no se podría obtener retroalimentación de un modo efectivo [Néron et al., 2009] sobre las generaciones de artefactos realizadas. Para conseguirlo, habría que adaptar el generador de artefactos dirigido por modelos y la herramienta de CI.

El beneficio de solventar el problema de cara a cumplir el objetivo de este trabajo sería que se podría obtener retroalimentación, de forma efectiva y homogénea, de la ejecución del generador de artefactos. De ese modo, se podría diferenciar si la operación ha sido exitosa, qué artefactos se han generado y, en definitiva, cualquier información de interés que pueda ofrecer el generador. Se puede mostrar información mediante interfaces gráficas y amigables a los usuarios o se puede, por ejemplo, avisar de incidencias al responsable mediante correo electrónico, etc.

1.4.4. Generación incremental de artefactos

En un nivel de abstracción inferior a los generadores incrementales MDE están los compiladores incrementales, como por ejemplo el compilador incremental de Java que viene incorporado en la plataforma Eclipse[3]. Dicho compilador, proporciona los siguientes beneficios que podrían ser extrapolados a la ingeniería dirigida por modelos:

- Se evita tener que reconstruir aplicaciones enteras con cada cambio. Cuando la aplicación es grande, se ahorra mucho tiempo.

[3]http://www.eclipse.org/

- Se logran compilaciones casi instantáneas cuando los cambios en el código son pequeños.

- Se reduce la granularidad de las unidades de compilación, al mismo tiempo que se mantiene la semántica del lenguaje.

- Se consigue crear un entorno de desarrollo mucho más interactivo, ya que el desarrollador puede percatarse de errores de forma casi instantánea.

El problema es que cada vez que los actuales generadores de artefactos dirigidos por modelos generan artefactos, los vuelven a regenerar todos (salvo de forma parcial en casos limitados o con herramientas específicas como OptimalJ [Compuware, 2005]). Habría que lograr que los generadores de artefactos dirigidos por modelos trabajaran de manera incremental, generando únicamente lo mínimo imprescindible.

El beneficio de solventar el problema de cara a cumplir el objetivo de este trabajo sería que se reduciría la cantidad de artefactos que se regeneran tras modificaciones en los modelos, ya que a mayor cantidad de artefactos regenerados, mayor será la probabilidad de que haya que recompilar código, rehacer pruebas, analizar las salidas, etc. En definitiva, se disminuiría el impacto sobre los sistemas ya generados o desplegados.

1.4.5. Construcción de un prototipo MDCI

Para permitir realizar la práctica de la integración continua en los desarrollos dirigidos por modelos MDCI (Model-Driven Continuous Integration o Integración Continua Dirigida por Modelos)[4], habría que, una vez solventados los principales problemas encontrados, integrar todas las soluciones parciales en una única herramienta y comprobar la utilidad del trabajo planteado. De ahí, la necesidad de construir un prototipo.

1.4.6. Especificación de ámbitos de aplicación para MDCI

Con la herramienta prototípica desarrollada, habría que tener disponibles diferentes ámbitos de aplicación en los que utilizarla. Esta actividad es un paso previo imprescindible para la realización de una evaluación cuantitativa. Los ámbitos de aplicación podrían ser verticales, horizontales o transversales.

[4]MDCI es un término genérico utilizado en este trabajo para englobar todo lo relacionado con la unión entre la integración continua y la ingeniería dirigida por modelos

1.4.7. Evaluación de MDCI

Con el prototipo desarrollado y los ámbitos de aplicación determinados, el siguiente paso sería la evaluación de MDCI (Model-Driven Continuous Integration) mediante casos prácticos en los que se estudien diferentes ámbitos de aplicación conjuntamente con el prototipo. El objetivo de la evaluación sería obtener valores para determinar numéricamente las ventajas del empleo de la propuesta de este trabajo sobre otras aproximaciones de desarrollo.

1.4.8. Definición de un proceso MDCI

Esta actividad es el objetivo principal y último de este trabajo (*llevar a cabo un proceso en el que se aplique la práctica de la integración continua en un desarrollo de software dirigido por modelos de forma eficiente.*), para ello, a partir de la información obtenida en todas las actividades previas, se extraerían las diferentes actividades de las que consta dicho proceso, mediante la respuesta a una serie de preguntas, útiles para definir cualquier proceso:

- ¿Quién?¿Cuándo?¿Dónde?
- ¿Qué?¿Por qué?
- ¿Entradas?¿Salidas?
- ¿Métodos?
- ¿Mediciones?

Tras exponer brevemente las principales actividades de esta Tesis Doctoral, en el siguiente capítulo se explicará la metodología de trabajo y el desarrollo de la investigación realizada.

CAPÍTULO 2

Metodología y desarrollo de la investigación

En este capítulo se presentará la metodología utilizada para el desarrollo de esta Tesis Doctoral. A continuación se muestra el desarrollo temporal de la investigación, exponiendo los principales hitos en orden cronológico. Por último, se especifica la organización de este documento, estructurado tanto en bloques como en capítulos para ofrecer una mejor visión del contenido al lector.

⁎ ⁎ ⁎ ⁎

2.1. Metodología de trabajo

Este trabajo de investigación ha sido desarrollado en varias etapas, utilizando un enfoque iterativo e incremental. En la Fig. 2.1 se muestra el marco metodológico de la investigación con referencias temporales.

El punto inicial de la investigación es el proyecto fin de carrera de la titulación de *Ingeniería en Informática*, realizado en la Escuela Politécnica de Ingeniería (Campus de Gijón), Universidad de Oviedo. A partir de los conocimientos adquiridos durante el curso académico 2006/2007 se presentó un prototipo de generador que, a partir de un modelo software, genera sitios Web para la gestión y el alquiler de casas rurales. Las experiencias de dicho proyecto sirvieron para realizar con éxito un proyecto de investigación denominado *Desarrollo de software para la realización de trazabilidad en la industria de los alimentos*, en el que, también mediante tecnologías MDE, se realizó un generador de aplicaciones de trazabilidad alimentaria [García-Díaz et al., 2008, García-Díaz et al., 2009a, García-Díaz et al., 2011], cuyas principales carencias se tomaron como motivación para la presente Tesis Doctoral.

Paralelamente a la realización de dicho proyecto de investigación, se adquirieron nuevos conocimientos en un ámbito meramente investigador debido a la realización del programa de doctorado *Sistemas y servicios informáticos para Internet*, del Departamento de Informática de la Universidad de Oviedo. Así, la fase de docencia sirvió como introducción a la investigación y a la publicación de artículos científicos.

Con una línea de investigación marcada se comenzó el trabajo de investigación, que consistió en profundizar en los problemas detectados en [García-Díaz et al., 2008] mediante la realización de un profundo estudio del estado del arte y el planteamiento de una línea a seguir para cumplir con los objetivos dados, siendo dicho trabajo una base sólida para la realización de la investigación aquí presentada. A partir del estado del arte realizado y de las conclusiones obtenidas, se inició el desarrollo de esta Tesis Doctoral, estableciendo de forma precisa el problema a resolver y los objetivos e hipótesis de partida.

Para llevar a cabo la investigación se ha optado por un enfoque iterativo e incremental. Así, en cada iteración, se desarrolló parcialmente el prototipo y el proceso finalmente expuesto. Se han realizado varias publicaciones en congresos internacionales y en revistas de reconocido prestigio con el objetivo de obtener una valiosa retroalimentación por parte de la comunidad científica, que ha servido para ayudar a desarrollar este trabajo en la línea adecuada.

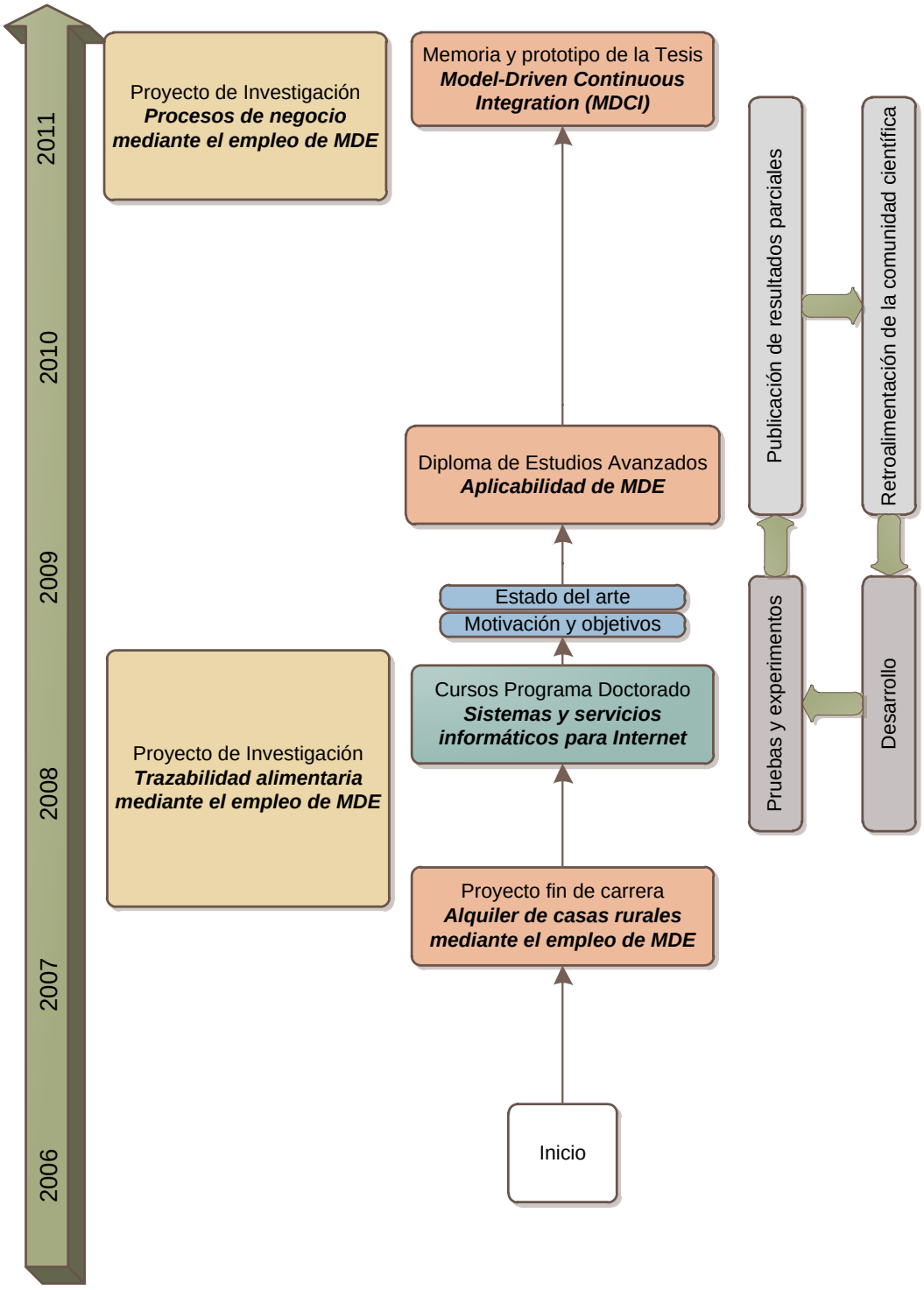


Figura 2.1: Metodología utilizada en la investigación

2.1.1. Ciclo de retroalimentación

El enfoque iterativo e incremental se basa en la iteración deductiva-inductiva, entendiéndose como un proceso de retroalimentación. Dicho proceso, conocido desde Aristóteles, es parte de nuestra experiencia diaria. Así, una idea inicial (o hipótesis) conduce, mediante un proceso de deducción, a ciertas consecuencias que se pueden comparar con los datos disponibles. Cuando las consecuencias y los datos no concuerdan, se puede llegar, por un proceso de inducción, a la modificación de la hipótesis, iniciándose un nuevo ciclo de iteración. En la Fig. 2.2 se reflejan estas ideas. Cabe destacar que la adquisición de datos durante esta Tesis viene fundamentalmente reflejada en las referencias finales del documento y en los enlaces Web expuestos en los pies de página.

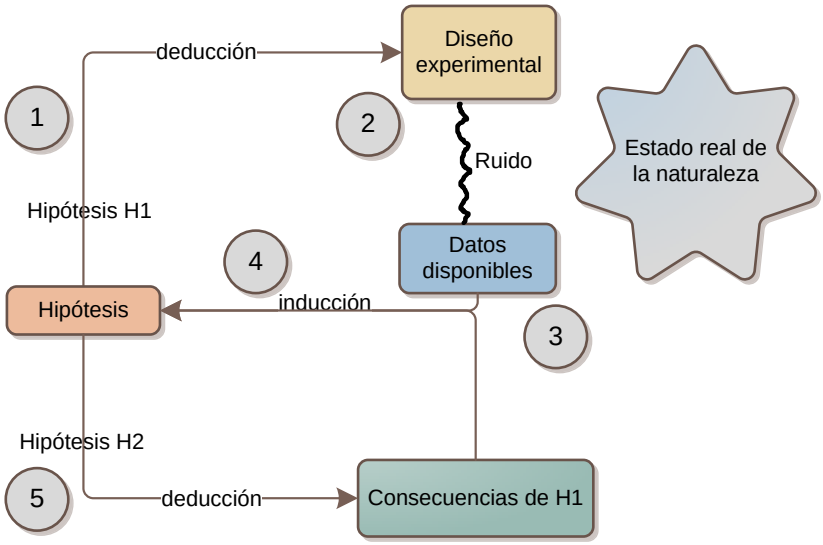


Figura 2.2: Proceso iterativo de aprendizaje

2.2. Desarrollo temporal de la investigación

A continuación, se presentan algunos de los hitos más significativos que han sido llevados a cabo durante la realización de este trabajo.

- **Junio de 2006**. Comienzo de los estudios relacionados con MDE. Esta primera etapa se corresponde con el inicio del proyecto fin de carrera realizado en la Escuela Politécnica de Ingeniería de la Universidad de Oviedo.
- **Junio de 2007**. Lectura del proyecto fin de carrera con la calificación de diez. Se realiza un generador de artefactos cuyo objetivo es la creación automática de

sitios Web para la gestión y alquiler de casas rurales. Para lograrlo se utilizan tecnologías MDE.

- **Julio de 2007**. Comienzo del proyecto de investigación *Desarrollo de software para la realización de trazabilidad en la industria de los alimentos*. El objetivo es crear una línea de productos software mediante el empleo de tecnologías MDE.

- **Octubre de 2007**. Se realiza la asignatura *Metodología de la Investigación en Informática*, perteneciente a la etapa de formación del programa de doctorado *Sistemas y servicios informáticos para Internet*, del Departamento de Informática de la Universidad de Oviedo. Se imparten conocimientos sobre los pasos necesarios para realizar una Tesis Doctoral.

- **Enero de 2008**. Se realiza la asignatura *Ingeniería de la Usabilidad*, que pertenece a la etapa de formación del programa de doctorado mencionado.

- **Febrero de 2008**. Se realiza la asignatura *Pruebas y validación de software y servicios*, perteneciente a la etapa de formación del programa de doctorado. Para la superación del curso se presenta una simulación de un artículo para un congreso titulado *Model Based Testing and UML*.

- **Marzo de 2008**. Se realiza la asignatura *Web Semántica*, perteneciente a la etapa de formación del programa de doctorado. Para la superación del curso se presenta una simulación de un artículo para un congreso titulado *Ontologías y modelos de desarrollo de software*.

- **Abril de 2008**. Se realiza la asignatura *Sistemas adaptables, reflectivos, y separación de aspectos con AspectJ*, perteneciente a la etapa de formación del programa de doctorado.

- **Junio de 2008**. Se realiza la asignatura *Lenguajes dinámicos para el desarrollo ágil de aplicaciones*, perteneciente a la etapa de formación del programa de doctorado.

- **Julio de 2008**. El artículo *Intelligent traceability system of Cabrales cheese using MDA TALISMAN* [García-Díaz et al., 2008] se presenta en The International Conference on Artificial Intelligence. El objetivo es mostrar los progresos realizados en el proyecto de investigación *Desarrollo de software para la realización de trazabilidad en la industria de los alimentos*. El resultado es un generador de sistemas de trazabilidad alimentaria desarrollado mediante

el uso de tecnologías MDE. Los sistemas desarrollados se utilizan actualmente en la industria alimentaria.

- **Julio de 2008**. El artículo *A review of Intelligent Software Development Tools* [Palacios-González et al., 2008a] se presenta en The International Conference on Artificial Intelligence. El resultado de este trabajo es una clasificación de las herramientas para trabajar con tecnologías MDE, muy beneficioso para tener constancia del abanico de posibilidades existentes.

- **Julio de 2008**. El artículo *Design of intelligent business applications based on BPM and MDE* [Fernández-Fernández et al., 2008] se presenta en The International Conference on Artificial Intelligence. El trabajo consiste en el uso de una arquitectura basada en BPM (Business Process Modeling) para que resulte más sencillo el uso de la herramienta de modelado presentada en [García-Díaz et al., 2008]. Para ello se presenta SBPMN (Simple Business Process Modeling Notation), una simplificación del estándar BPMN (Business Process Modeling Notation).

- **Septiembre de 2008**. El artículo *General purpose MDA Tools* [Palacios-González et al., 2008b] es publicado en la revista International Journal of Artificial Intelligence and Interactive Multimedia. La idea es profundizar más en las herramientas MDE de propósito general presentadas en [Palacios-González et al., 2008a].

- **Junio de 2009**. El artículo *TALISMAN MDE Framework. An Architecture for Intelligent Model-Driven Engineering* [García-Díaz et al., 2009c] se presenta en The International Work-Conference on Artificial Neural Networks. En este trabajo se ofrece una propuesta con la que se podría combinar en un único entorno de desarrollo las ventajas de diferentes iniciativas MDE.

- **Junio de 2009**. El artículo *Towards Meta-model Interoperability of Models and Transformation Models* [Tolosa et al., 2009] se presenta en The International Work-Conference on Artificial Neural Networks. Este trabajo es una primera toma de contacto para lograr la interoperabilidad entre diferentes metamodelos.

- **Junio de 2009**. Presentación del trabajo de investigación *Aplicabilidad de la Ingeniería Dirigida por Modelos* [García-Díaz, 2009] con la calificación de Sobresaliente.

- **Junio de 2009**. Obtención del Certificado-Diploma de Estudios Avanzados tras la realización del periodo de docencia y de investigación del Programa de doctorado *Sistemas y servicios informáticos para Internet*, del Departamento de Informática de la Universidad de Oviedo.

- **Julio de 2009**. El artículo *Version control for an intelligent traceability system focusing on underlying source code changes control* [García-Díaz et al., 2009b] se presenta en The International Conference on Artificial Intelligence. El objetivo es tratar la problemática del control de versiones cuando se trabaja con el paradigma MDE, importante porque el versionado es un elemento clave en la integración continua.

- **Julio de 2009**. El artículo *Intelligent Generation of Web Applications for eGovernment* [Palacios-González et al., 2009] se presenta en The International Conference on Artificial Intelligence. Se muestra cómo se puede aplicar MDE en el dominio de las aplicaciones relacionadas con trámites públicos en Internet. Este trabajo trata un posible ámbito de aplicación del trabajo realizado en esta Tesis (véase sección 15.1.1).

- **Diciembre de 2009**. El artículo *Developing a Business Application with BPM and MDE* [Fernández-Fernández et al., 2009] es publicado en la revista International Journal of Artificial Intelligence and Interactive Multimedia. En este trabajo se aborda cómo se puede utilizar un lenguaje de modelado de procesos para generar artefactos de menor nivel de abstracción. Es una ampliación del trabajo presentado previamente en [Fernández-Fernández et al., 2008].

- **Diciembre de 2009**. El artículo *Automated code generation support for BI with MDA TALISMAN* [García-Díaz et al., 2009a] es publicado en la revista International Journal of Artificial Intelligence and Interactive Multimedia. Este trabajo detalla cómo funciona el primer generador de código realizado para crear sistemas de trazabilidad alimentaria. Es una ampliación del trabajo presentado previamente en [García-Díaz et al., 2008].

- **Enero de 2010**. El artículo *SBPMN - An Easier Business Process Modeling Notation for Business Users* [Fernández-Fernández et al., 2010] es publicado en la revista Computer Standards & Interfaces, con un último **factor de impacto en el índice Journal Citation Reports de 1,373**. En este trabajo se ofrece una alternativa, más fácil de utilizar, a la notación estándar de modelado de procesos. Las base de la publicación puede verse en [Fernández-Fernández et al., 2008, Fernández-Fernández et al., 2009].

- **Julio de 2010**. El artículo *TALISMAN MDE: Mixing MDE principles* [García-Díaz et al., 2010a] es publicado en la revista Journal of Systems and Software, con un último **factor de impacto en el índice Journal Citation Reports de 1,340**. Con esta publicación se da una solución a la problemática de la elección de una iniciativa MDE, una de las dificultades abordadas en el desarrollo de esta Tesis.

- **Septiembre de 2010**. El artículo *Towards an adaptive integration trigger* [García-Díaz et al., 2010b] se presenta en el International Symposium on Distributed Computing and Artificial Intelligence. El objetivo es dejar patente la necesidad de mejora de los disparadores utilizados para iniciar el proceso de integración continua.

- **Septiembre de 2010**. El artículo *Bridging the gap between Model-Driven Engineering and Continuous Integration* [García-Díaz et al., 2011a] es enviado para su posible publicación a la revista Science of Computer Programming, con un último **factor de impacto en el índice Journal Citation Reports de 1.459**. El objetivo de este trabajo es mostrar de forma general el trabajo realizado para unir MDE y CI en esta Tesis.

- **Noviembre de 2010**. El artículo *Towards the Systematic Measurement of ATL Transformation Models* [Tolosa et al., 2010] es publicado en la revista Software Practice and Experience, con un último **factor de impacto en el índice Journal Citation Reports de 0,667**. El trabajo profundiza en la medición de la calidad de diferentes transformaciones entre modelos y metamodelos, muy útil para ayudar a realizar las transformaciones propuestas en [García-Díaz et al., 2010a].

- **Diciembre de 2010**. Comienzo del proyecto de investigación *Generación automática de software de procesos de negocio basada en desarrollo dirigido por modelos*. El objetivo es crear aplicaciones a partir del proceso de negocio definido por un experto en un dominio de conocimiento determinado.

- **Enero de 2011**. El capítulo *Traceability Systems for Sustainable Development in Rural Areas* [García-Díaz et al., 2011] es publicado en el libro Regional Innovation Systems and Sustainable Development. Emerging Technologies (Editorial IGI-Global). El objetivo del trabajo es presentar cómo MDE puede ayudar a los productores de alimentos en la tarea de gestión y control de la trazabilidad alimentaria de sus fábricas. Complementa a [García-Díaz et al., 2008, García-Díaz et al., 2009a].

- **Enero de 2011**. El artículo *Metamodel, Domain Specific Language (DSL) tool, and transformations for Learning Management Systems (LMS)* [Montenegro-Marín et al., 2011] es enviado para su posible publicación a la revista Journal of Systems and Software, con un último **factor de impacto en el índice Journal Citation Reports de 1,340**. Se muestra cómo se puede aplicar MDE en el dominio de las aplicaciones de aprendizaje electrónico. Este trabajo trata un posible ámbito de aplicación del trabajo realizado en esta Tesis (véase sección 15.1.2).

- **Febrero de 2011**. El artículo *BMPN MUSIM: Notación BPMN MUy SIMplificada* [Martínez et al., 2011a] es aceptado para su publicación en la Conferencia Ibérica de Sistemas y Tecnologías de Información. El objetivo de este trabajo es seguir la línea de [García-Díaz et al., 2008, Fernández-Fernández et al., 2008, Fernández-Fernández et al., 2010] para seguir profundizando en la definición de procesos de negocio de la forma más sencilla posible. Este trabajo trata un posible ámbito de aplicación del trabajo realizado en esta Tesis (véase sección 15.2.1).

- **Febrero de 2011**. El artículo *Isastur Modeler: A tool for BPMN MUSIM* [Martínez et al., 2011b] es aceptado para su publicación en la Conferencia Ibérica de Sistemas y Tecnologías de Información. El objetivo es mostrar los progresos realizados en el proyecto de investigación *Generación automática de software de procesos de negocio basada en desarrollo dirigido por modelos*. El resultado es una herramienta que permite definir fácilmente procesos de negocio. Este trabajo es un complemento del trabajo presentado en [Martínez et al., 2011a].

- **Febrero de 2011**. El artículo *MCTest: Towards an improvement of match algorithms for models* [García-Díaz et al., 2011b] es enviado para su posible publicación a la revista IET Software, con un último **factor de impacto en el índice Journal Citation Reports de 0.62**. El trabajo presenta la herramienta MCTest, un sistema que probar y estudiar algoritmos de comparación de modelos, interesante porque los algoritmos actuales pueden ser mejorados y, además, no existe un algoritmo óptimo para todos los posibles problemas, teniendo que ser adaptados manualmente para cada caso particular.

- **Abril de 2011**. El artículo *Incremental Generation in Model-Driven Engineering* [García-Díaz et al., 2011c] es enviado para su posible publicación a la revista Journal of Software Maintenance and Evolution Research and Practice,

con un último **factor de impacto en el índice Journal Citation Reports de 1,143**. El trabajo explica el modo en el que se ha realizado la generación incremental de artefactos utilizando modelos como los elementos iniciales en el desarrollo de software, muy importante para lograr una óptima integración continua.

2.3. Organización de este documento

Esta Tesis está estructurada en *bloques* y *capítulos*. A continuación se describe la organización principal del presente documento con el objetivo de facilitar su lectura.

- Bloque I (*Planteamiento del Problema*). El capítulo 1 se centra en asentar la motivación, justificación y objetivos de la Tesis. Por su parte, el capítulo 2 refleja la metodología general utilizada y el desarrollo temporal, incluyendo también la presente organización estructural.
- Bloque II (*Marco Teórico*). Los capítulos 3 y 4 explican, respectivamente, los conceptos básicos de MDE y de CI, necesarios para una mejor comprensión del resto del trabajo expuesto.
- Bloque III (*Problema: Elección de una iniciativa MDE*). Los capítulos 5 y 6 están dedicados a las principales iniciativas MDE existentes en la actualidad. El capítulo 7 propone una solución ante el problema de la dificultad de elección de una iniciativa MDE específica para afrontar desarrollos dirigidos por modelos.
- Bloque IV (*Problema: Detección de nuevas versiones en los VCSs*). El capítulo 8 explica la problemática de los sistemas de control de versiones en lo referente al trabajo con modelos. El capítulo 9 ofrece el trabajo realizado para tratar de salvar este inconveniente.
- Bloque V (*Problema: Integración de MDE con herramientas de CI*). El capítulo 10 aborda las diferentes técnicas y herramientas existentes para generar artefactos a partir de modelos. En el capítulo 11 se integra una herramienta de CI con una herramienta MDE.
- Bloque VI (*Problema: Generación incremental de artefactos*). El capítulo 12 se centra en la generación incremental de artefactos en la ingeniería dirigida por modelos y las ventajas que proporcionaría. Así, en el capítulo 13 se ofrece una alternativa para conseguirlo.

- Bloque VII (*Integración Continua Dirigida por Modelos*). Bloque centrado en MDCI (Model-Driven Continuous Integration). En el capítulo 14 se encuentra la herramienta prototípica MDCIP (Model-Driven Continuous Integration Prototype), realizada para dar solución a las diferentes problemáticas planteadas en esta Tesis. En el capítulo 15 se muestran diferentes ámbitos en los que podrían aplicarse las ideas de este trabajo. En el capítulo 16 se muestra una evaluación y un análisis de MDCI mediante los resultados obtenidos al utilizar el prototipo en algunos de los ámbitos de aplicación citados. Por último, en el capítulo 17 se muestra el proceso MDCI propuesto en función del prototipo desarrollado.

- Bloque VIII (*Conclusiones y futuro trabajo*). En los capítulos 18 y 19 están reflejadas las conclusiones finales y el posible trabajo futuro a realizar tras la finalización de esta Tesis Doctoral.

- Bloque IX (*Anexos*). En los anexos A, B y C se incluye diversa información que complementa al trabajo realizado: diferentes definiciones de procesos de fabricación, el software utilizado para el desarrollo de este trabajo y los metamodelos creados en las pruebas.

- Finalmente, tras los anexos, se incorporan las referencias bibliográficas, los acrónimos utilizados a lo largo de todo el trabajo y el índice alfabético de los conceptos más relevantes.

Bloque II

Marco teórico

Índice del bloque

CAPÍTULO 3

Ingeniería dirigida por modelos

La *crisis del software* es un concepto que se ha empezado a utilizar en 1968 en la primera conferencia organizada por la OTAN sobre desarrollo de software [Dijkstra, 1972]. Lo cierto es que, aunque se han propuesto nuevas metodologías cuyo objetivo es solventar los problemas que habitualmente tienen los desarrollos de software, aún no existe ningún método que haya permitido hacer estimaciones de manera fiable de tiempo y coste en el desarrollo de grandes sistemas software[1].

Ese es el principal motivo de la aparición de la aproximación para desarrollar software denominada MDE (Model-Driven Engineering o Ingeniería Dirigida por Modelos) [Kent, 2002].

En este capítulo se da una visión general de MDE, indicando el origen a través de las aplicaciones en el ámbito empresarial, su ámbito de aplicación, los principales conceptos subyacentes y las iniciativas existentes más relevantes de la actualidad para llevar a cabo desarrollos siguiendo sus principios.

* * * *

[1] El Standish Group [StandishGroup, 2008] ha estimado a través de su CHAOS Report que en el año 2004 un 70 % de los proyectos informáticos NO fueron considerados un éxito

3.1. Las aplicaciones en el ámbito empresarial

Cuando se desarrollan aplicaciones en el ámbito empresarial, es muy importante pensar en los requisitos necesarios y en el tipo de negocio para el que se destinará la aplicación, ya que hay aplicaciones en las que prima la disponibilidad, en otras la seguridad y, por ejemplo, en otras la escalabilidad. En cualquier caso, en prácticamente todas las aplicaciones de cierta envergadura hay aspectos comunes a tener en cuenta, los cuales son independientes de la aproximación o metodología de desarrollo empleada. Algunos de estos aspectos son:

- Comunicación con bases de datos, insertando, borrando y recuperando información.
- Autenticación y autorización de usuarios.
- Configuración de las aplicaciones desarrolladas.
- Utilización de algún sistema de registro de eventos o *logging* para tener guardada información importante durante el funcionamiento del sistema.
- Gestión de errores para evitar cuelgues u otros tipos de resultados inesperados en las aplicaciones.
- Mecanismos de caché.

3.1.1. Componentes de un framework empresarial

Las aplicaciones empresariales tienen una arquitectura formada por componentes, que típicamente se separan en capas. Aunque no todas las aplicaciones tienen por qué tener todos los componentes, la Fig. 3.1 muestra los componentes de una aplicación típica. Todos estos elementos, al ser comunes a casi todas las aplicaciones, son los candidatos perfectos a formar parte de un framework empresarial [Lenz and Wienands, 2006].

Capa de datos

En la mayoría de las aplicaciones la capa de datos puede dividirse en tres subcapas:

- Capa de almacén de datos. Provee mecanismos para almacenar y gestionar los datos que maneja la aplicación. Aunque se podría utilizar cualquier soporte como por ejemplo documentos XML (Extensible Markup Language), archivos

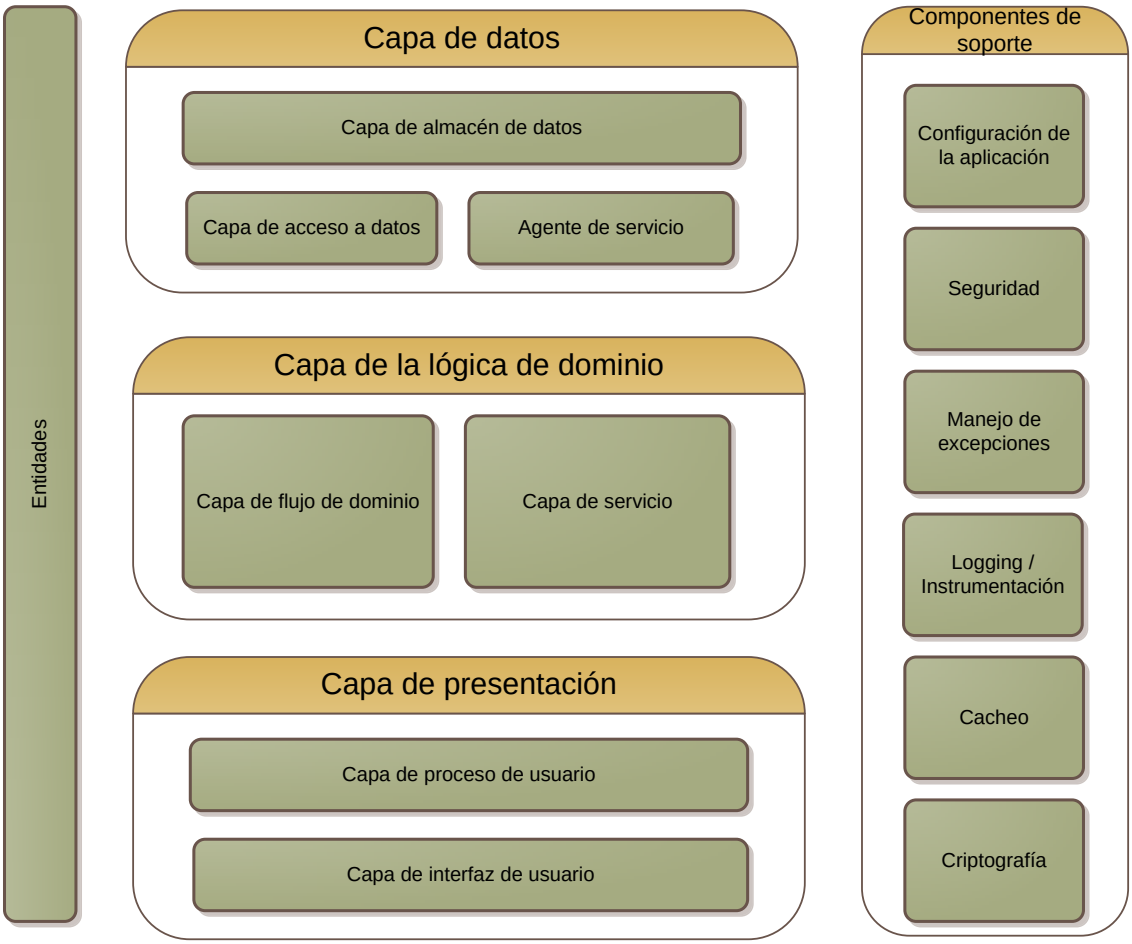


Figura 3.1: Componentes de una aplicación típica

INI (Windows Initialization file) o incluso hojas de cálculo Excel, las capas de almacén de datos más típicas son bases de datos relacionales como Microsoft SQL Server, Oracle o MySQL. Generalmente, una misma capa de almacén de datos puede ser compartida por varias aplicaciones simultáneamente.

- Capa de acceso a datos. Provee la lógica necesaria para trabajar con los datos almacenados por la aplicación a través de la capa de almacén de datos, evitando que el resto de la aplicación tenga que preocuparse de cómo acceder a los datos. La separación con el resto de capas debe ser, por motivos de mantenimiento, al menos lógica. Por motivos de escalabilidad y también de mantenimiento, es aconsejable que la separación sea también física.

- Agente de servicio. El agente de servicio (también llamado proxy) está a la altura de la capa de acceso a datos, pero en lugar de trabajar con la capa de

almacén de datos, trabaja con componentes externos como pueden ser servicios Web o componentes COM+ a través de DCOM (ambos componentes son ejemplos de la capa de servicio que se comentará a continuación).

Capa de la lógica de dominio

La capa de la lógica de dominio (también llamada capa de la lógica de negocio) es la capa que está entre la capa de presentación y la capa de datos. Por lo tanto, esta capa se encarga de procesar y realizar todas las validaciones necesarias para realizar una acción. Al igual que ocurre con la capa de acceso de datos, debería tener al menos una separación lógica con el resto de capas de la aplicación. La capa de la lógica de dominio tiene habitualmente dos componentes o subcapas:

- Capa del flujo de dominio. Esta capa suele utilizarse cuando las aplicaciones realizan acciones de una cierta complejidad y sirve para orquestar los flujos que siguen las actividades que pueda tener una determinada acción.

- Capa de servicio. Provee un punto de acceso para aplicaciones externas que puedan y quieran utilizar diferentes funcionalidades o servicios que ofrece la aplicación a modo de caja negra. Generalmente un agente de servicio de una aplicación externa es quien se pone en contacto con la capa de servicio.

Capa de presentación

La capa de presentación generalmente se divide en dos subcapas:

- Capa de interfaz de usuario. Provee una interfaz para que los usuarios se puedan comunicar con la aplicación. Generalmente esta capa se comunica con la capa de la lógica del dominio aunque también es válido que se comunique con la capa de datos para realizar acciones simples y directas como rellenar *comboboxes*. También se acepta que esta capa tenga algo de lógica del dominio sencilla, como podría ser la comprobación de si un número de teléfono es correcto.

- Capa de proceso de usuario. Esta capa sirve para manejar los flujos de trabajo de las interfaces con las que el usuario trabaja y permite dejar la capa de interfaz de usuario muy ligera, posibilitando de ese modo migrar una interfaz a otra tecnología de forma muy directa. Por ejemplo, el patrón de diseño MVC (Modelo-Vista-Controlador) [Krasner and Pope, 1988] hace uso de dicha separación.

Entidades de negocio

Las entidades de negocio representan entidades en un dominio concreto y son compartidas por todas las capas de una aplicación debido a que son el medio para trabajar con los diferentes datos que maneje el sistema (ejemplos de entidades en una aplicación destinada a la gestión de notas de alumnos podrían ser *Alumno, Profesor, Asignatura, Examen, etc.*). Además, las entidades deben poseer atributos de forma que puedan realizarse acciones con ellas (una entidad *Asignatura* podría tener como atributos *NombreAsignatura, ProfesorResponsable, NúmeroAlumnosMatriculados, etc.*).

Componentes de soporte

Hay varios componentes que se pueden considerar de soporte para las aplicaciones empresariales y que, al igual que las entidades de negocio, están típicamente compartidos por todas las capas de las aplicaciones:

- Configuración de la aplicación. Todas las aplicaciones de un cierto tamaño necesitan metadatos para definir aspectos clave de las mismas. Por ejemplo, las cadenas de conexión a las bases de datos o la ruta en la que se quiere guardar los archivos de *log*. Para almacenar esta información se pueden utilizar simples archivos de texto, bases de datos o por ejemplo los archivos de configuración en formato XML que provee la plataforma .NET para facilitar y uniformar la configuración de las aplicaciones.

- Seguridad. La seguridad de los datos es otro de los aspectos clave de las aplicaciones modernas, debiéndose tener en cuenta aspectos como si se accederá a la aplicación a través de una intranet, extranet o Internet, o qué parte de la aplicación podrá utilizar cada uno de los posibles usuarios de la misma. Hay dos conceptos clave a la hora de gestionar la seguridad: autenticación y autorización. La autenticación es la determinación de la identidad de las personas o máquinas que acceden a la aplicación. La autorización es posterior a la autenticación, ya que consiste en determinar qué acciones y qué funcionalidades de la aplicación pueden realizar las identidades autenticadas, pudiéndose realizar la gestión de la autorización por identidades individuales o por grupos de identidades (diferentes roles o perfiles).

- Gestión de errores. Durante el desarrollo de las aplicaciones, casi siempre se producen errores que en tiempo de ejecución ocasionan problemas a los usuarios finales. Por este motivo, se considera necesario un correcto manejo de las

excepciones. Una vez producida una excepción se puede ignorar, mostrar un mensaje al usuario, almacenar en un archivo de texto, almacenar en una base de datos, o se puede, por ejemplo, enviar un correo electrónico al encargado de mantenimiento.

- Registro de eventos. Crear un registro de eventos o *log* es otro aspecto importante que tiene múltiples usos como ayudar a la depuración, calcular rendimientos, crear auditorias o almacenar las excepciones ocurridas. Los *logs* se pueden almacenar, por ejemplo, en ficheros de texto plano, en archivos XML, en una base de datos o en el registro de eventos de Windows, en el caso de trabajar bajo dicha plataforma.

- Caché. Duplicar datos a partir de otros datos costosos de acceder, normalmente en tiempo, respecto a la copia duplicada, proporciona a las aplicaciones beneficios tales como mejoras de rendimiento, aumento de escalabilidad y reducción de la carga a la que es sometida la capa de almacén de datos. Un ejemplo podría ser una lista de países que se muestra en un *combobox*, que podría ser recuperada de la base de datos únicamente la primera vez que se solicita. Las siguientes ocasiones, en lugar de volver a acceder a la base de datos, se accedería a través de la caché. Como la memoria de los sistemas es finita, para no sobrecargar ni uno ni otro elemento, habrá que buscar una solución de compromiso entre la memoria del sistema utilizada por la caché y la carga a la que la base de datos es sometida cuando no se utiliza caché.

- Criptografía. Muchas aplicaciones necesitan encriptar y desencriptar datos, lo que justifica un componente destinado a gestionar la criptografía. Es muy importante cuando se maneja información sensible de usuarios, empresas, clientes, etc.

- Despliegue. Otro aspecto importante es el despliegue de las aplicaciones una vez finalizadas, ya que dependiendo del tipo de aplicación y de su entorno de ejecución, el despliegue se hará de una u otra forma. Hay varios sistemas que se utilizan en la actualidad para facilitar el despliegue como por ejemplo ClickOnce [Hashimi and Hashimi, 2006] o algunos entornos de construcción, muy utilizados, como se verá en los próximos capítulos de esta Tesis.

3.1.2. Reutilización de componentes en un framework empresarial

Un factor clave para el éxito de un desarrollo de software es la necesidad de evitar tener que *reinventar la rueda* una y otra vez. Así, los aspectos y componentes nombrados pueden reutilizarse de múltiples formas, destacando típicamente:

- Copiar y pegar. Es la forma más antigua y fácil de reutilización de código, pero tiene gravísimos inconvenientes de mantenimiento, ya que si bien parece fácil copiar y pegar algún pequeño fragmento de código, siempre que haya que modificar uno de los fragmentos habrá que buscar los fragmentos copiados y modificarlos del mismo modo. Si además, la aplicación es una aplicación que crece, se puede convertir en una aplicación muy difícil de mantener incluso para las mismas personas que la han desarrollado.

- Generadores de código [Dollard, 2004]. Los generadores de código sirven para generar código de forma rápida y segura pero en ocasiones tienen inconvenientes. Por ejemplo, si se ha generado código y el desarrollador lo modifica, ya no se podrá volver a regenerar el código puesto que no incluiría las nuevas modificaciones. Además, si se desea añadir o modificar las funcionalidades ofrecidas por el generador de código, habrá que hacer modificaciones, incluso en el propio generador.

- Frameworks [Chen, 2004]. Los frameworks (marcos de desarrollo) son una colección de componentes que pueden ser reutilizados en diferentes aplicaciones para proporcionar una funcionalidad deseada. Estos componentes, generalmente se pueden modificar o extender para ofrecer funcionalidades específicas que luego pueden ser empleadas en otras aplicaciones. Los frameworks son una forma fácil, a través de interfaces, de ofrecer potencia con la máxima mantenibilidad. Hay dos tipos de frameworks que se utilizan en conjunto, ofreciendo todo lo necesario para desarrollar software. Por un lado, los frameworks de entorno (también llamados frameworks de desarrollo o de ejecución), que ofrecen APIs (Application Programming Interface) para proporcionar características como gestión de memoria o creación de ventanas, con el objetivo de facilitar el desarrollo (p.e., .NET Framework). Por otro lado, los frameworks empresariales (también llamados frameworks de aplicación), que proveen una forma común para realizar ciertas tareas repetitivas, que en realidad podrían hacerse de múltiples formas diferentes. Así, con los frameworks empresariales se trabaja siguiendo las mejores prácticas disponibles, reduciendo la cantidad

de código, evitando errores y facilitando el mantenimiento de las aplicaciones (p.e., ASP.NET MVC).

Aunque los generadores, los frameworks y otras opciones existentes son formas muy potentes para reutilizar código, no aumentan suficientemente el nivel de abstracción como para ser mecanismos que permitan automatizar el desarrollo de software de forma eficiente. Por ese motivo, no son capaces de evitar la problemática tradicional de los desarrollos de software.

3.2. La problemática tradicional en el desarrollo de software

Las aplicaciones empresariales siempre han sido muy propensas a sufrir problemas durante su desarrollo [Dijkstra, 1972]. A continuación, se muestra una lista con los problemas más comunes, lista que no ha cambiado con el paso de los años, y que es la principal motivación de la aparición de MDE:

- Por lo general, hay baja calidad en el software que se desarrolla.
- El software no cumple con las especificaciones y su funcionalidad no es la adecuada.
- Los proyectos no se ajustan a la planificación prevista.
- Los proyectos no se ajustan al presupuesto.
- El mantenimiento se vuelve costoso a medida que el proyecto crece en tamaño.

Las posibles causas de los anteriores problemas podrían ser las siguientes [Greenfield et al., 2004]:

- En muchas ocasiones se aísla el desarrollo, esto es, se desarrolla el software sin tener en cuenta otros desarrollos que podrían haber sido utilizados como base de conocimiento.
- El software monolítico, con componentes muy interconectados, se trata de una práctica muy desaconsejable.
- Los lenguajes con bajo nivel de abstracción son mucho más flexibles que los demás lenguajes pero hacen que el desarrollador tenga que preocuparse de muchos aspectos, como por ejemplo la liberación de memoria, reduciendo drásticamente la productividad de desarrollo.

- Los procesos de desarrollo de software no están tan maduros como pueden estar los procesos de otros sectores como el del automóvil o la construcción. Ello es debido a la relativa juventud de la informática.

- La creciente demanda de software en la sociedad, que provoca que en algunas ocasiones se intente desarrollar software excesivamente rápido.

3.2.1. Necesidad de automatización en los desarrollos

La manera de evitar los anteriores y otros muchos problemas es automatizando el desarrollo de software lo máximo posible. Este cambio no se ha producido de forma plena en el sector del software informático; aunque se avanza en el camino adecuado con la aparición de patrones de diseño [Gamma et al., 1995], especificaciones[2], estándares[3], frameworks y lenguajes de programación con mayor nivel de abstracción, que permiten entre otras cosas:

- Automatizar parcialmente el proceso de desarrollo.

- Encontrar las mejores formas para solucionar los problemas a los que se enfrentan los desarrolladores habitualmente.

- Buscar formas homogéneas de realizar una tarea con el fin de mejorar el mantenimiento y la interoperabilidad de las aplicaciones.

Sin embargo, a simple vista, estos problemas pueden no parecer tan complicados. Un proyecto software siempre se empieza cuando alguien tiene un problema y quiere darle una solución. La idea es sencilla, hay que recoger lo que necesita el cliente e implementarlo. ¿Qué ha ocurrido tradicionalmente? El cliente indica sus deseos y el encargado de recoger las especificaciones para después implementarlas no las traslada a un lenguaje formal, pero aún así los sistemas se implementan y se prueban. ¿Qué sería lo más conveniente? Se debería utilizar un lenguaje formal para recoger correctamente las especificaciones del cliente como fase inicial de una potencial automatización en el desarrollo. *Para realizar la especificación de una manera formal se utilizarán los modelos software.*

[2]Especificar es fijar o determinar algo de modo preciso

[3]Un estándar generalmente procede de una especificación pero sólo puede ser producido por cuerpos internaciones reconocidos por uno o varios gobiernos nacionales, salvo en el caso que sea utilizado por un número tan importante de personas que se considere un *estándar de facto*

3.2.2. Nivel de abstracción en el desarrollo

Por otra parte, la historia del desarrollo de software está inevitablemente ligada a las diferentes generaciones existentes de los lenguajes de programación:

- Lenguajes de primera generación (lenguaje máquina). Los primeros desarrolladores de software tenían que crear sus aplicaciones introduciendo directamente los bits, ceros y unos lógicos, de su software, con la consiguiente complejidad que esto acarreaba.

- Lenguajes de segunda generación (lenguaje ensamblador). Los ensambladores elevaron el nivel de abstracción y permitieron dar un gran salto cuantitativo y cualitativo en el desarrollo de software.

- Lenguajes de tercera generación (lenguajes procedimentales). Son mucho más sencillos de utilizar que los anteriores lenguajes porque tienen una mayor similitud con la forma de comunicación humana, aunque no son tan eficientes en términos de rapidez y utilización de memoria. Ejemplos son C, Pascal o Cobol.

- Lenguajes de cuarta generación (lenguajes orientados a objetos). Los lenguajes de cuarta generación utilizan por primera vez las clases y sus instancias, los objetos. Se emplean términos como encapsulación, polimorfismo y herencia. La reutilización de código y el parecido con el lenguaje humano hace que estos lenguajes aumenten la productividad de desarrollo. Ejemplos son C++, C# o Java.

- Lenguajes de quinta generación (lenguajes orientados a aspectos) [Elrad et al., 2001]. Muchos autores no están de acuerdo con esta categorización porque no existe un gran consenso sobre qué lenguajes pertenecen a la quinta generación. La idea de AOP (Aspect Oriented Programming o Programación Orientada a Aspectos) es permitir realizar una adecuada modularización de las aplicaciones y mejorar así la separación de conceptos dentro de la misma. El objetivo principal es la separación de funcionalidades dentro de la aplicación. Ejemplos son AspectJ (extensión de Java para Eclipse) o PHPAspect (extensión de PHP).

Si se observa el nivel de abstracción de los diferentes lenguajes, se podría hablar de varios niveles:

- Lenguajes de bajo nivel. Incluyen al lenguaje máquina y al lenguaje ensamblador. Trabajan directamente con el hardware por lo que están muy próximos al funcionamiento del ordenador.

- Lenguajes de medio nivel. Están entre los lenguajes de bajo nivel y los lenguajes de alto nivel. Por ejemplo el lenguaje C puede realizar operaciones de bajo nivel como trabajar con los registros del sistema pero también puede realizar otras operaciones de alto nivel.

- Lenguajes de alto nivel. Son lenguajes independientes del hardware del ordenador y que, por tanto, se pueden migrar de una máquina a otra fácilmente mediante el uso de traductores. Gracias a estos lenguajes no es necesario conocer los detalles internos del hardware de la máquina sobre la que está trabajando. Los lenguajes más utilizados hoy en día como C# o Java son lenguajes de alto nivel.

A medida que se aumenta el nivel de abstracción se aumenta la productividad, ya que además de utilizar términos más cercanos a la forma en la que se comunican los humanos, se pueden utilizar instrucciones más sofisticadas.

El último gran paso importante para aumentar la productividad y calidad del desarrollo de software, aumentando así el nivel de abstracción, es la aparición de MDE (Model-Driven Engineering o Ingeniería Dirigida por Modelos) [Kent, 2002].

MDE, también llamado MDD (Model-Driven Development o Desarrollo Dirigido por Modelos) o MDSD (Model-Driven Software Development), se considera un nuevo paradigma dentro del campo de la ingeniería del software. Se basa en la separación entre la funcionalidad del sistema a desarrollar y la implementación de dicho sistema para una plataforma concreta, por tanto, se busca separar claramente el análisis y el diseño de la implementación. Para conseguirlo se utilizan diferentes *modelos software*.

Según [Selic, 2008] existen dos tipos de complejidades en el desarrollo de software:

- *la complejidad esencial*, inevitable y propia del problema que se quiere solucionar.

- *la complejidad arbitraria*, debida a las herramientas y métodos utilizados durante el desarrollo.

MDE sirve para paliar la complejidad arbitraria, elevando el nivel de abstracción y evitando problemas léxicos, sintácticos y semánticos con los diferentes lenguajes de programación que existen y existirán en el futuro.

Un aspecto clave es que elevando el nivel de abstracción utilizando modelos, se pueden recoger las especificaciones de los clientes utilizando un lenguaje formal.

3.3. Modelos y diagramas para la construcción software

Según la RAE (Real Academia Española)[4], la palabra *modelo* tiene varios significados, entre los que pueden destacarse los siguientes:

1. *Arquetipo o punto de referencia para imitarlo o reproducirlo.*
2. *En las obras de ingenio y en las acciones morales, ejemplar que por su perfección se debe seguir e imitar.*
3. *Representación en pequeño de alguna cosa.*
4. *Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento.*

Por tanto, los modelos se han utilizado históricamente para representar y validar sistemas antes de realizar el esfuerzo superior que supone realizar o fabricar el sistema completo. Ejemplos de modelos podrían ser los planos de un edificio o maquetas del diseño un automóvil. En la Fig. 3.2 puede verse un ejemplo de un modelo.

Según [Selic, 2003], las características deseables de los modelos serían:

- Baratos. Es lógico pensar que la principal característica que deben poseer los modelos es que sean mucho más baratos que los sistemas a los que representan, tanto en términos económicos como en el tiempo que es necesario emplear para construirlos.
- Precisos. Los modelos deben representar correcta y precisamente a los sistemas reales, ya que de otra forma su empleo no tendría ninguna utilidad. De nada sirven los planos de un puente si el puente al que representan no se corresponde con el puente que se quiere construir.
- Comprensibles. De nada sirve un modelo si está expresado o representado de una forma confusa o difícil de comprender para quien deba hacer uso de él.

Refiriéndose al software, hay bastante confusión entre lo que es un modelo y un diagrama y por ello se utilizan en muchas ocasiones indistintamente ambos conceptos, cuando en realidad no son lo mismo.

[4] http://www.rae.es/

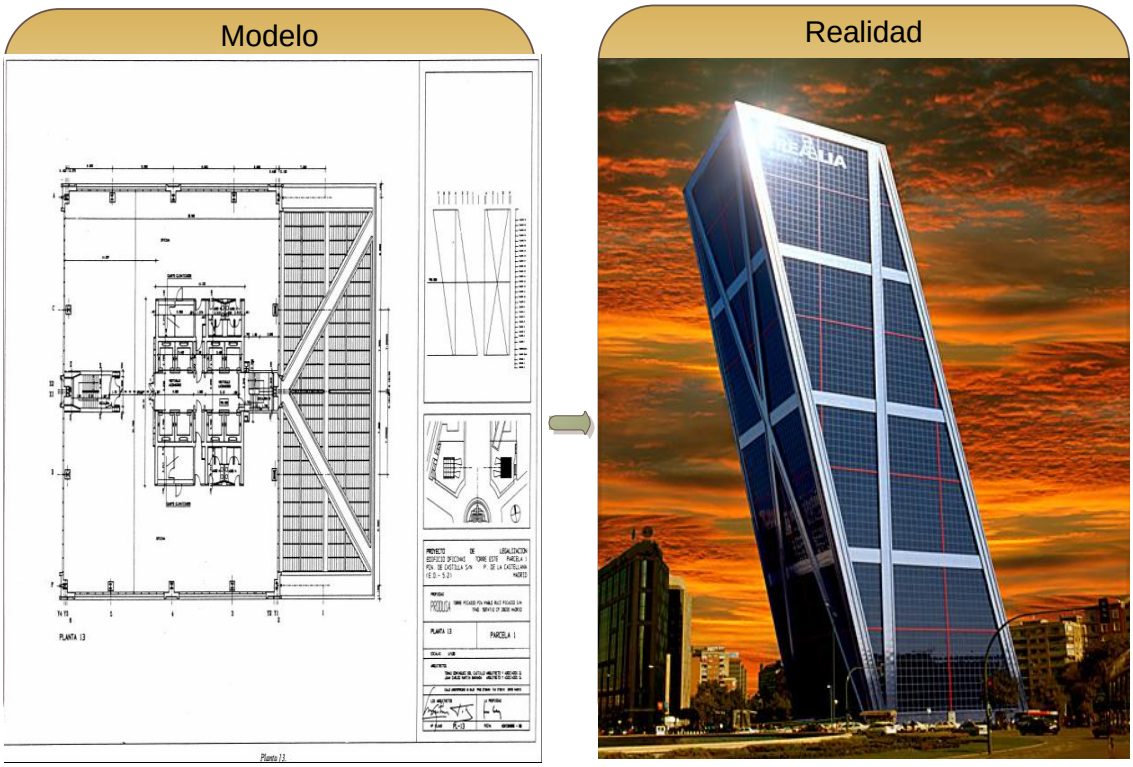


Figura 3.2: Modelo para la construcción de un edificio (ejemplo)

Un *modelo* es una abstracción de un sistema del mundo real que captura una vista (un sistema puede tener multitud de vistas). Por lo tanto, el modelo describe aquellos aspectos del sistema que son relevantes desde su punto de vista, con un apropiado nivel de detalle.

Un *diagrama* es una representación gráfica de una colección de elementos de modelado, frecuentemente dibujada en forma de grafo. Un ejemplo muy conocido de diagrama es el diagrama de clases UML (Unified Modeling Language) [OMG, 2007c], que sirve para representar gráficamente los conceptos del modelo de clases (clases, herencia, atributos, etc.), que a su vez captura la vista estática de un sistema software.

El principal objetivo de MDE es basarse en modelos [Seidewitz, 2003] para desarrollar software.

3.4. Ciclo de vida del desarrollo de software

El término ciclo de vida del software describe el desarrollo de software, desde la fase inicial hasta la fase final. Generalmente, comprende la obtención de requisitos,

análisis, diseño, implementación, pruebas y despliegue de un subsistema o de una aplicación completa.

Con MDE, el ciclo de vida en el desarrollo de software cambia: se aumenta el nivel de abstracción para generar aplicaciones y se automatizan total o parcialmente diferentes pasos del ciclo de vida de desarrollo tradicional (Fig. 3.3).

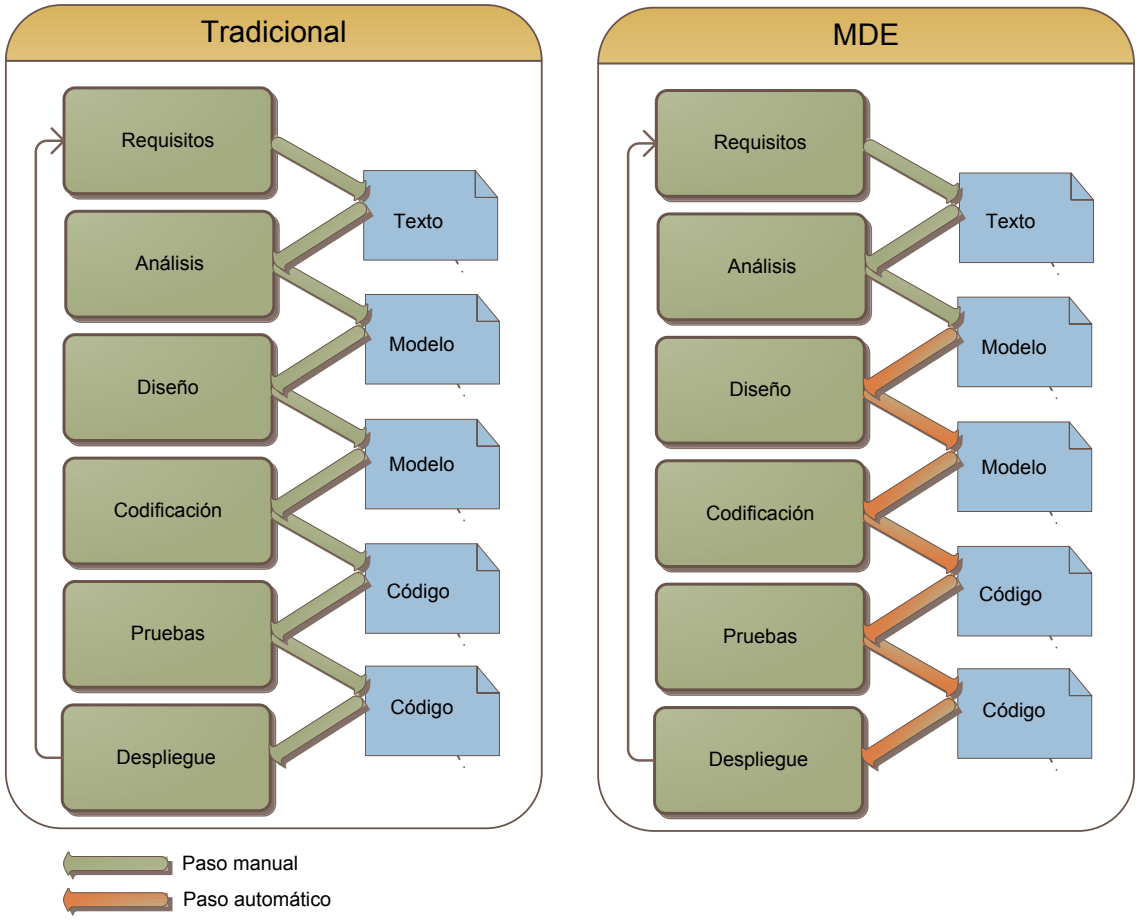


Figura 3.3: Ciclo de vida del desarrollo de software

La principal diferencia es que el documento de análisis es ahora un artefacto de primer orden en el desarrollo, ya que a partir del análisis de la aplicación se puede generar código de manera automática, saltándose parcial o totalmente la etapa de codificación. Por otra parte, muchas de las pruebas necesarias en el desarrollo tradicional ya no son necesarias, y otras se pueden realizar de manera automatizada.

3.5. Conceptos generales de MDE

La Fig. 3.4 [Völter and Stahl, 2006] representa los conceptos más importantes y básicos, que son independientes de la iniciativa MDE (véase sección 3.8) que se pueda emplear.

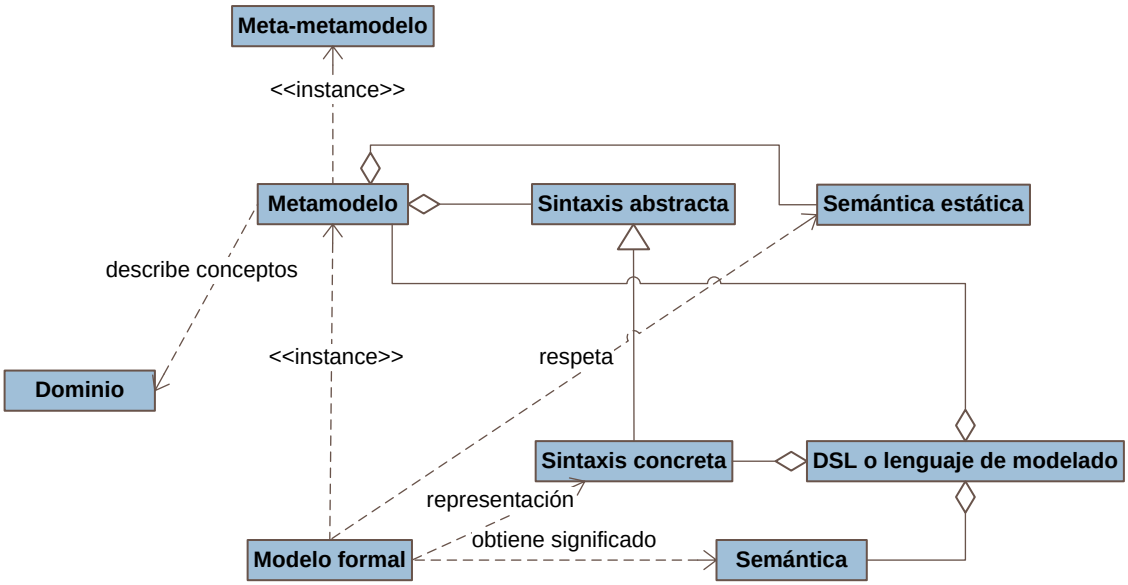


Figura 3.4: Conceptos generales de MDE

3.5.1. Dominio

El punto inicial de MDE siempre es un dominio, que delimita un campo de conocimiento. Hay dos tipos de dominios: los *dominios tecnológicos*, referentes a la tecnología de desarrollo de software y los *dominios profesionales*, referentes a los conceptos que manejará la aplicación. Los dominios pueden estar subdivididos en dominios más pequeños.

3.5.2. Metamodelo

El metamodelo sirve para describir de manera formal los conceptos relevantes que tiene el dominio. Además, es fundamental para conseguir automatizar el desarrollo de software [Frankel, 2003].

3.5.3. Meta-metamodelo

Para que los metamodelos puedan ser reutilizables, interoperables y portables, tiene que existir otro metamodelo en un nivel de abstracción superior, describiendo de forma única los conceptos que sirven para representar cualquier metamodelo de cualquier dominio. El meta-metamodelo es quien realiza esa función. Tiene la peculiaridad de que se define a sí mismo.

3.5.4. Sintaxis abstracta y sintaxis concreta

Los metamodelos están formados por una sintaxis abstracta y por una semántica estática. La sintaxis abstracta se centra en los elementos a nivel conceptual mientras que la sintaxis concreta se centra en cómo se representan los conceptos. De ahí se deduce que un metamodelo tendrá una sintaxis abstracta invariable pero podrán existir varias sintaxis concretas para representar los mismos conceptos.

La *sintaxis abstracta* de un lenguaje especifica su estructura, es decir, las construcciones, sus propiedades y los conectores que puede tener dicho lenguaje. Generalmente, también se especifican reglas del lenguaje en el metamodelo y así se evita la mala práctica de tener que validar los modelos en los generadores de código. Cuanto primero se detecten anomalías, más sencilla será la tarea de los demás componentes.

La *sintaxis concreta* de un lenguaje es necesaria para especificar la notación que los usuarios del lenguaje deben utilizar. Idealmente, cada concepto del dominio y del lenguaje se mapeará a una representación en la notación específica.

3.5.5. Semántica estática

La semántica estática de los metamodelos se basa en la sintaxis abstracta y tiene como misión hacer comprobaciones semánticas en los modelos para asegurar que están bien construidos.

3.5.6. Lenguajes de dominio específico

Los DSLs (Domain-Specific Languages o Lenguajes de Dominio Específico) [van Deursen, 1997, van Deursen et al., 2000] tienen el objetivo de poder expresar los conceptos de un dominio.

Están formados por uno o varios metamodelos, una o varias sintaxis concretas, y habitualmente una herramienta que lo soporta para facilitar la usabilidad.

Un DSL es un lenguaje definido pensado específicamente para abordar un problema específico de un dominio concreto y es el elemento principal de cualquier

solución de dominio específico. Los DSLs son denominados frecuentemente lenguajes de modelado.

3.5.7. Modelos formales

Con la infraestructura definida hasta ahora, se puede hablar ya de modelos formales, que son el punto inicial desde el que se automatizan las posibles transformaciones a entidades de menor nivel de abstracción (p.e., a partir de un modelo se podría generar automáticamente una aplicación Java).

Los modelos formales son instancias de los metamodelos y se representan gracias a una sintaxis concreta. Además, tienen que respetar la semántica estática que tenga el metamodelo para poder realizar construcciones coherentes dentro de un dominio.

3.5.8. Semántica del espacio del problema

La semántica de un DSL hace referencia a que cada concepto de los modelos tiene un significado: cada vez que se incluye un elemento en un modelo, lo que se está añadiendo es significado.

A diferencia de lo que ocurre con los lenguajes de propósito general, gracias al uso de DSLs, se consigue que los conceptos de un lenguaje se mapeen directamente a conceptos del dominio que se modela, sin posibilidad de interpretaciones erróneas.

La semántica de un DSL debe estar bien documentada o ser lo suficientemente intuitiva para que los creadores de los modelos sepan qué conceptos se están utilizando del espacio del problema; se deben asociar fácilmente los elementos de un lenguaje con los conceptos del dominio.

3.6. Modelado de Dominio Específico

Cuando se tiene cierta experiencia en el desarrollo de software, se observa que muchos problemas son repetitivos. Además, en muchas ocasiones, estos problemas pertenecen a un dominio concreto de conocimiento. Como solución se puede utilizar un GPL (General Purpose Language o Lenguaje de Propósito General) como Java y C# o se puede recurrir a la utilización de un DSL.

3.6.1. Fundamentos del Modelado de Dominio Específico

A partir del concepto de DSL (Domain-Specific Language o Lenguaje de Dominio Específico) [van Deursen, 1997], se puede hablar también de DSM (Domain-Specific Modeling o Modelado de Dominio Específico) [Kelly and Tolvanen, 2008], que tiene

su origen en la existencia de muchos desarrollos de software similares para un mismo dominio de conocimiento que tienen una parte común y una parte variable (en algunas ocasiones la parte común no existe).

La parte común podría desarrollarse utilizando las técnicas de desarrollo tradicionales y la parte variable podría desarrollarse utilizando un DSL pensado para ese dominio específico, aumentando así la productividad.

Un ejemplo son las aplicaciones de trazabilidad alimentaria, en las que todas comparten un mismo motor de ejecución y una misma base de datos, pero deben ser adaptadas al proceso de fabricación de diferentes alimentos, como quesos o carnes. Si la parte variable del software dependiera únicamente de los diferentes procesos de fabricación, se podría crear un DSL para definirlos.

Tanto el concepto de DSM como de DSL son fundamentales para trabajar con MDE. La idea básica es crear lenguajes especialmente pensados para solucionar un problema en un dominio muy concreto, permitiendo así que las construcciones del lenguaje sean muy cercanas a los conceptos del propio dominio.

Para unificar la parte fija o común y la parte variable del software, se presentan dos posibles aproximaciones [Cook et al., 2007]

- Interpretativa. La parte común tiene un intérprete para procesar la parte variable. De ese modo, se consigue flexibilidad pero ofrece inconvenientes como la pérdida evidente de rendimiento y la dificultad para realizar la depuración o *debug*.

- Generativa. La parte común y la parte variable se unen y se compilan para generar la solución como un todo. Es más compleja de realizar pero evita las desventajas de la aproximación interpretativa.

3.6.2. Clasificación de los DSLs

Existen varias clasificaciones para organizar a los DSLs según alguna de sus propiedades; las dos clasificaciones más destacadas son:

Desde el punto de vista de la manipulación del lenguaje

Según se maneje el lenguaje, se puede hablar de lenguajes de dominio específico:

- Textuales. La mayoría de los lenguajes informáticos son textuales y están formados por un conjunto ordenado de sentencias. Un ejemplo muy conocido es el lenguaje SQL (Structured Query Language) para las consultas a bases de datos. Se podrían crear DSLs textuales de varias formas. La primera de ellas

consistiría en hacer una gramática (por ejemplo en la notación BNF - Backus-Naur formalism [Knuth, 1964]) para el lenguaje y posteriormente crear o utilizar un *parser* para la gramática (Yacc [Johnson and Johnson, 1975], Bison o Antlr son herramientas que sirven para generar un *parser*), con la dificultad que esto supone. Otra forma de crear un DSL podría ser utilizando XML, con la consiguiente limitación sintáctica pero con la gran ventaja que tiene la existencia de gran cantidad de herramientas para trabajar con XML. Lo primordial para trabajar con MDE es que el DSL esté basado en un metamodelo formal.

- Gráficos. En los últimos años están ganando gran aceptación los lenguajes gráficos. Como ejemplo puede citarse UML. Crear un lenguaje gráfico puede considerarse análogo a crear un lenguaje textual, con la diferencia de que en lugar de trabajar directamente con texto habrá que crear mapeos desde la notación gráfica. Prácticamente todos los DSLs gráficos tendrán una notación que consistirá en varios conectores y figuras simples que serán la base para crear otras más complejas. Un DSL gráfico tendrá un metamodelo compuesto por clases que representan un concepto del dominio (típicamente mapeados como figuras en sus representaciones en diagramas), y por relaciones entre clases (típicamente mapeados como conectores). También contarán con restricciones que sirven para comprobar que los diagramas que representan al modelo son válidos. Otro importante concepto es la serialización, necesaria para guardar los elementos de los diagramas de manera persistente, siendo aconsejable que se haga en un formato que fomente la interoperabilidad como XML.

Hay que añadir trabajos que, como el de Tolvanen [Tolvanen, 2008], mencionan la existencia de otros tipos de DSL, como por ejemplo la mezcla entre el textual y el gráfico, tablas, formularios, árboles, etc.

Desde el punto de vista del dominio del problema

Según el punto de vista del dominio del problema, los lenguajes de dominio específico se clasifican en:

- Horizontales. Los DSLs horizontales se emplean cuando el cliente que utilizará el software final no pertenece a ningún sector industrial específico. Un ejemplo sería un DSL para generar interfaces de usuario en aplicaciones de escritorio como los Windows Forms de Visual Studio.

- Verticales. A diferencia de los DSLs horizontales, en los DSLs verticales los clientes pertenecen a un mismo sector industrial. Un ejemplo podría ser un DSL para desarrollar la parte variable del software de trazabilidad alimentaria comentado previamente en este capítulo (véase sección 3.6.1).

3.6.3. Requisitos de un DSL

Según [Kolovos et al., 2006] los requisitos necesarios para la construcción de un DSL son:

Partes interesadas

Las personas interesadas (*stakeholders*) que intervienen en el desarrollo de un DSL son:

- Ingenieros. Son los responsables de escoger o desarrollar un DSL; necesitan poseer una alta capacidad de abstracción.
- Clientes. Son los expertos en el dominio del DSL, y por tanto conocedores de la información más relevante.
- Desarrolladores. Son quienes típicamente utilizan el DSL para desarrollar la solución. También hacen otras tareas como configurar o integrar el software.

Límites

Es muy importante delimitar qué parte del sistema será creada con un DSL y qué parte no, es decir, que parte se desarrollará con un GPL y qué parte es más susceptible de hacerse con un DSL.

Características

Hay muchas características que son importantes en el desarrollo de un DSL:

- Los elementos del lenguaje deben corresponderse con los conceptos del dominio al que pretenden representar.
- Cada elemento del lenguaje se utiliza para representar exactamente un concepto del dominio.
- Deben existir herramientas para trabajar con el lenguaje.
- El DSL y las herramientas que lo soportan tienen que poder interoperar con otros lenguajes con un mínimo esfuerzo.

- El DSL y las herramientas que lo soportan tienen que poder ser extendidas con otros elementos adicionales.
- Debe existir una justificación temporal para la creación de un DSL de modo que sea rentable: puede no ser adecuado crear un DSL que únicamente sea válido un periodo de tiempo corto.
- El lenguaje debe ser lo más simple posible para representar los conceptos del dominio.
- Se deben proveer mecanismos para crear sistemas con calidad como por ejemplo *pre y post* condiciones.
- La escalabilidad, pese a ser una característica deseable, no es un requisito estrictamente necesario, puede haber DSLs pensados únicamente para sistemas muy pequeños.
- Por razones obvias, la usabilidad del lenguaje también es una característica deseable.

3.6.4. Ventajas e inconvenientes del uso de DSLs

Según se cita en [Cook et al., 2007], existen múltiples beneficios derivados del uso de DSLs, entre los que se pueden destacar:

- Con un DSL es mucho menos probable cometer errores en la representación de un problema de un dominio que con un lenguaje de propósito general.
- Trabajar con los términos de un dominio concreto facilita la comprensión de los modelos que representan al software a personas no expertas en tecnologías de desarrollo de software.
- Cuando se trabaja con modelos expresados utilizando DSLs, dichos modelos pueden ser validados en el mismo nivel de abstracción que el espacio del problema, lo cual implica que los errores serán detectados con más antelación.
- Los modelos podrían ser utilizados para simular salidas de las soluciones que se crearán.
- Cuando se captura conocimiento de un determinado dominio en un modelo, es más sencillo realizar migraciones entre diferentes tecnologías.

- Los DSLs suelen proporcionar un API específica de dominio para manipular sus modelos y así aumentar la productividad.

Sin embargo, si se piensa en crear un DSL desde cero para solucionar un problema concreto, habría que estudiar algunos factores que pueden afectar a la decisión final:

- Tiempo en investigación para determinar el DSL que se creará.
- Coste necesario para probar el DSL.
- Dificultades añadidas en el despliegue de la aplicación.
- Necesidad de documentación adicional.
- Preparación del equipo de desarrollo.

Por lo tanto, habrá que estudiar en cada caso cuando merece realmente la pena crear y utilizar un DSL, o cuando no merece la pena el coste que esto supone. Dicho estudio no es trivial y es objetivo de múltiples investigaciones.

3.6.5. Elementos necesarios para DSM

Existen varios elementos imprescindibles para lograr crear con éxito una solución de dominio específico [Kelly and Tolvanen, 2008]. Básicamente, se utiliza una arquitectura en capas (Fig. 3.5), que puede variar dependiendo del caso, llegándose incluso a eliminar la plataforma base o *framework de dominio* en algunas ocasiones:

Lenguajes y modelos

La clave en DSM consiste en crear un modelo que confirme con un metamodelo mediante el empleo de un DSL. Como los lenguajes y los modelos ya han sido abordados en este documento, no se entrará en más detalle.

Generadores

Los generadores tienen que obtener la información de los modelos y generar artefactos (p.e., código fuente) a partir de ellos. En los casos más sencillos se mapea cada símbolo del lenguaje a un determinado fragmento de código; el código generado también podría ser variable dependiendo de los valores de las entrada. La idea es que a partir del modelo se generen artefactos sin necesidad de intervención manual.

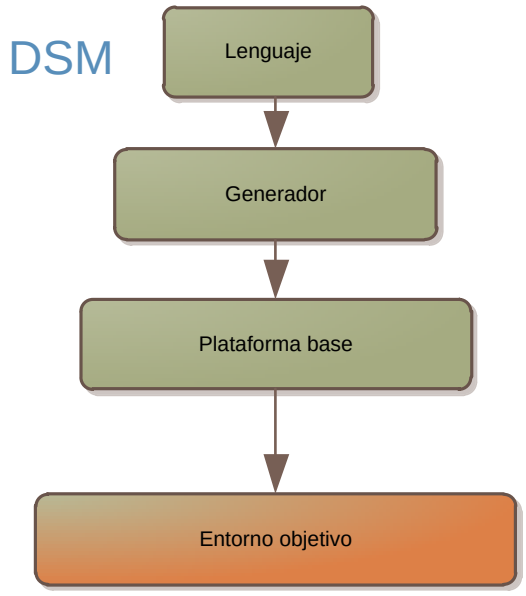


Figura 3.5: Arquitectura básica de una solución de dominio específico

Plataforma base

Es la interfaz entre el código generado y la plataforma o entorno objetivo. En algunos casos la relación es directa y no hace falta añadir más código que el que se haya generado automáticamente. En otros casos, sin embargo, es necesario utilizar plataformas base para añadir código de forma que todas las soluciones utilicen un código común fijo creado previamente. Las plataformas base habitualmente reciben otros nombres como *framework arquitectónico*, *framework base* o *framework de dominio*.

Entorno objetivo

El entorno objetivo es la máquina física o virtual para la cual se pretende desarrollar un sistema. Por ejemplo, un entorno objetivo podría ser una determinada versión de la máquina virtual de Java.

3.6.6. Herramientas DSM

A continuación se mencionan algunas de las herramientas software que se utilizan en la actualidad para dar soporte a DSM.

MetaEdit+

MetaEdit+[5] está basada en la herramienta MetaEdit, pero mejora aspectos arquitecturales que no habían sido resueltos correctamente y que ahora aumentan la escalabilidad y la eficiencia de la herramienta. Se puede realizar el metamodelo y el modelado en un mismo entorno (Fig. 3.6).

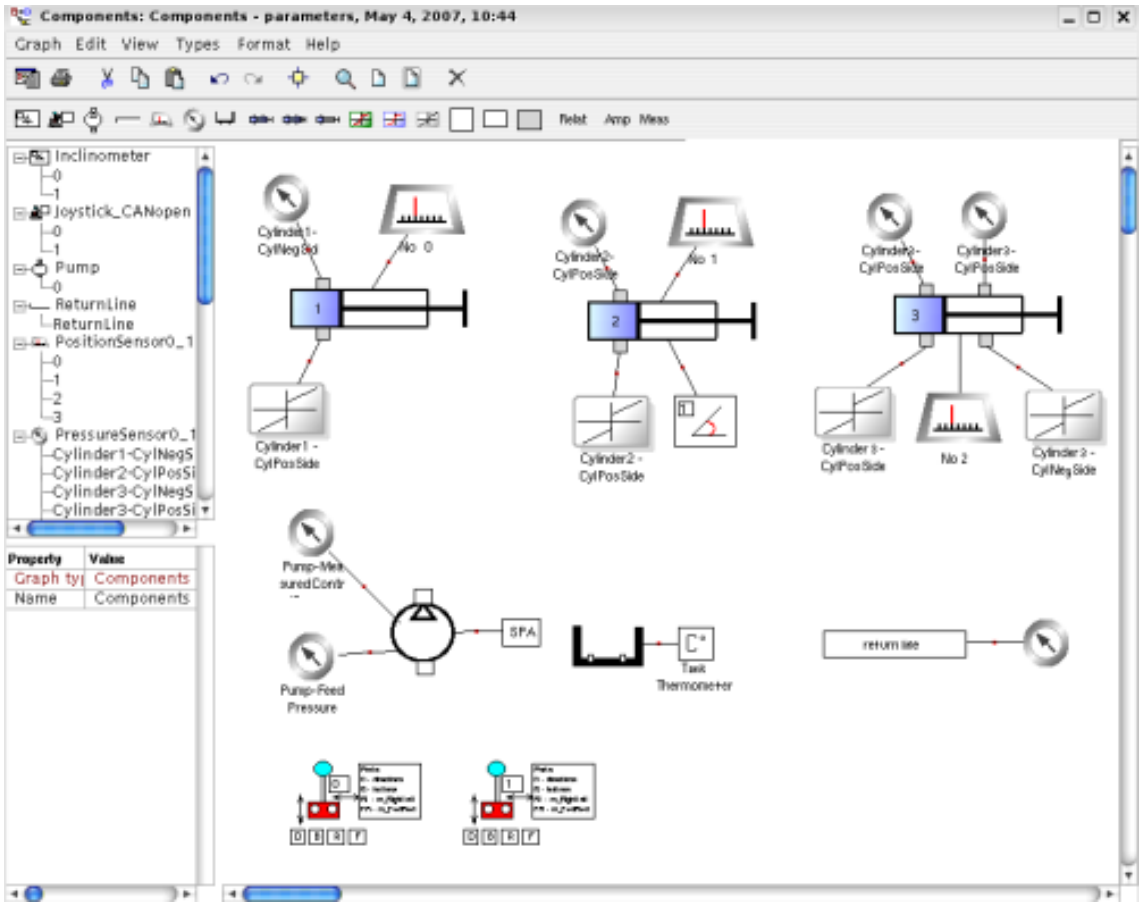


Figura 3.6: Aspecto general de MetaEdit+ (ejemplo)

General Modeling Environment

GME (General Modeling Environment) [Ledeczi et al., 2001] está basado en una Tesis Doctoral en la que se muestra un meta-metamodelo para crear metamodelos en el dominio de la ingeniería eléctrica, y un Entorno General de Modelado que se configura con unos archivos generados automáticamente a partir de los metamodelos.

El meta-metamodelo de GME difiere significativamente respecto a otros meta-metamodelos, debido fundamentalmente a la herencia de la ingeniería eléctrica. Los

[5] http://www.metacase.com/

metamodelos se especifican con un lenguaje de modelado específico y las restricciones con un lenguaje declarativo (Fig. 3.7).

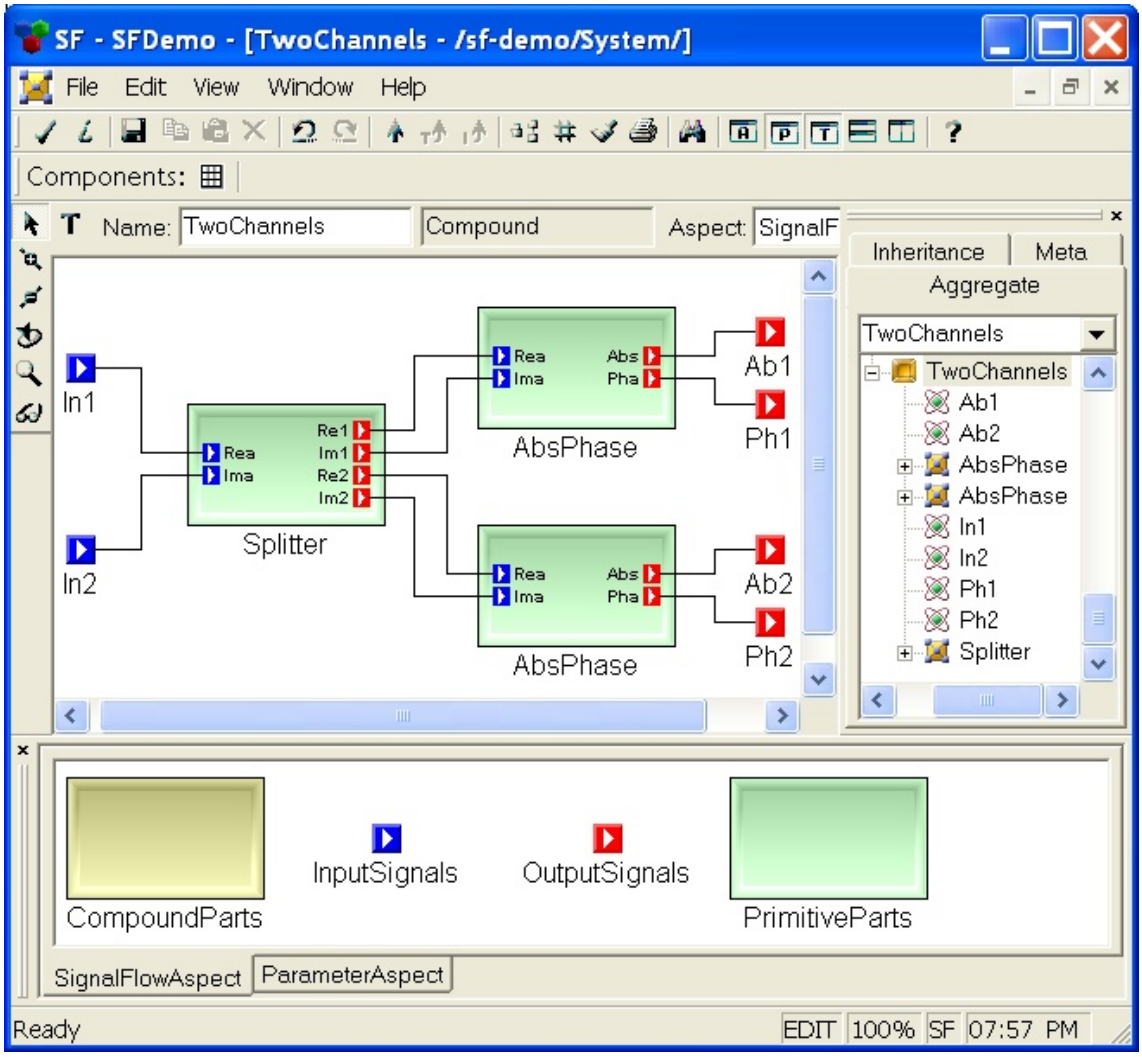


Figura 3.7: Aspecto general de General Modeling Environment (ejemplo)

DSL Tools

La primera versión de DSL Tools (Domain-Specific Language Tools) [Cook et al., 2007] fue lanzada en Visual Studio 2005 SDK 3.0 y sirve para ofrecer nuevas herramientas para llevar a cabo la visión de las Software Factories de Microsoft (véase capítulo 6). Es un software gratuito pero sólo puede utilizarse dentro de Visual Studio.

Las DSL Tools son un conjunto de frameworks, lenguajes, editores, generadores y guías que facilitan al usuario crear su propio lenguaje y herramientas (Fig. 3.8).

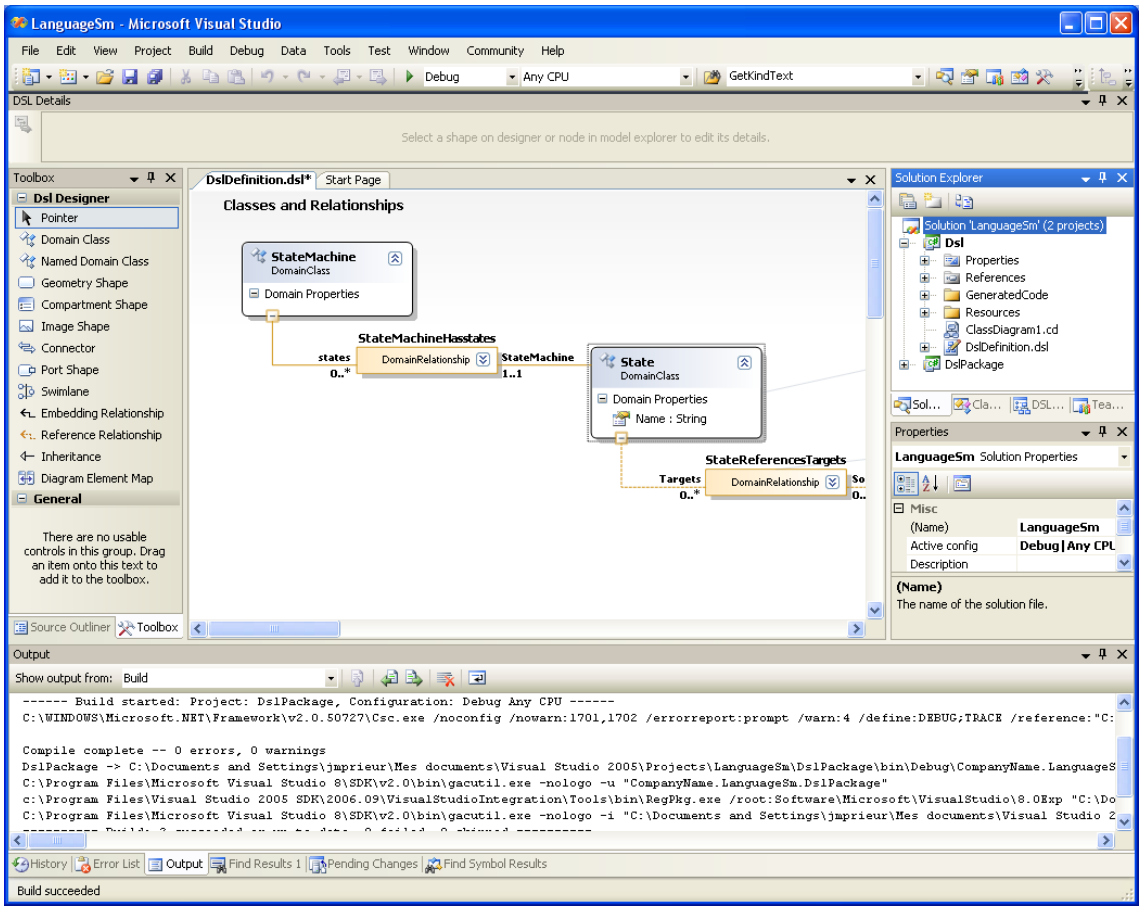


Figura 3.8: Aspecto general de DSL Tools (ejemplo)

Graphical Modeling Framework

El Eclipse Modeling Project [Gronback, 2009] es un proyecto creado para el entorno Eclipse[6] que consta de varios subproyectos:

- El EMF (Eclipse Modeling Framework) [Budinsky et al., 2003,Steinberg et al., 2009] permite la generación de vistas en forma de árbol y hojas de propiedades a partir de metamodelos definidos en XML.

- El GEF (Graphical Editing Framework) fue un framework que permite la generación de editores gráficos utilizando para ello Draw2D.

- El problema fue que EMF y GEF no estaban correctamente integrados por lo que se creó el GMF (Graphical Modeling Framework)[7], como una evolución de

[6] http://www.eclipse.org/
[7] http://www.eclipse.org/modeling/gmf/

GEF que sí se integra con EMF (Fig. 3.9) (véase sección 10.5 para más detalle sobre las herramientas MDE de Eclipse).

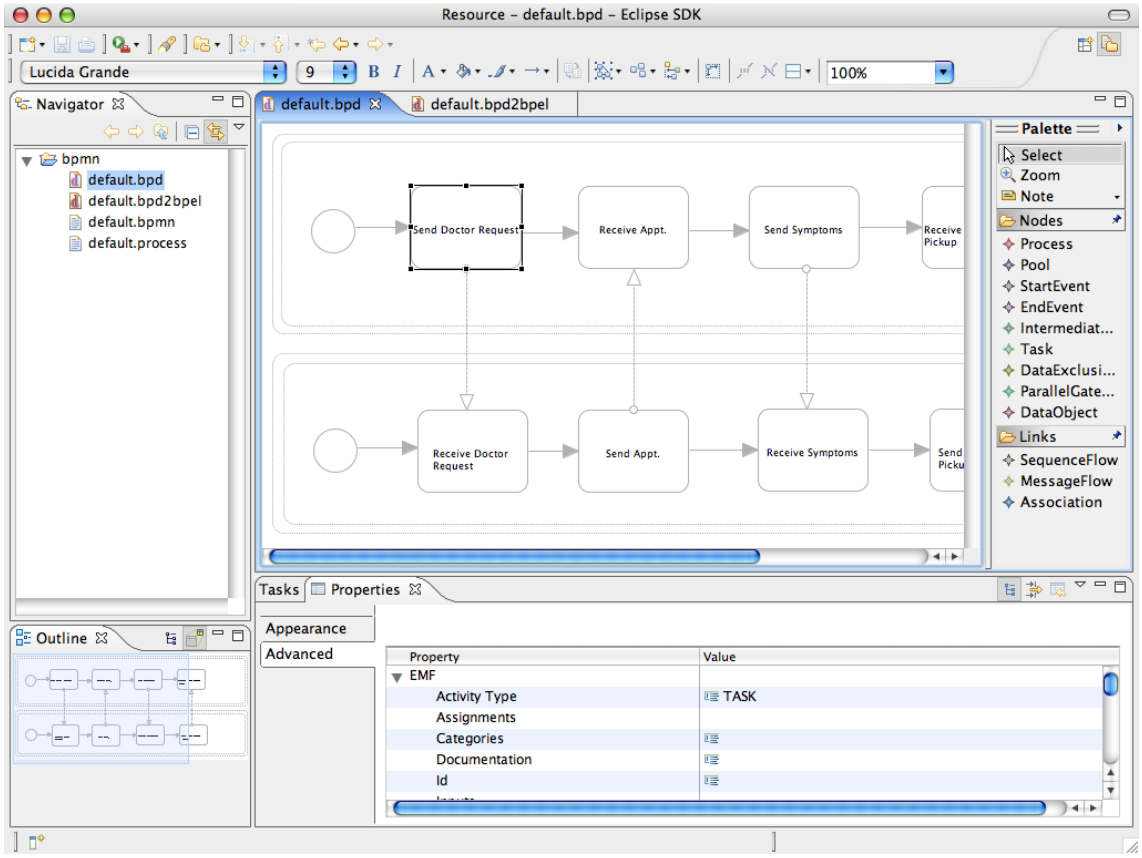


Figura 3.9: Aspecto general de Graphical Modeling Framework (ejemplo)

3.7. MDE versus desarrollo tradicional

Existe una serie de diferencias derivadas de utilizar un planteamiento basado en MDE respecto al desarrollo de software basado en un enfoque más tradicional [Kleppe et al., 2003].

3.7.1. Productividad

Cuando se genera un producto software, se crea paralelamente una gran cantidad de documentación como por ejemplo la captura de requisitos, el análisis o el diseño. El problema es que el software casi nunca permanece tal cual se ha ideado inicialmente, por lo que cambios de cualquier tipo provocan que se tenga que cambiar toda

la documentación, desde el más alto nivel hasta el más bajo nivel. En la práctica, lo que se hace casi siempre es modificar el código sin actualizar la documentación. De ese modo, al final puede parecer que realizando una documentación exhaustiva se está perdiendo el tiempo.

MDE permite que cualquier cambio afecte de manera directa a los modelos, ya que los artefactos se generarán de forma automatizada (en un caso ideal), permitiendo así aumentar la productividad de desarrollo. En lugar de generar artefactos, otra posibilidad válida sería interpretar o compilar los modelos para ser ejecutados en una máquina física o virtual.

3.7.2. Portabilidad

Continuamente aparecen nuevas tecnologías o versiones de tecnologías existentes, lo cual provoca que haya que realizar migraciones de unas tecnologías a otras para mantener al día los sistemas, con el consiguiente gasto tanto económico como temporal.

Con MDE, los modelos pueden ser independientes de la plataforma, por lo que las modificaciones afectan solamente al resto de las fases, es decir, el peso de las modificaciones recae en las herramientas utilizadas para realizar las transformaciones.

3.7.3. Interoperabilidad

Cada vez más, los sistemas necesitan comunicarse con otros sistemas y es muy probable que los sistemas estén utilizando tecnologías diferentes, con lo que conseguir una buena interoperabilidad es una tarea difícil de lograr. Puede que los modelos tengan relaciones entre ellos, y debido a que diferentes modelos pueden estar pensados para diferentes tecnologías, es muy probable que no se puedan comunicar entre ellos. Para solucionar este problema, se genera junto con los modelos los llamados puentes de comunicación (*bridges*), permitiendo así la interoperabilidad entre sistemas.

3.7.4. Mantenimiento y documentación

La tarea de documentar el software suele interesar más que a los que lo desarrollan, a los que lo tendrán que mantener o utilizar, por lo que generalmente no se pone mucho empeño en lograr una documentación de una buena calidad. Actualmente, el hecho de que los comentarios en el código fuente se puedan utilizar como documentación permite que se genere una documentación de una mayor calidad. El problema de la documentación en el código parece resuelto pero no así otros documentos como puede ser por ejemplo el diagrama objeto-relacional. Con MDE se

puede generar todo el sistema a partir de los modelos y toda la documentación de alto nivel estaría en los modelos, reduciéndose así la cantidad de documentación a crear y mantener.

3.8. Iniciativas MDE

A lo largo de los años han surgido varias iniciativas importantes para hacer realidad MDE. Algo a tener en cuenta es que aunque los autores o promotores de una iniciativa no mencionen explícitamente a otras, en realidad prácticamente siempre se manejan los mismos conceptos desde otras perspectivas. Así, con los conceptos básicos ya señalados en este capítulo se podría tener una base suficiente para entender cualquier iniciativa o punto de vista respecto a cómo trabajar con MDE. Las iniciativas más relevantes son:

- MDA (Model-Driven Architecture o Arquitectura Dirigida por Modelos). MDA ha adquirido tanta importancia que muchas veces se asocia directamente con el concepto de MDE (véase capítulo 5).

- Software Factories o Factorías Software. Gracias al apoyo de Microsoft, las Software Factories están consiguiendo cada día más protagonismo en el desarrollo de software (véase capítulo 6).

- AC-MDSD (Architecture-Centric Model-Driven Software Development o Desarrollo de Software Dirigido por Modelos y centrado en la Arquitectura). AC-MDSD [Völter and Stahl, 2006] es una iniciativa MDE que se centra fundamentalmente en crear un framework en el que se incluyen los conceptos arquitectónicos más importantes de una familia de software. De esa forma, se pueden hacer más sencillos los modelos ya que se centrarían únicamente en aspectos relacionados con un domino específico. Al tener la arquitectura software construida, se crean aplicaciones completas. A partir de los modelos se realiza una transformación directa a código mediante una plantilla. El código específico que faltara por incluir se inserta manualmente en regiones del código reservadas para tales efectos; así, al regenerar la solución, el código escrito a mano se conserva.

- GP (Generative Programming o la Programación Generativa). GP [Czarnecki and Eisenecker, 2000] es una iniciativa para MDE, que según sus autores se puede definir como:

> ❝*La Programación Generativa es un paradigma de la ingeniería del software basado en modelar familias de software de modo que dada una particular especificación de requisitos, se pueda desarrollar automáticamente y bajo demanda, un producto final o casi final ajustado a las necesidades y permitiendo la reusabilidad de componentes gracias a la configuración del conocimiento.*❞

Se generan aplicaciones completas por medio de la composición de componentes atómicos. Estos componentes han sido optimizados para aspectos específicos gracias a la configuración del conocimiento realizada con un generador que transforma los modelos a código.

- MIC (Model-Integrated Computing o Computación Integrada con Modelos). MIC [Sztipanovits and Karsai, 1997] es una iniciativa MDE que surgió en el mundo de los sistemas distribuidos en tiempo real. Propone usar DSLs y varios modelos para especificar todos los aspectos que puede tener un sistema. Con MIC, los modelos son el centro de todo el ciclo de vida de los sistemas y por lo tanto deben existir transformaciones entre modelos para tener diferentes representaciones y así poder realizar el análisis, la verificación o, por ejemplo, simulaciones.

- Language-Oriented Programming o Programación Orientada al Lenguaje [Ward, 1994]. El software MPS (Meta Programming System) permite definir lenguajes propios (por medio de un metamodelo) integrados en el entorno de desarrollo de MPS. Cuando se define un lenguaje, también se define su editor, su compilador, las transformaciones que se hacen y el soporte para hacer operaciones de depuración.

La lista de iniciativas MDE no está cerrada puesto que en un área cambiante donde continuamente aparecen nuevas ideas que, en algunos casos, pueden considerarse una nueva iniciativa MDE. Sin embargo, se han citado las que en el momento de la escritura de esta Tesis son consideradas como las más relevantes.

3.9. Conclusiones

La ingeniería dirigida por modelos es la última aportación importante de la ingeniería del software en cuanto a la mejora de los métodos de desarrollo de software. Ofrece grandes ventajas sobre el desarrollo tradicional, principalmente cuando se trabaja con familias de productos. Sin embargo, también requiere un esfuerzo extra

y una gran capacidad de abstracción por parte de quienes realizan las herramientas para que otros se vean beneficiados.

En los siguientes capítulos se detallarán algunas de las principales iniciativas MDE existentes. Pero primero, en el próximo capítulo, se introduce el otro concepto que, junto con MDE, es imprescindible en este trabajo, *la integración continua (CI)*.

CAPÍTULO 4

Integración continua

La CI (Continuous Integration o Integración Continua) es una práctica de desarrollo de software en la que los diferentes miembros de un equipo de desarrollo de software integran frecuentemente su trabajo para obtener un sistema completo o parcialmente completo [Herbsleb and Grinter, 1999]. Cada integración es verificada con una construcción automática del software acompañada, generalmente, de pruebas para detectar errores tan rápido como sea posible. Además, se generan informes para mostrar el resultado de cada integración.

Las principales ventajas de esta práctica son la reducción de problemas de integración y el desarrollo de un software con una mayor cohesión y, por tanto, con una mayor calidad y menores riesgos [Duvall et al., 2007].

El objetivo de este capítulo es introducir qué es y para qué se utiliza la práctica de la integración continua en los desarrollos de software. Se analizan las ventajas y los inconvenientes, así como las mejores prácticas recomendadas para implantar su uso. Por último, se estudian algunas de las herramientas que actualmente se utilizan para llevar a cabo dicha práctica.

* * * *

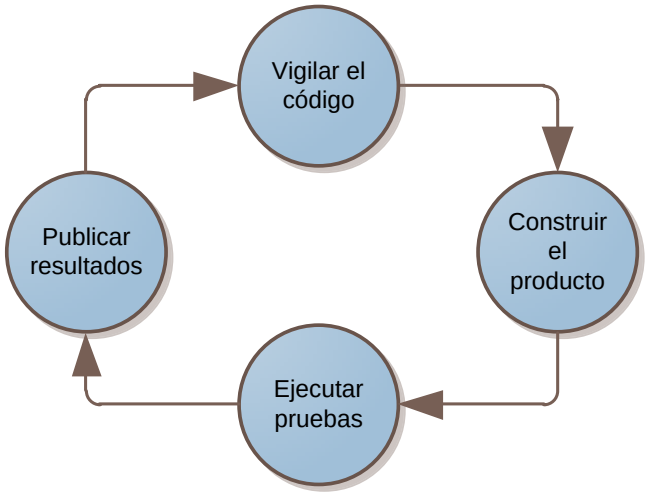


Figura 4.1: Ciclo de vida de la integración continua

4.1. Introducción

La integración continua[1] es una práctica recomendada por muchas metodologías software. Por ejemplo, fue recogida como una de las doce prácticas originales del XP (Extreme Programming) [Beck, 1999] y forma parte de las recomendaciones del UP (Unified Process) [Jacobson et al., 1999].

En realidad, la CI se considera un miembro más del equipo de desarrollo que se encarga de:

- Monitorizar el código fuente.
- Compilar cada cambio.
- Probar el resultado una vez compilado mediante pruebas automatizadas.
- Notificar a los responsables cualquier problema que se haya producido durante el proceso [Richardson and Gwaltney, 2005].

La Fig. 4.1 ilustra el proceso de CI y en la Fig. 4.2 se detalla el mismo.

Existen dos tipos de CI, aunque a veces es difícil conocer cuál es la línea que separa una clasificación de otra. Por un lado está los *centralizados*, utilizados por grandes empresas como Microsoft [Cusumano and Selby, 1997], y por otro lado

[1]En la sección 17.1 se explica cómo es un proceso de integración continua tradicional

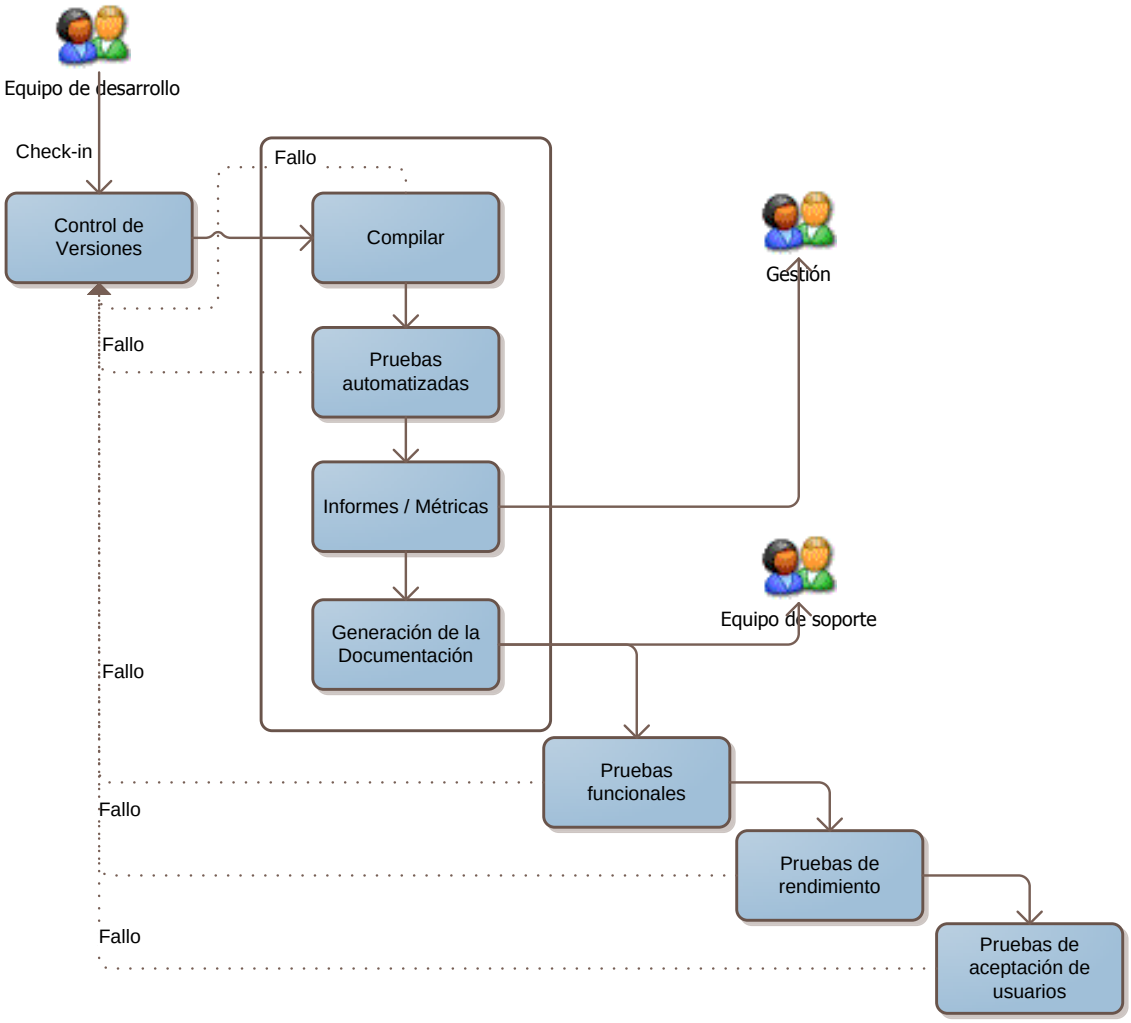


Figura 4.2: Ciclo de vida de la integración continua (extendido)

está los *descentralizados*, de uso mayoritario por su sencilla aplicación. La diferencia entre ambos tipos es que en la segunda opción, la decisión de cuando integrar y la responsabilidad para conseguir una integración satisfactoria es de cada desarrollador individual.

Cabe destacar que existen pocos trabajos científicos relacionados con la CI; uno de los más destacados estudia cómo los desarrolladores trabajan en dos proyectos de código abierto de éxito, FreeBSD y Mozilla [Holck and Jorgensen, 2004]. La conclusión de este estudio fue que la integración continua reemplaza en cierto grado a la ingeniería tradicional del software respecto a mecanismos de coordinación y documentos de diseño.

4.2. Desarrollo de Software Ágil

El desarrollo ágil de software es un término acuñado en 2001 que hace referencia al conjunto de metodologías de desarrollo basadas en un desarrollo incremental, en el que tanto los requisitos de los clientes como la solución del desarrollo evolucionan mediante la colaboración entre los diferentes equipos de las organizaciones.

Debido a que el desarrollo de software ágil va ganando aceptación con el paso del tiempo, también lo hacen sus prácticas, y precisamente una de las más importantes es la integración continua [Richardson and Gwaltney, 2005].

4.2.1. Manifiesto Ágil

El Manifiesto Ágil[2] fue creado en marzo de 2001 por 17 críticos convocados por Kent Beck, autor del libro Extreme Programming Explained [Beck, 1999]. La idea era debatir sobre nuevas técnicas y procesos para desarrollar software como alternativa a las metodologías tradicionales formales, debido a que son consideradas excesivamente pesadas y poco flexibles.

El debate derivó en cuatro postulados que contienen los principios en los que se basan las metodologías alternativas, dando lugar al Manifiesto Ágil, firmado por todos los integrantes de la reunión:

> ❝*Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar:*
>
> 1. *A los individuos y su interacción, por encima de los procesos y las herramientas.*
> 2. *El software que funciona, por encima de la documentación exhaustiva.*
> 3. *La colaboración con el cliente, por encima de la negociación contractual.*
> 4. *La respuesta al cambio, por encima del seguimiento de un plan.*
>
> *Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda.*❞

[2] http://agilemanifesto.org/

4.2.2. Principios

Los firmantes del Manifiesto Ágil redactaron doce principios o prácticas que ellos mismos utilizan:

> ❝*Seguimos estos principios:*
>
> 1. *Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.*
> 2. *Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.*
> 3. *Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.*
> 4. *Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.*
> 5. *Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.*
> 6. *La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.*
> 7. *El software que funciona es la principal medida del progreso.*
> 8. *Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.*
> 9. *La atención continua a la excelencia técnica enaltece la agilidad.*
> 10. *La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.*
> 11. *Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.*
> 12. *En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.*❞

4.2.3. Adopción en el desarrollo de software

Hay un consenso en la comunidad científica respecto a que es más útil utilizar las prácticas recomendadas por las diferentes metodologías que más se adapten a las necesidades de un equipo u organización, que utilizar metodologías completas sin tener en cuenta otras prácticas que podrían ser importantes.

Como dijo Ivar Jacobson [Jacobson, 2007]:

> ❝*En el futuro no va a haber más metodologías prescriptivas como lo fueron UP, procesos estilo CMMI o procesos ágiles estilo XP, SCRUM y otros. En cambio, lo que va a haber es una paleta de prácticas. Las prácticas van a venir primero y las metodologías van a ser meras colecciones de prácticas que las organizaciones van a escoger de la paleta.*❞

4.3. Patrones recomendados

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software. En cualquier práctica del día a día es aconsejable utilizar patrones y evitar anti patrones que expertos (p.e., [Fowler, 2009a]) en la materia ya han identificado y validado. A continuación se listan patrones para la realización de CI que a día de hoy están siendo utilizados con éxito:

4.3.1. Guardar todas las fuentes en un repositorio de código

Es aconsejable que todos los proyectos de software tengan un sistema de control de versiones (véase capítulo 8) que permita trabajar con el repositorio en el que se almacenan las fuentes.

Se deberían almacenar en un único repositorio todas las fuentes necesarias para construir los productos, teniendo en cuenta que únicamente serán necesarias las fuentes y nada de lo que se construye a partir de las mismas. Estas fuentes incluyen *scripts* de instalación, archivos de configuración, esquemas de la base de datos e incluso librerías de terceros. La idea es que se puedan construir los productos en una máquina que cuente únicamente con lo mínimo indispensable.

También es aconsejable que todo el proyecto siga una misma línea base y evitar el uso de múltiples ramificaciones de versiones en el repositorio salvo en casos muy puntuales.

4.3.2. Automatizar la construcción

Desplegar el sistema que se construye es una tarea complicada que implica muchas operaciones como: compilaciones, movimiento de archivos y/o inserción de datos en una base de datos. Como la mayoría de dichas operaciones puede ser automatizada, la idea es que nunca se hagan de forma manual.

Para automatizar la construcción del software se utilizan herramientas como Ant[3], NAnt[4] o MSBuild[5], que analizan qué partes del proceso tienen que ser cambiadas y evitan de ese modo tener que realizar toda la construcción para cada cambio.

4.3.3. Utilizar pruebas automatizadas

Una buena manera de detectar errores rápida y eficientemente es incluir pruebas automáticas en el proceso de construcción, de modo que si algo falla, el software no se genera. Obviamente, las pruebas probablemente no detecten la totalidad de los errores en el código, pero detectarán una gran cantidad de ellos. En los últimos tiempos, han experimentado un gran auge aproximaciones de desarrollo de software como XP o TDD (Test Driven Development) [Beck, 2003] que apuestan fuertemente por las pruebas automatizadas. Para lograrlo, se suelen utilizar herramientas como JUnit[6] o NUnit[7].

4.3.4. Actualizar el repositorio todos los días

Cada desarrollador debería actualizar su copia con la información del repositorio cada día, realizar pruebas en su copia, y si todo es correcto, subirlo al repositorio compartido por todo el equipo de desarrollo.

De este modo se evita que haya muchas diferencias entre una actualización y otra; además permite solucionar rápidamente posibles conflictos que pueden ocurrir en el código desarrollado paralelamente por diferentes personas. Así, será mucho más probable detectar un defecto que si ha pasado mucho tiempo y el software ha seguido evolucionando.

4.3.5. Construir siempre el software

Pese a las grandes ventajas que puede ofrecer la CI, es muy complicado concienciar a todos los desarrolladores que han de actualizar su copia regularmente, después

[3] http://ant.apache.org/
[4] http://nant.sourceforge.net/
[5] http://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx
[6] http://www.junit.org/
[7] http://www.nunit.org/

probarla y por último subir todos los cambios al repositorio.

Así, muchas veces se actualizan los repositorios sin tener el software probado, provocando fallos y problemas. También, es frecuente que los desarrolladores utilicen distintas configuraciones software para trabajar, con lo que lo que a unos les funciona, a otros les falla.

4.3.6. Lograr construcciones rápidas

En algunas ocasiones la construcción del software se hace un proceso demasiado lento, siendo la ejecución de ciertas pruebas como las de carga o rendimiento un cuello de botella típico, requiriendo incluso horas para ser llevadas a cabo. Para evitar este tipo de problemas se hacen *construcciones por etapas*.

La idea es tener dos o más construcciones realizándose en secuencia, aunque sólo una será la construcción que se iniciará directamente cuando el desarrollador actualiza el repositorio. La construcción inicial deberá contener pruebas básicas que no requieran un tiempo excesivo para llevarse a cabo y será con la que trabajen el resto de desarrolladores. Sin embargo, cuando haya tiempo y recursos, se irán haciendo el resto de construcciones de la secuencia, que al ser secundarias pueden llevarse a cabo en mucho más tiempo (incluso horas).

4.3.7. Probar en un clon del equipo de producción

Cuantas más diferencias haya entre el equipo en el que se hacen las pruebas y el equipo en el que se desplegará el sistema final, más riesgo habrá de que se produzcan problemas. Recientemente está creciendo el uso de máquinas virtuales que son clones del sistema final. De ese modo, se consigue hacer muchas pruebas o simular muchas máquinas, en un único equipo.

4.3.8. Obtener la última versión ha de ser fácil

Todas las personas implicadas en el desarrollo de software han de poder obtener la última versión del software y ejecutarla fácilmente. Así, podrán ver los últimos cambios o hacer pruebas, con la seguridad de que están trabajando con la versión de software correcta.

4.3.9. Saber el estado de la última versión del software

Del mismo modo que todo el equipo ha de poder obtener fácilmente la última versión del software, también sería conveniente que pudieran visualizar de algún

modo el estado de dicha versión. Por ejemplo, se pueden utilizar Webs, correos electrónicos o archivos de registro de eventos en los que se dan detalles de la última construcción.

4.3.10. Automatizar el despliegue

Es aconsejable tener automatizado el proceso de despliegue, ya que ahorra una gran cantidad de tiempo. Por ese motivo, se suelen emplear *scripts* para poder desplegar el software directamente. Además, es recomendable disponer de mecanismos de *rollback* para evitar problemas y poder volver a un estado anterior cuando se desee.

4.4. Ventajas de la integración continua

Gracias a la CI se pueden obtener unos interesantes beneficios, algunos de los cuales has sido citados en trabajos como [McConnell, 2004,Olsson, 1999,Ebert et al., 2001]. A continuación se muestra una lista de beneficios que diversos autores han identificado:

4.4.1. Reducción del riesgo

El riesgo es algo intrínseco a los proyectos dado su carácter temporal y único. Tiene su origen en la incertidumbre que está presente en todos los proyectos, y es por ello que la gestión de riesgos es la parte fundamental de la gestión activa de un proyecto [PMI, 2005].

Sin embargo, en el desarrollo de software todavía no se tiene una mentalidad tan global y ello provoca que no se tengan en cuenta los riesgos que hay que asumir cuando se integran diferentes componentes software de un proyecto.

Generalmente, la integración de los componentes se hace durante las últimas etapas planificadas del proyecto, y con el añadido de que es poco frecuente que un proyecto llegue a la etapa de integración según las previsiones iniciales. Este hecho hace que fracasen muchos proyectos debido a que no se entregan a tiempo, se aumentan los costes o se tiene que reducir su alcance. Con la CI se reduce el riesgo dado que los problemas durante el desarrollo se detectan antes.

4.4.2. Eliminación de defectos

Gracias a la integración continua es mucho más fácil detectar y eliminar defectos, ya que al hacerse pruebas en cortos periodos de tiempo, se aumentan las garantías

de que en el software no surjan fallos, o que al menos puedan aparecer muchos menos de los que aparecerían de no ser por la integración temprana.

Además, siempre es mucho más sencillo corregir defectos justo después de que aparezcan, sabiendo en qué fragmento del código pueden estar, que hacerlo mucho tiempo después. Por otra parte, los defectos son acumulativos y hay un factor psicológico que hace que resulte mucho más frustrante corregir muchos defectos al mismo tiempo, que hacerlo de uno en uno [Hunt and Thomas, 1999].

4.4.3. Mejores relaciones con los clientes

Realizar despliegues de manera frecuente y rápida es una gran ventaja porque puede ayudar a eliminar las barreras que existen entre los clientes y los desarrolladores. Esto permite observar las características del software rápida y fácilmente, y así crear un ambiente más colaborativo entre ambas partes.

4.4.4. Aumento de la moral del equipo de desarrollo

Gracias a que los desarrolladores ven rápidamente los resultados de su trabajo mediante la realización de despliegues iterativos y progresivos, se produce un aumento de su moral.

4.4.5. Estimaciones más acertadas

Como se tiene una idea más real del estado del desarrollo del proyecto en cada momento, se pueden hacer estimaciones de cuánto tiempo falta para acabar un proyecto con una mayor fiabilidad que de forma tradicional.

Una de las características de la CI es que evita la aparición del llamado *big bang*, originado cuando se realiza la integración únicamente al final del ciclo de desarrollo, provocando la aparición de una gran cantidad de errores e invalidando las estimaciones realizadas [Pressman, 1986].

4.4.6. Disponibilidad de la última versión del código

Gracias a la CI se puede tener acceso inmediato a la última versión de las fuentes integradas de un proyecto. De ese modo, resulta mucho más fácil hacer pruebas, demostraciones, liberación de versiones, etc.

4.4.7. Mejora de las capacidades colaborativas del equipo

El impacto negativo de introducir artefactos en el repositorio que no funcionan y que pueden ser incompatibles con los del resto, hace que los miembros del equipo realicen un trabajo más incremental y cuidadoso, teniendo siempre presente el trabajo de los demás.

4.4.8. Reducción de costes

Como se puede observar en la Fig. 4.3, cuanto más se tarda en detectar un error, más costosa será su reparación [Grady, 1999]. Por medio de la CI, los errores se detectan antes que utilizando prácticas de metodologías más tradicionales.

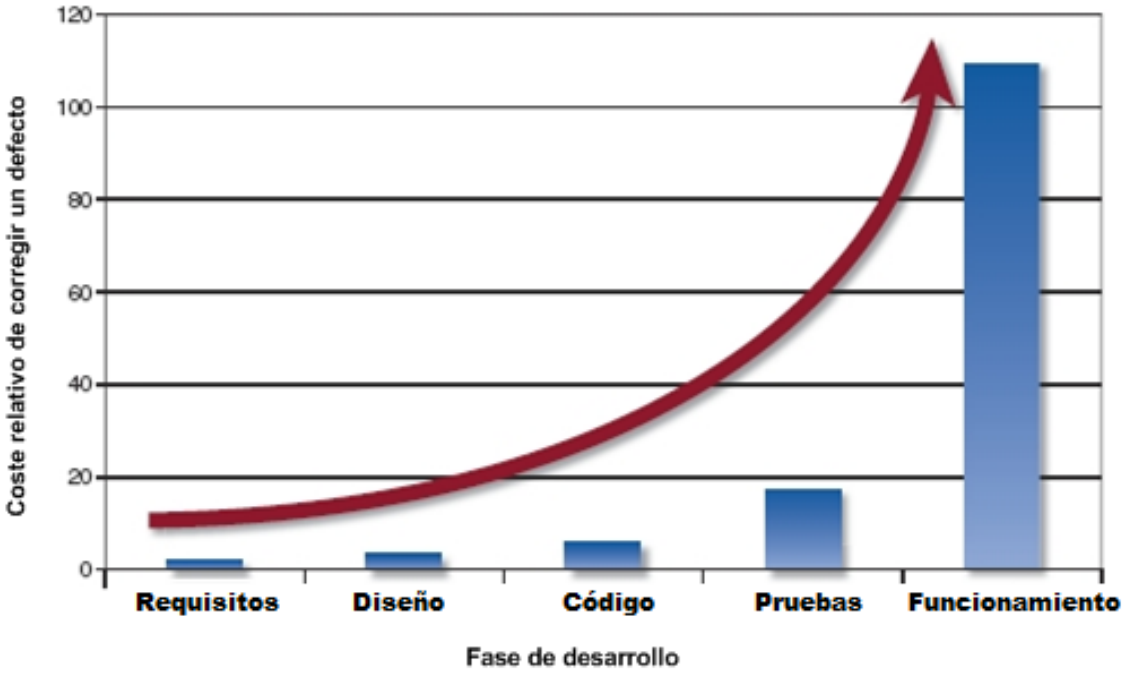


Figura 4.3: Coste de corregir un defecto según la fase de desarrollo

Existen otros datos económicos que justifican el uso de la CI. Barry Boehm, uno de los mayores expertos en calidad del software, ha publicado varios estudios que demuestran cómo el coste de eliminar un defecto en un software crece exponencialmente en cada fase en la que aún no se haya descubierto dicho fallo [Boehm, 2002]. Otros trabajos han confirmado las teorías de Boehm:

- En [Sharpe, 2003] se explica que reparar un defecto durante la etapa de desarrollo de un módulo podría tener un coste aproximado de un dólar. Sin embargo, la reparación costaría más de cien dólares cuando ya se ha integrado todo el

código y miles de dólares si el defecto se detecta cuando el software ya se ha distribuido.

- Un estudio del NIST (National Institute of Standards and Technology o Instituto Nacional de Estándares y Tecnologías) ha revelado que los defectos en el software hacen que los Estados Unidos se gasten 60 billones de dólares cada año [Tassey, 2002]. También el NIST ha reconocido que casi el 80 % del coste total de los proyectos se destina a corregir sus defectos.

4.5. Inconvenientes de la integración continua

Pese a las múltiples ventajas, la CI también tiene inconvenientes:

4.5.1. Degeneración de la arquitectura

[Olsson, 1999] ha detectado una degeneración de la arquitectura debido a que los desarrolladores se centran más en el corto plazo.

4.5.2. Sobrecarga del sistema

Debido a que continuamente se hacen integraciones, el sistema se verá sobrecargado, aunque no es un problema muy importante dada la potencia que actualmente tienen los ordenadores.

4.5.3. Necesidad de un servidor

Generalmente la integración se hace en un ordenador diferente al de los desarrolladores para que todos puedan acceder a él en cada momento. Este hecho provoca la necesidad de tener un servidor dedicado, lo cual es asumible dado el decreciente coste de los ordenadores.

4.5.4. Miedo a subir código erróneo

Algunos desarrolladores sientan miedo por el hecho de tener que subir continuamente código a un repositorio. Pese a todo, siempre será más problemático subir el código completo en una única iteración que hacerlo progresivamente con la práctica de la integración continua.

4.5.5. No todos utilizan correctamente los repositorios

Hay desarrolladores que cargan código con errores en los repositorios haciendo que la construcción falle. Este problema se evita concienciando a los desarrolladores de la necesidad de realizar pruebas.

4.6. Estudio del uso de integración continua

Tras comprobar que las ventajas de la integración continua son muy superiores a los posibles inconvenientes, se mostrarán los resultados de una encuesta realizada [Fleischer, 2009] a profesionales de empresas de desarrollo de software que utilizan la integración continua como una práctica habitual. En dicho trabajo se ha realizado un análisis cuantitativo y cualitativo, utilizando métodos analíticos y estadísticos.

Como se puede observar en la Fig. 4.4, los encuestados tienen perfiles bastante diferentes, siendo los arquitectos e ingenieros de software los más abundantes.

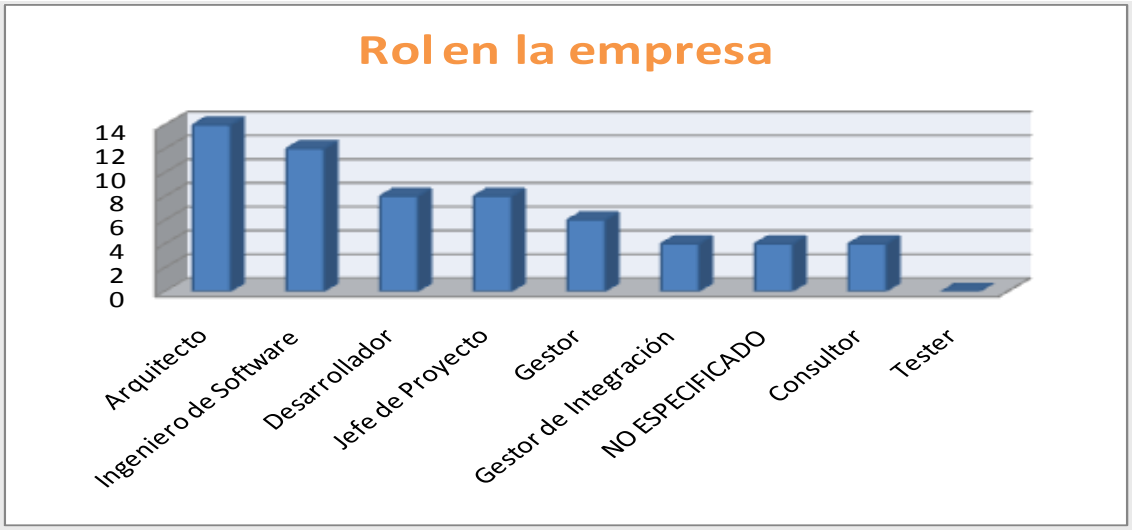


Figura 4.4: Pregunta 1: Rol en la empresa de las personas encuestadas

Aunque el 58 % de encuestados trabajan en Europa (Fig. 4.5), existe una diversidad que garantiza que los datos obtenidos no son dependientes del lugar de trabajo de las personas.

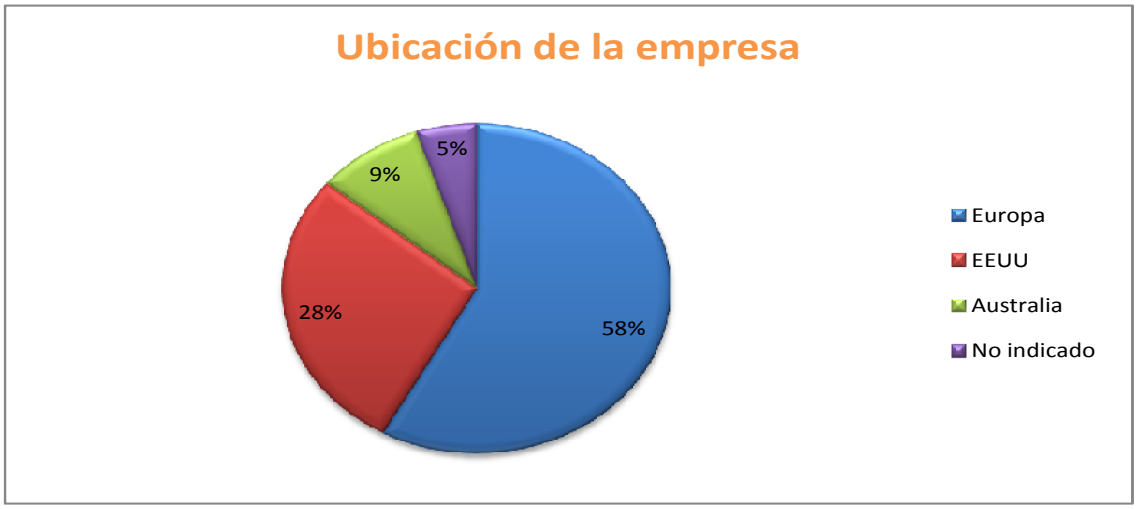


Figura 4.5: Pregunta 2: Ubicación de la empresa

La mayoría de las empresas del análisis son de pequeño tamaño (Fig. 4.6), teniendo menos de 50 empleados dedicados en el departamento de desarrollo. Sin embargo, hay 12 empresas con más de 200 personas en dicho departamento.

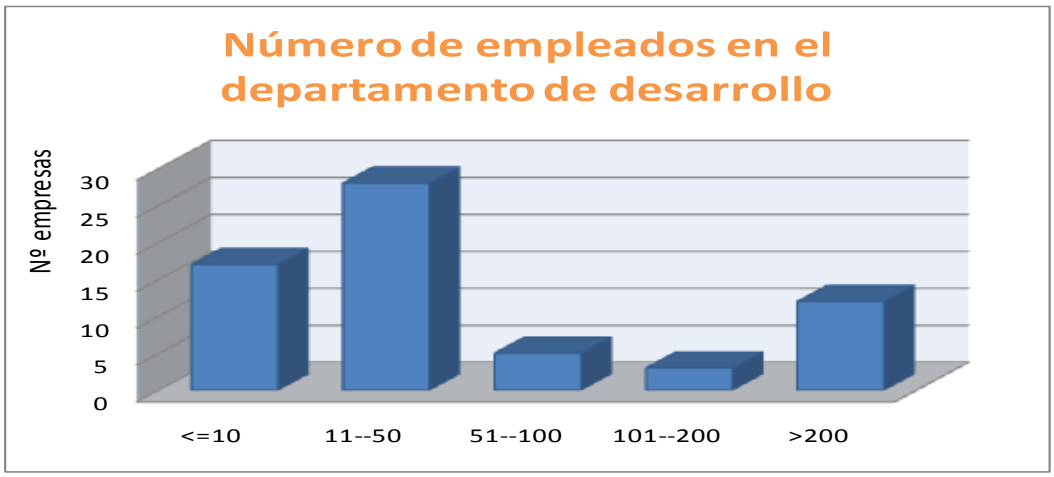


Figura 4.6: Pregunta 3: Número de empleados en el departamento de desarrollo

Generalmente, el tamaño de los equipos de desarrollo está entre 3 y 10 personas (Fig. 4.7).

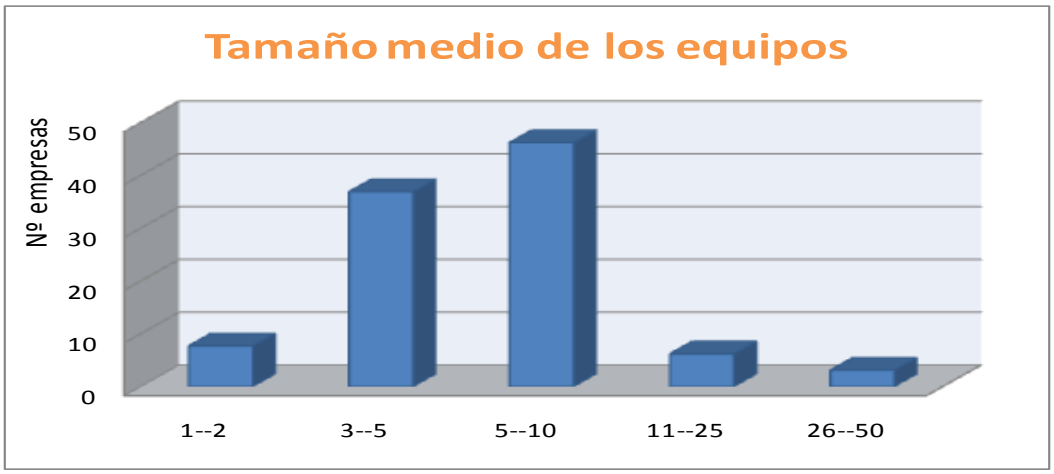


Figura 4.7: Pregunta 4: Tamaño medio de los equipos

Pese a lo que pueda parecer natural, la mayoría de las empresas desarrollan sus productos simultáneamente en más de una ubicación física (Fig. 4.8). Este hecho hace que la coordinación de los equipos, y por tanto la integración, sea más compleja.

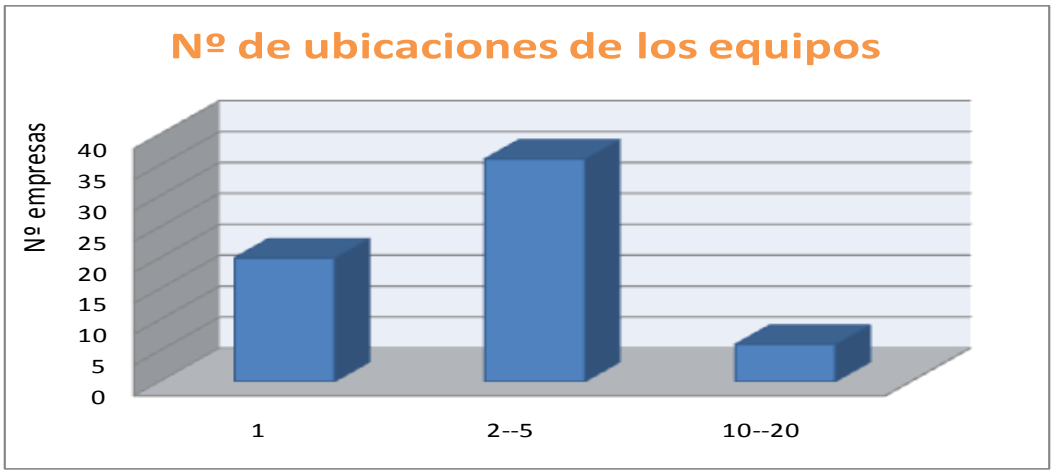


Figura 4.8: Pregunta 5: Número de ubicaciones de los equipos

Como generalmente las empresas son de pequeño tamaño, la cantidad de proyectos que manejan también lo es (Fig. 4.9).

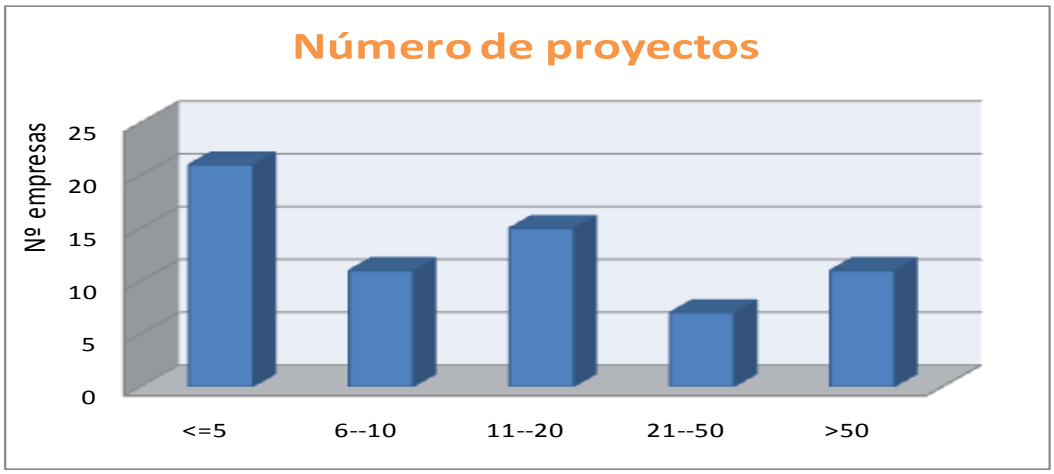


Figura 4.9: Pregunta 6: Número de proyectos

Java, con un 45 %, es la plataforma más utilizada (Fig. 4.10). Sin embargo .NET y C++ también están siendo ampliamente utilizados.

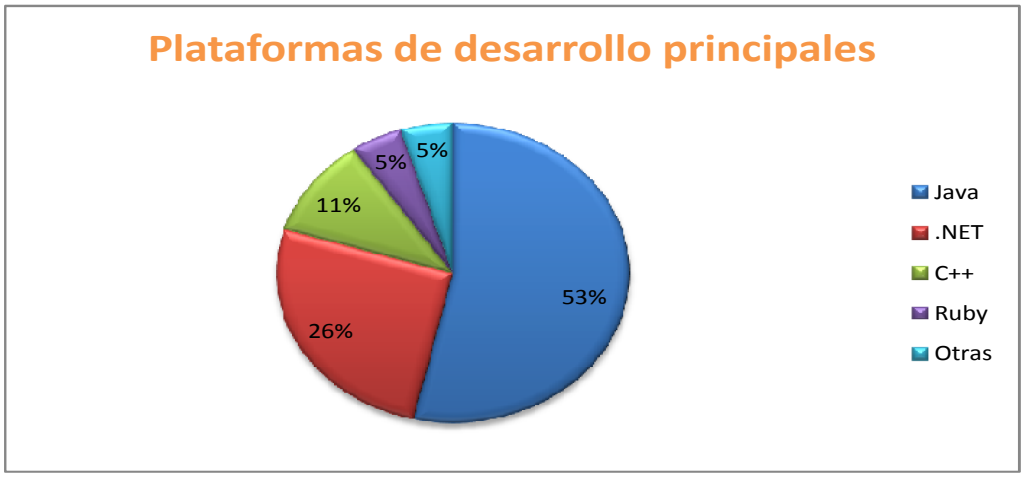


Figura 4.10: Pregunta 7: Plataformas de desarrollo principales

El tipo de software desarrollado es muy heterogéneo (Fig. 4.11). Lo más abundante es el software para la Web, pero también hay muchas aplicaciones para servidores o escritorio.

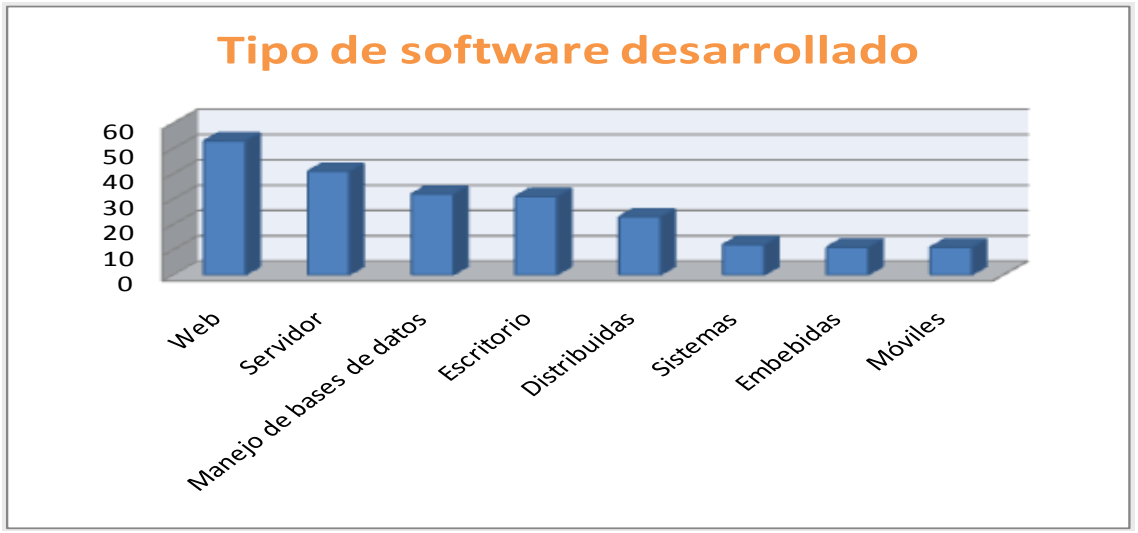


Figura 4.11: Pregunta 8: Tipo de software desarrollado

La herramienta de control de versiones Subversión, con un 60 %, se consolida como más utilizada (Fig. 4.12). CVS le sigue a mucha distancia.

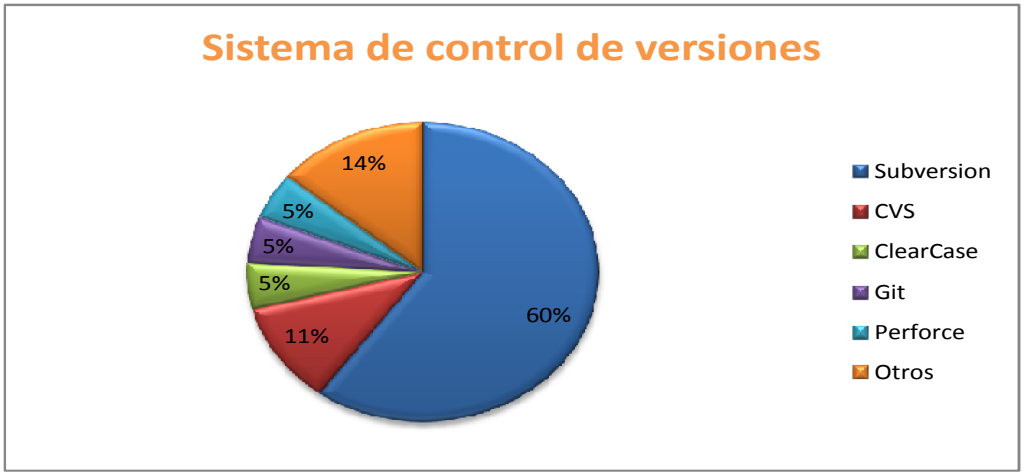


Figura 4.12: Pregunta 9: Sistema de control de versiones utilizado

La mayoría de encuestados utiliza como herramientas de integración continua Cruise Control y Hudson (Fig. 4.13). El resto de herramientas son utilizadas minoritariamente.

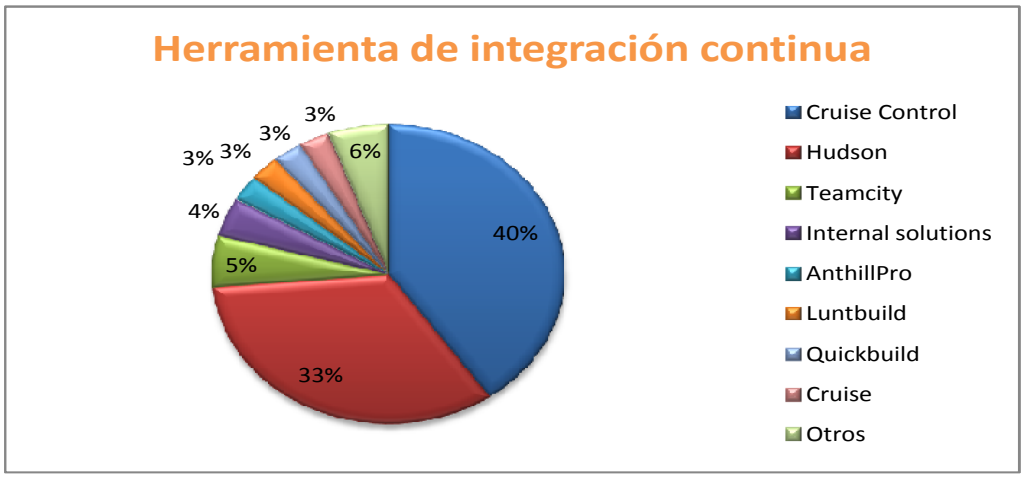


Figura 4.13: Pregunta 10: Herramienta de integración continua utilizada

Todos los encuestados utilizan la práctica de integración continua en sus empresas (Fig. 4.14), que permite que todos tengan criterio para opinar.

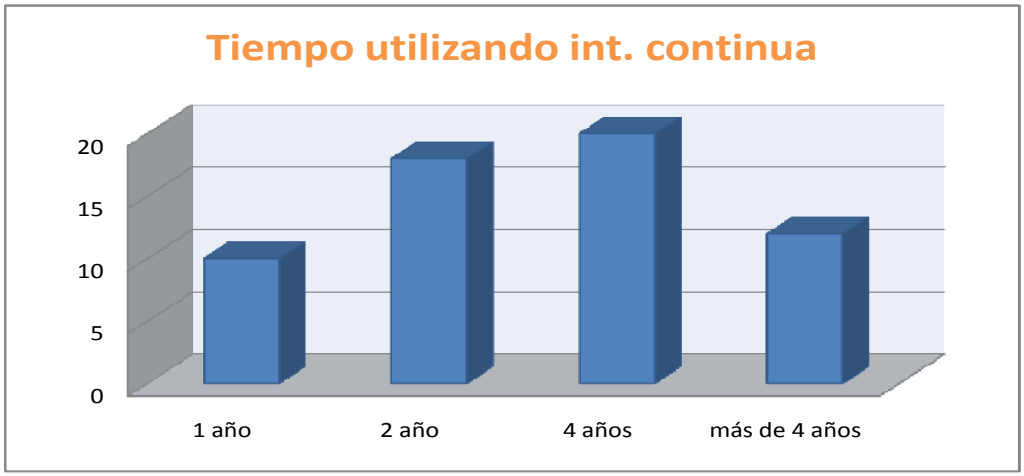


Figura 4.14: Pregunta 11: Tiempo utilizando integración continua

El 60 % de las empresas utiliza siempre integración continua en sus proyectos (Fig. 4.15). Las empresas que no lo utilizan, suele ser porque cuando se realizan proyectos de tamaño muy reducido, los beneficios que obtienen al emplear la práctica son menores.

Figura 4.15: Pregunta 12: ¿Tu empresa utiliza siempre integración continua?

Cabe resaltar que el 89 % de las empresas tiene una máquina exclusiva en la que se hacen las construcciones diferente a la de desarrollo (Fig. 4.16). Es algo normal si se tiene en cuenta que generalmente los proyectos los realizan varias personas

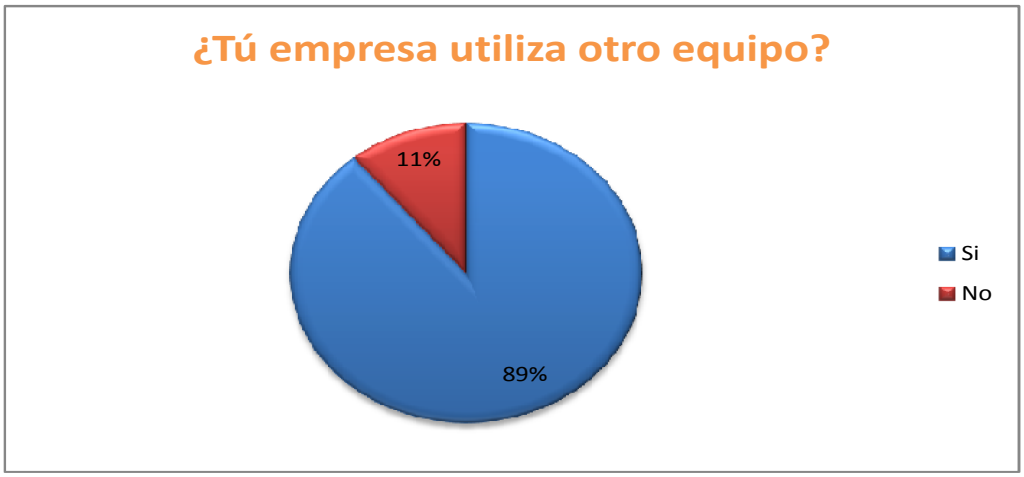


Figura 4.16: Pregunta 13: ¿Tu empresa utiliza otro equipo para la construcción?

Cuando a los encuestados se les preguntó cómo comienza el proceso de integración continua en sus proyectos, la mayoría respondió que justo cuando se hace *check-in* en el repositorio (Fig. 4.17). Otros lo hacen cada cierto tiempo preestablecido y otros manualmente cuando alguien así lo decida.

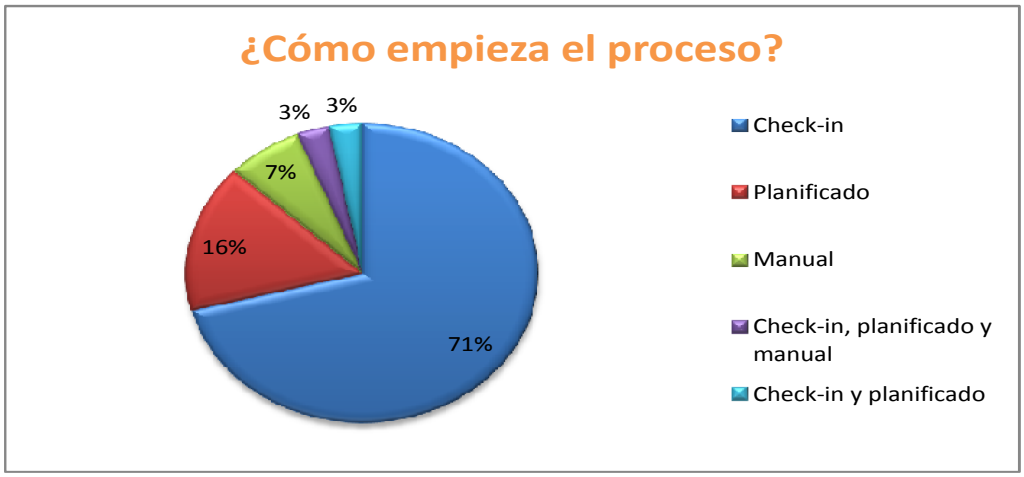


Figura 4.17: Pregunta 14: ¿Cómo empieza el proceso de integración?

Sobre las características más frecuentemente utilizadas durante la integración continua, la mayoría respondió que pruebas automáticas, compilación y análisis de código (Fig. 4.18).

Figura 4.18: Pregunta 15: Características más frecuentemente utilizadas

Para los encuestados hay diversas características que las herramientas de integración continua deberían proporcionar (Fig. 4.19). Algunas de ellas son comprobación del estado del repositorio, Web de control y monitorización con interfaz amigable, mecanismos de retroalimentación o puntos de extensibilidad en la herramienta.

Figura 4.19: Pregunta 16: ¿Cuáles son las características más importantes?

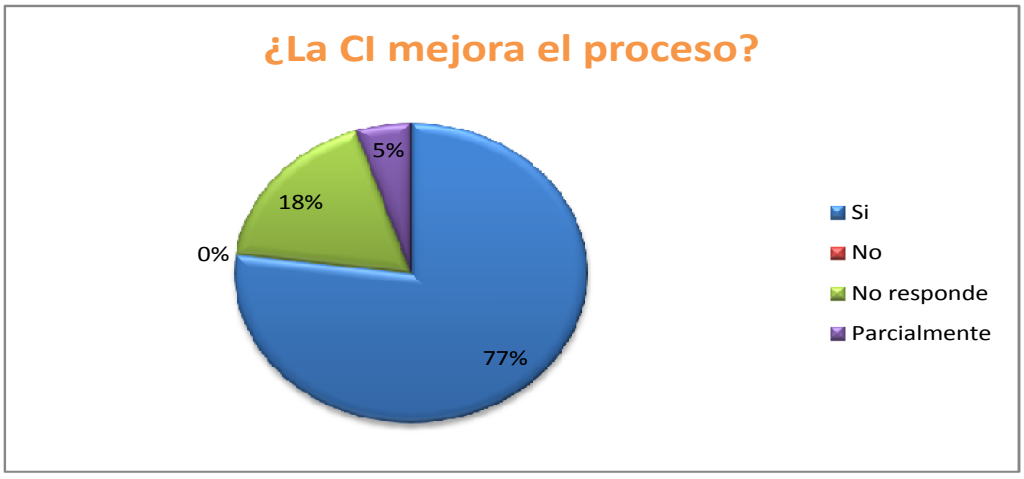


Figura 4.20: Pregunta 17: ¿CI mejora el proceso de desarrollo?

El 77 % de los encuestados dijo que la integración continua ha mejorado significativamente su proceso de desarrollo. El 5 % dice que lo ha hecho parcialmente y el 18 % no ha respondido a la pregunta (Fig. 4.20). El punto importante es que no hay ninguna respuesta negativa sobre este asunto.

Algunos de los encuestados ven como único punto negativo la dificultad que tienen las herramientas para ser configuradas, motivo por el cual sólo utilizan dichas herramientas en proyectos de gran tamaño. Sin embargo, ven como ventajas que el software ha aumentado en calidad y disminuido en tiempo de desarrollo.

Para concluir, se preguntó si utilizarían integración continua desde el principio en los nuevos proyectos. La gran mayoría, un 77 %, ha afirmado que si (Fig. 4.21). Sólo un 1 % ha respondido negativamente a esta pregunta. El único inconveniente volvió a ser el difícil uso y configuración de las herramientas actuales.

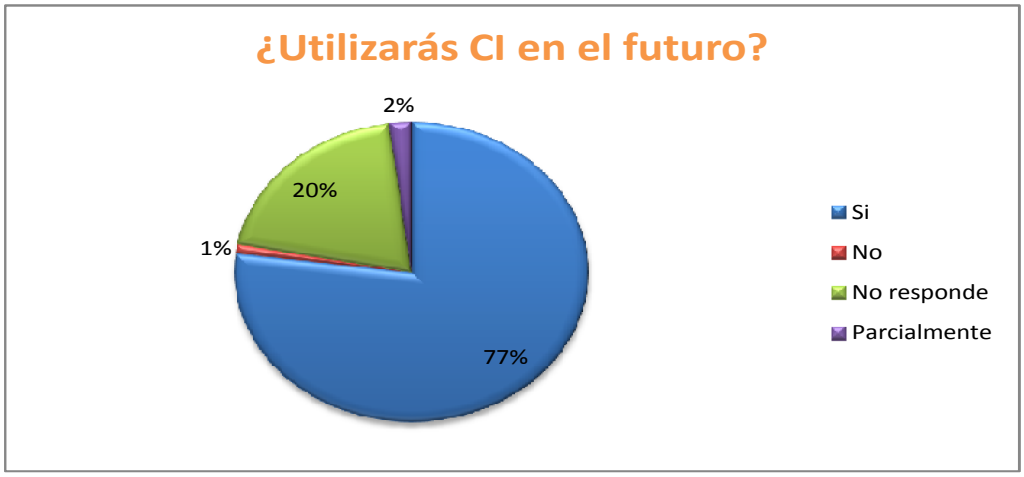


Figura 4.21: Pregunta 18: ¿Usarás CI en el futuro desde el comienzo del proyecto?

4.7. Herramientas

Existe una gran cantidad de herramientas que facilitan la realización de la práctica de la integración continua. Todas ellas están orientadas a lenguajes de programación y sistemas de control de versiones tradicionales. Además, debido a la gran diversidad, es muy complicado elegir cuál es la mejor de todas las herramientas para los propósitos y objetivos citados en la introducción de este trabajo[8], pero se pueden

[8]La elección de una herramienta servirá como base para construir el prototipo. No es excesivamente importante el hecho de elegir una herramienta u otra, aunque es necesario que proporcione capacidades de adaptación para ser extendida

descartar algunas de ellas:

- Soluciones comerciales como las herramientas *Cruise* o *Parabuild*, ya que las soluciones *Open Source* son muy maduras y potentes.
- Con poco soporte como las herramientas *Sin* o *Gump*.
- Sin flexibilidad para añadir nuevas herramientas de construcción de código como *Anthill*.
- Sin soporte para múltiples proyectos como *Buildbot*.

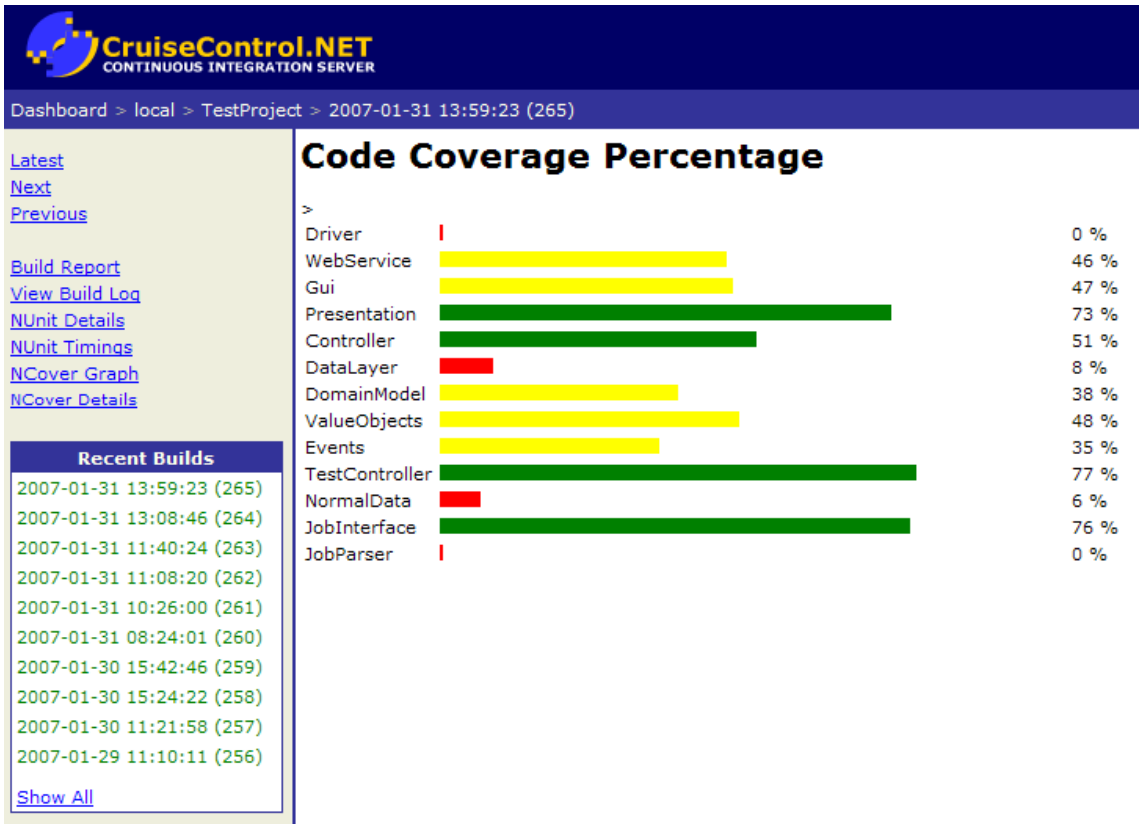


Figura 4.22: Entorno Web de Cruise Control .NET

Dado que las herramientas de CI son principalmente un nexo de unión entre otras herramientas como sistemas de control de versiones y herramientas de construcción de código (véase sección 17.1), una de las características más importantes que pueden tener es la flexibilidad para adaptarse a ellas, ya que la productividad dependerá, sobre todo, de las herramientas externas.

Después de estas consideraciones, quedan varias alternativas como Cruise Control, Hudson, LuntBuild o Continuum. Sin embargo, Cruise Control (Fig. 4.22) en

sus tres versiones (Java, C# y Ruby) es una herramienta más madura y con un mayor soporte por parte de la comunidad *Open Source* que el resto de sus competidores. La Tabla 4.1 muestra una lista con las herramientas más importantes.

Herramienta	**Autor**	**Enlace**
AnthillPro	Urbancode	http::/www.anthillpro.com/
Bamboo	Atlassian	http::/www.atlassian.com/
Buildbot	Buildbot	http://buildbot.net/
Continuum	Apache Project	http::/continuum.apache.org/
Cruise	ThoughtWorks	http::/studios.thoughtworks.com/cruise/
Cruise Control (.NET)	ThoughtWorks	http://cruisecontrol.sourceforge.net/
Cruise Control (Java)	ThoughtWorks	http://cruisecontrol.sf.net/
Cruise Control (Ruby)	ThoughtWorks	http://cruisecontrolrb.thoughtworks.com/
FinalBuilder	VSoft Technologies	http://www.FinalBuilder.com/
Gump	Apache Project	http://jakarta.apache.org/gump/
Hudson	Java.net project	http://hudson.dev.java.net/
Lunt build	Javaforge project	http::/luntbuild.javaforge.com/
Parabuild	Viewtier	http://www.viewtier.com/
Pulse	Zutubi	http::/www.zutubi.com/
Sin	Tigris	http://sin.tigris.org/
Quick build	PMEase	http::/www.pmease.com/
TeamCity	JetBrains	http::/www.jetbrains.com/teamcity/

Tabla 4.1: Herramientas de integración continua

Existen algunas herramientas surgidas de trabajos de investigación como [Pavlo et al., 2006], en el que se describe una herramienta de integración continua para entornos heterogéneos y distribuidos; pero lo cierto es que existen muy pocos trabajos y muy poca información sobre ellas.

4.8. Conclusiones

La práctica de la integración continua lleva varios años ayudando a gestionar mejor proyectos en multitud de organismos debido a que proporciona una gran cantidad

de ventajas. Sin embargo, dada la dificultad de uso de las herramientas actuales, aún no es una práctica conocida y utilizada intensivamente en los desarrollos de software.

Este capítulo cierra la introducción al marco teórico que introduce al resto del trabajo. A partir del siguiente capítulo se comenzará a profundizar en las claves para llevar a cabo los objetivos de esta Tesis.

Bloque III

Elección de una iniciativa MDE

Índice del bloque

CAPÍTULO 5

Arquitectura Dirigida por Modelos

A mediados del año 2000 surgió MDA (Model-Driven Architecture o Arquitectura Dirigida por Modelos) [Miller et al., 2003] con la idea de aumentar el nivel de abstracción con el que se definen y se desarrollan los sistemas software. Para ello, se utilizan modelos como principales artefactos en el proceso de desarrollo. Se puede afirmar que el nacimiento del estándar MDA ha marcado un antes y un después en la evolución de las tecnologías relacionadas con modelos.

En este capítulo se profundizará en el origen del estándar MDA, en sus características y en otras tecnologías y estándares, que son la base y el soporte sobre el que se sustenta MDA.

⁂

5.1. Origen de MDA

Andrew Watson [Watson, 2008] hace un repaso de la historia que rodea a MDA (Model-Driven Architecture o Arquitectura Dirigida por Modelos) [Miller et al., 2003] desde su aparición el 8 de marzo de 2000 como iniciativa de OMG (Object Management Group)[1].

OMG nació a finales de 1980 como una organización independiente y sin ánimo de lucro, que hoy en día está integrada por más de 800 compañías, para intentar dar una solución al problema de la interoperabilidad entre diferentes sistemas mediante tecnologías *middleware*[2].

La primera aproximación de OMG fue pretender crear una serie de estándares orientados a objetos denominados en su conjunto como OMA (Object Management Architecture) con la intención de desarrollar aplicaciones distribuidas en entornos heterogéneos. Gracias a CORBA (Common Object Request Broker Architecture) [Pope, 1998] se consiguió este primer objetivo con un reconocido éxito.

A partir de 1990, OMG empezó a definir estándares de interoperabilidad para diferentes dominios específicos de aplicación, como pueden ser el económico o el sanitario. Para dicho fin se utilizó IDL (Interface Definition Language) [OMG, 1991]. La idea es, mediante CORBA, definir como traducir la información que parte desde un cliente a un lenguaje intermedio común IDL de forma que en el destino se pueda traducir en el sentido inverso, de IDL al lenguaje del destinatario.

Cerca del año 2000, IDL y CORBA ya estaban muy extendidos pero presentaron dos importantes limitaciones:

- Dificultad para describir los aspectos no estructurales de las interfaces.
- Complicación para traducir las interfaces a otras notaciones porque es necesario conocer con bastante profundidad la tecnología CORBA.

La intención de que IDL y CORBA sustituyeran al resto de *middlewares* se vio truncada por la aparición de tecnologías como DCOM (Distributed Component Object Model) [Redmond, 1997], RMI (Remote Method Invocation) [Downing, 1998] o SOAP (Simple Object Access Protocol) [Scribner et al., 2000].

El siguiente salto natural fue pasar de una aproximación basada en *middlewares* a otra basada en una especificación independiente de la plataforma tecnológica con un elevado nivel de abstracción. En este contexto nace MDA, cuyos principales

[1] http://www.omg.org/

[2] Software que conecta varios sistemas para que puedan intercambiar datos entre ellos

objetivos son mejorar la productividad, la portabilidad, la interoperabilidad y la reutilización de los sistemas.

5.2. Modelos en MDA

MDA se basa fundamentalmente en el uso de modelos como forma de adaptarse a los constantes cambios tecnológicos. Además de la definición ya dada (véase sección 3.3), existen otras definiciones reconocidas y válidas de modelos:

- Los modelos son simplificaciones de la realidad [Selic, 2003].
- Los modelos son un conjunto de extractos de un sistema bajo estudio [Seidewitz, 2003].
- Los modelos son un conjunto de elementos formales que describen algo que está siendo desarrollado para un propósito específico y pueden ser analizados utilizando varios métodos [Mellor et al., 2003].

5.2.1. Puntos de vista

La especificación MDA define cuatro puntos de vista o niveles de abstracción para analizar sistemas, de modo que la idea principal es empezar desde el nivel de abstracción más alto e ir haciendo transformaciones automática o semi-automáticamente hasta llegar a código ejecutable por una máquina física o virtual.

Para cada punto de vista o nivel de abstracción se define un modelo del sistema (Fig. 5.1).

- CIM (Computational-Independent Model o Modelo Independiente de Computación). Idealmente se desarrolla software sin preocuparse del sistema operativo o hardware sobre el que funciona; sin embargo en un mundo tan heterogéneo eso es imposible. El primer punto de vista sirve para mostrar detalles de la estructura del sistema a desarrollar sin tener en cuenta aspectos informáticos. El CIM se corresponde con la captura de requisitos o modelo de dominio de la aplicación.
- PIM (Platform-Independent Model o Modelo Independiente de Plataforma). Cuando se realiza la transformación de CIM a PIM, se baja el nivel de abstracción de modo que se tiene un modelo que está pensado para ejecutarse en un ordenador pero sin tener en cuenta la plataforma, es decir, como si fuera a ejecutarse en una máquina virtual independiente.

- PSM (Platform-Specific Model o Modelo Específico de Plataforma). Tras la transformación de PIM a uno o varios PSMs se baja aún más el nivel de abstracción y se obtiene un modelo para una o varias plataformas específicas (por ejemplo .NET o Java).

- ISM (Implementation-Specific Model o Modelo Específico de la Implementación). Es el nivel de abstracción más bajo que se obtiene al transformar los PSMs generados a partir del PIM en ISMs. Típicamente cada PSM se corresponderá con un ISM, es decir, el código fuente específico del sistema para un entorno determinado.

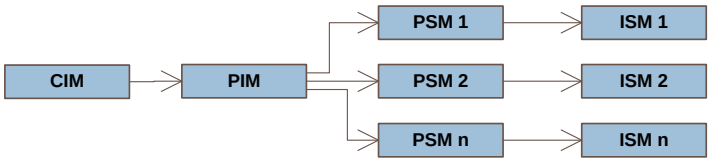


Figura 5.1: Modelos en MDA

5.2.2. Transformaciones entre modelos

Para generar artefactos a partir del CIM (el punto inicial), se realizan transformaciones entre los diferentes modelos MDA. Dichas transformaciones son necesarias para pasar del modelo CIM a PIM, del PIM a PSM y del PSM a ISM. Existen diferentes tipos de transformaciones:

- *Refinamientos*. Permiten construir un modelo más específico a partir de otro más abstracto. Pueden ser *horizontales* como por ejemplo de PIM a PIM o de PSM a PSM o pueden ser *verticales* como por ejemplo de PIM a PSM y de PSM a ISM.

- *Anti-refinamientos*. Permiten construir un modelo más abstracto a partir de otro más específico. Por ejemplo pasar del ISM al PSM. Este tipo de transformación es muy complicada y en ocasiones imposible. Muchos trabajos como [Völter and Stahl, 2006] dudan de su utilidad.

- *Refactorings*. Transforma un modelo en otro modelo del mismo nivel, mejorando su calidad pero sin cambiar su nivel de abstracción.

- *Puentes (bridges)*. Los puentes son transformaciones entre diferentes PSMs, por ejemplo entre un PSM destinado a .NET y otro destinado a la plataforma Java. Algunos trabajos como [Bézivin et al., 2005b] y [Bézivin et al., 2005a] están centrados en investigaciones sobre este aspecto.

Las transformaciones (Fig. 5.2) se basan en el hecho de que tanto el metamodelo de origen como el metamodelo de destino están basados en el mismo meta-metamodelo.

Para especificar los diferentes *mapas de transformaciones* entre modelos, OMG recomienda el uso de MOF QVT (Query/View/Transformation) [OMG, 2005c], que sirve para describir la correspondencia entre patrones de elementos en el modelo origen y los elementos a ser creados en el modelo destino. Basándose en QVT, existe un lenguaje más ampliamente difundido denominado ATL (ATLAS Transformation Language) [Jouault et al., 2006], que puede utilizarse para guiar las transformaciones como en el caso de [Mora et al., 2007].

Una vez especificadas las transformaciones entre modelos, el siguiente paso es realizar tales transformaciones. Se pueden realizar manualmente, automáticamente o mediante una combinación de ambas. También es importante el número de transformaciones, ya que hay quienes transforman todos los modelos que indica la especificación de MDA para ayudar a entender y depurar el código o hay quienes transforman el modelo PIM directamente al ISM.

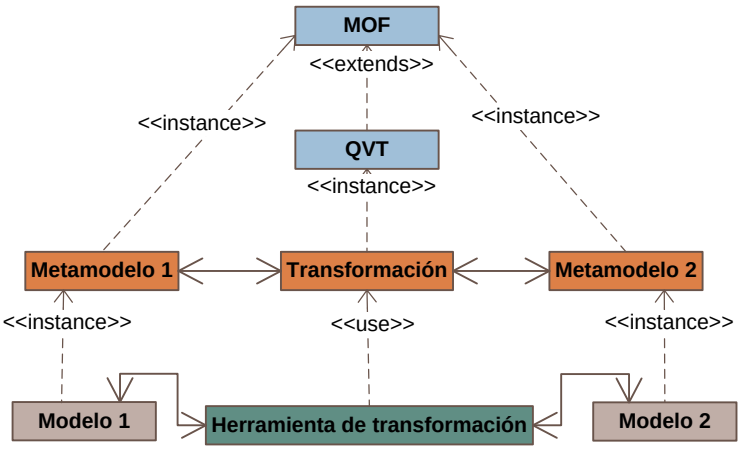


Figura 5.2: Transformaciones de modelos en MDA

5.3. Arquitectura en capas

La idea de la necesidad de un metamodelo no es reciente. En [Kotteman and Konsynski, 1984] se muestra que al menos son necesarios cuatro niveles de instanciación para integrar el modelado en la evolución de los sistemas.

MDA está basado en la arquitectura en capas definida por OMG (debido a que así se consigue una mayor capacidad de integración y de extensión de los modelos) y algunos estándares complementarios (Fig. 5.3 y Fig. 5.4):

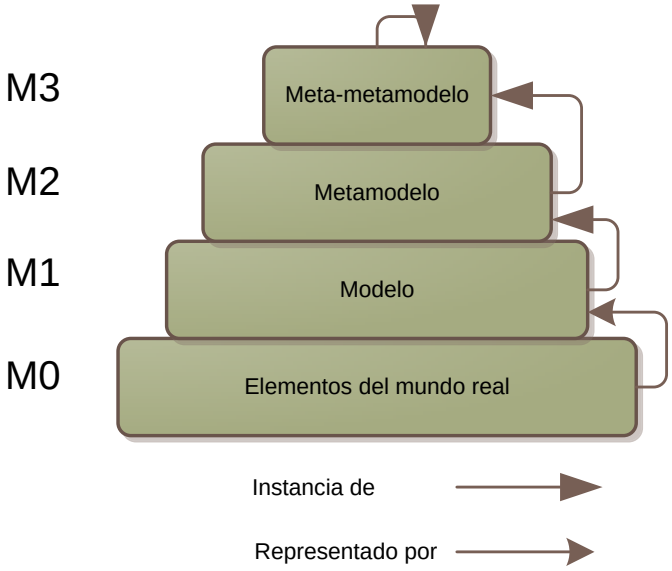


Figura 5.3: Arquitectura de cuatro capas

- El nivel M3 (Meta-metamodelo). El nivel M3 define los conceptos, las características y las relaciones de los elementos del nivel M2, mientras que los elementos situados en el nivel M2 son las instancias de los elementos del nivel M3. En este nivel, OMG ha definido un lenguaje que sirve para describir todos los elementos de nivel M2, MOF (Meta-Object Facility) [OMG, 2005a], cuya característica más destacada es que se ha utilizado para auto definirse a sí mismo, ya que de lo contrario no se podría cerrar el ciclo.

- El nivel M2 (Metamodelo). En este nivel se definen los elementos de los modelos del nivel M1. En el caso de un modelo UML [OMG, 2007b] se pueden citar ejemplos de conceptos que intervienen en este nivel como 'Clase', 'Atributo',

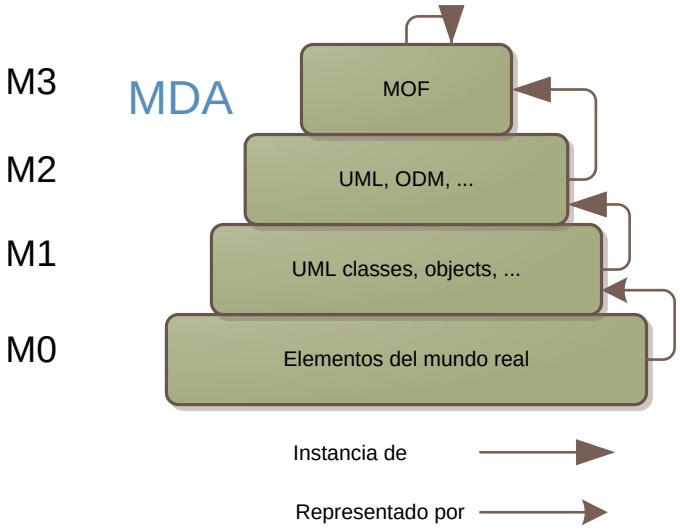


Figura 5.4: Arquitectura de cuatro capas MDA

o 'Asociación'. El nivel M2 define los elementos válidos en un determinado modelo del nivel M1, mientras que los elementos situados en el nivel M1 son las instancias de los elementos del nivel M2.

- El nivel M1 (Modelo). En este nivel se definen por ejemplo los conceptos 'Cliente', 'Compra' y 'Libro' y sus atributos 'Dirección', 'Nombre', 'Número', 'Título', etc. El nivel M1 define las clasificaciones de los elementos del nivel M0, mientras que los elementos situados en el nivel M0 son las instancias de los elementos del nivel M1. Un ejemplo en este nivel sería un modelo de clases o de casos de uso.

- El nivel M0 (Realidad). Existen dos aproximaciones diferentes para describir este nivel. La más extendida es que en este nivel hay instancias de los modelos de nivel M1 como por ejemplo objetos 'Coche' instanciados en un lenguaje de programación. La otra aproximación, más novedosa, dice que los objetos 'Coche' no son instancias, sino que son elementos, coches en este caso, del mundo real [Atkinson and Kühne, 2003].

Con la aproximación MDA no se indica explícitamente la necesidad de utilizar ninguna tecnología concreta pero existen estándares tecnológicos definidos por OMG que son la base para el surgimiento de MDA. Algunos son los ya mencionados UML, MOF, QVT. También existen otros como OCL (Object Constraint Langua-

ge) [OMG, 2003b] y XMI (XML Metadata Interchange) [OMG, 2005b]. Todos estos estándares se comentarán durante las siguientes secciones del capítulo.

5.4. Espacios de modelado

Cualquier elemento puede ser tomada como un modelo si es una abstracción del mundo real. Por ejemplo, una maqueta de un hotel representa a un hotel, por lo tanto la maqueta es un modelo. Si se considera la arquitectura en capas definida por OMG, los elementos utilizados para crear la maqueta como por ejemplo madera o pegamento serían el metamodelo y los elementos utilizados para crear la madera o el pegamento formarían el meta-metamodelo. En este caso, el meta-metamodelo también es llamado *super-metamodelo* [Gasevic et al., 2006] porque es la capa más alta de la arquitectura, pero otras arquitecturas podrían tener un número diferente de capas.

De hecho, inicialmente UML era un super-metamodelo pero para poder crear otros metamodelos compatibles entre sí, hubo que situar MOF por encima de UML. También hay que tener en cuenta que la propia maqueta sería un elemento del mundo real puesto que se puede tocar, pero de todos modos su rol en la arquitectura de cuatro capas de este ejemplo sería la de modelo de un hotel (su rol depende del contexto).

Se dice que un modelo *representa* cosas del mundo real porque actúan en su nombre y se dice que un modelo *conforma con* su metamodelo porque es quien define como puede ser el modelo.

Un MS (Modeling Space o Espacio de Modelado) es cualquier arquitectura de modelado basado en un super-metamodelo[3]. En la Fig. 5.5 se muestran únicamente dos ejemplos, ya que habría tantos ejemplos como se puedan imaginar.

El caso más típico es el MOF MS en el que se utiliza MOF como super-metamodelo. Debajo de MOF estarían, entre otros, UML y ODM (Ontology Definition Metamodel) [OMG, 2005d], y por debajo de ellos estarían respectivamente los modelos UML o los modelos ODM.

Otro ejemplo, diferente al del estándar, podría ser el de utilizar EBNF (Extended Backus Naur Form) [Essalmi and Ayed, 2006] como super-metamodelo (EBNF MS) para definir gramáticas libres de contexto. Debajo de EBNF estarían los diferentes lenguajes como Java, Visual Basic, C, C++, C# o XML, y en la capa M1 estarían los programas software que serían los modelos de la realidad que se representa en

[3]Los super-metamodelos siempre se definen a sí mismos porque de lo contrario no serían super-metamodelos, habría otra capa por encima

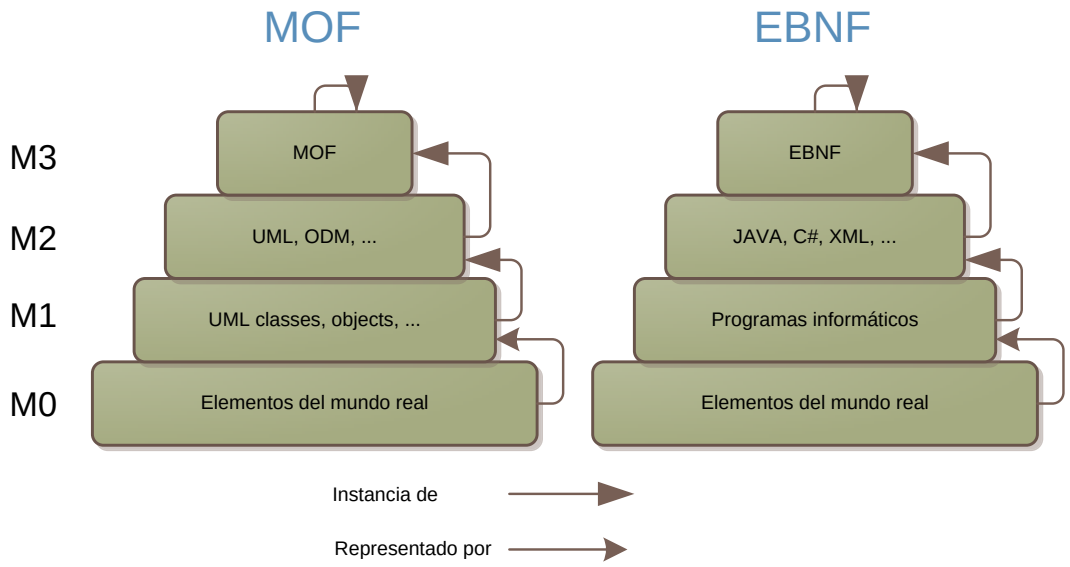


Figura 5.5: Espacios de modelado

ese particular espacio de modelado.

Hay que tener en cuenta que un elemento que es un modelo en un espacio de modelado, puede ser la realidad en otro espacio de modelado. Por lo tanto, existe una dualidad que se representa en la Fig. 5.6, en la que se puede observar que elementos que forman parte de la realidad desde el punto de vista del MOF MS, están en otras capas desde el punto de vista del EBNF MS.

5.5. Espacios técnicos

Los espacios de modelado son un concepto que está inspirado en el concepto de TS (Technical Space o Espacio Técnico) [Kurtev et al., 2002]. Se puede definir un TS como un contexto de trabajo que incluye varios MS relacionados. Por ejemplo, un TS podría ser el MDA TS, el cual incluye al MOF MS. Pero también incluye parcialmente a otros MS como por ejemplo al EBNF MS para dar soporte en la construcción de archivos XMI que sirven para serializar los modelos.

Resumiendo, un espacio técnico es una agrupación de espacios de modelado que tienen algo en común o que necesitan interactuar entre ellos mediante el uso de *bridges*, es decir, un TS incluye uno o más MS y cada MS es parte de uno o más TS.

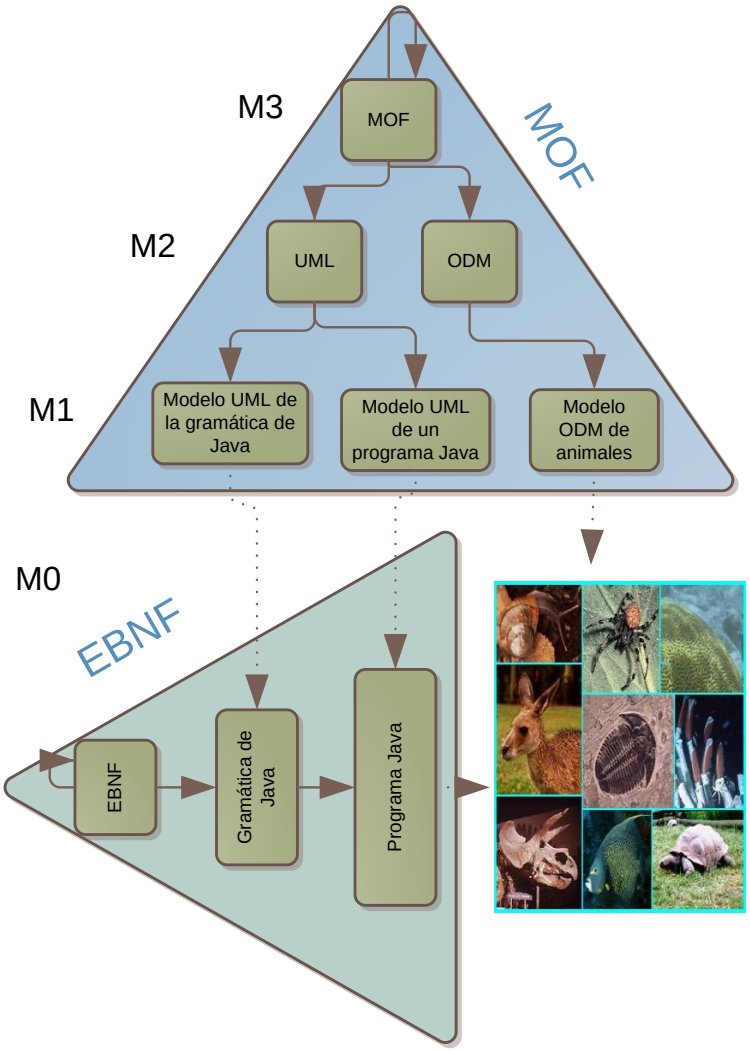


Figura 5.6: Dualidad de los espacios de modelado (ejemplo)

5.6. Lenguaje Unificado de Modelado

UML (Unified Modeling Language o Lenguaje Unificado de Modelado) es una especificación de OMG que comenzó a desarrollarse a principios de la década de los 90 para unificar los enfoques más importantes de la época para modelar sistemas orientados a objetos basándose en diagramas (Grady Booch, James Rumbaugh e Ivar Jacobson [Booch et al., 1999]). La primera versión oficial del estándar apareció en 1997.

UML se ha convertido en un estándar de facto para modelar sistemas software pero en la actualidad se utiliza incluso para modelar aspectos que nada tienen que ver con el software. Analizando su nombre se puede comprender qué es UML:

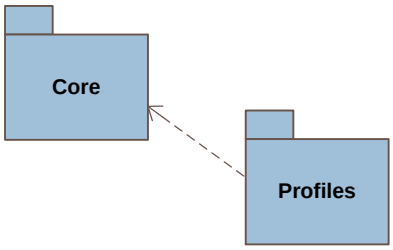


Figura 5.7: Infraestructura de UML

- *Unified*. Como se ha dicho al comienzo, UML unifica los enfoques de modelado más importantes en uno.
- *Modeling*. UML permite modelar aspectos del mundo real mediante el uso de diagramas.
- *Language*. UML es un lenguaje que cuenta con una sintaxis y una semántica bien definida.

Las primeras versiones tenían dos características que han sido muy criticadas, una es que el lenguaje era poco extensible y la otra es que no se permitía la validación y ejecución de modelos UML. Centrándose en esos dos objetivos, ha aparecido la última versión oficial de UML, UML 2.0 que está formada por cuatro especificaciones que se detallarán a continuación:

5.6.1. UML 2.0 Infrastructure

UML 2.0 Infrastructure [OMG, 2007b] es una especificación que contiene los conceptos básicos del metamodelo, proporcionando además mecanismos de extensión del lenguaje. En la Fig. 5.7 puede verse un esquema muy básico de la infraestructura de UML.

El paquete *Core* es utilizado tanto por el resto de UML como por MOF. Es decir, el núcleo de MOF se define utilizando la infraestructura de UML y a la vez UML se define utilizando MOF. Tanto MOF como los perfiles o *profiles* se verán más adelante. La Fig. 5.8 muestra más detalle sobre el paquete *Core*

5.6.2. UML 2.0 Superstructure

UML 2.0 Superstructure [OMG, 2007c] se encarga de los elementos a nivel de usuario, siendo una especificación cuyo soporte es UML 2.0 Infrastructure. En UML

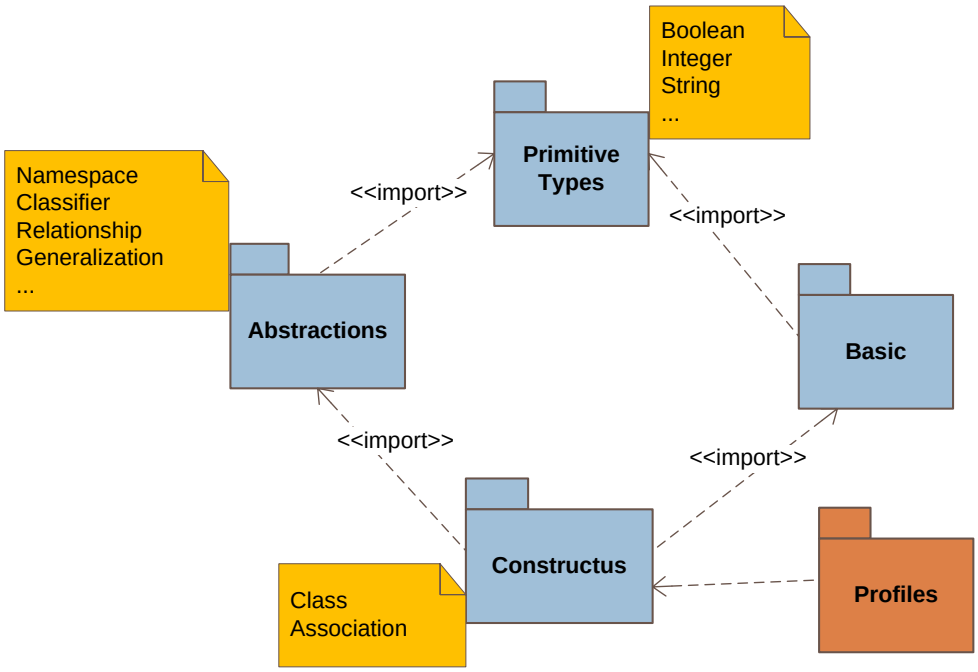


Figura 5.8: Infraestructura de UML (core)

existen tres tipos básicos de bloques de construcción: elementos, conectores y diagramas. Los elementos son las unidades básicas de construcción, los conectores sirven para unir elementos y los diagramas se encargan de agrupar colecciones de elementos y de conectores. En la Fig. 5.9 puede observarse la taxonomía de diagramas [Fowler, 2004] que existe en UML 2.0:

Diagrama de clases

Representa un modelo estático que captura la estructura lógica de un sistema principalmente mediante las clases que lo forman. No se especifica cómo se hacen las cosas, sino que se especifican las clases que hay, los atributos que tienen y las operaciones que realizan. Es un diagrama muy útil para mostrar la relación que hay entre las clases y las interfaces de un sistema. Con las generalizaciones se muestra la herencia, con las agregaciones se muestra la composición o el uso y con las asociaciones se muestra las conexiones. Está considerado un diagrama de prioridad alta.

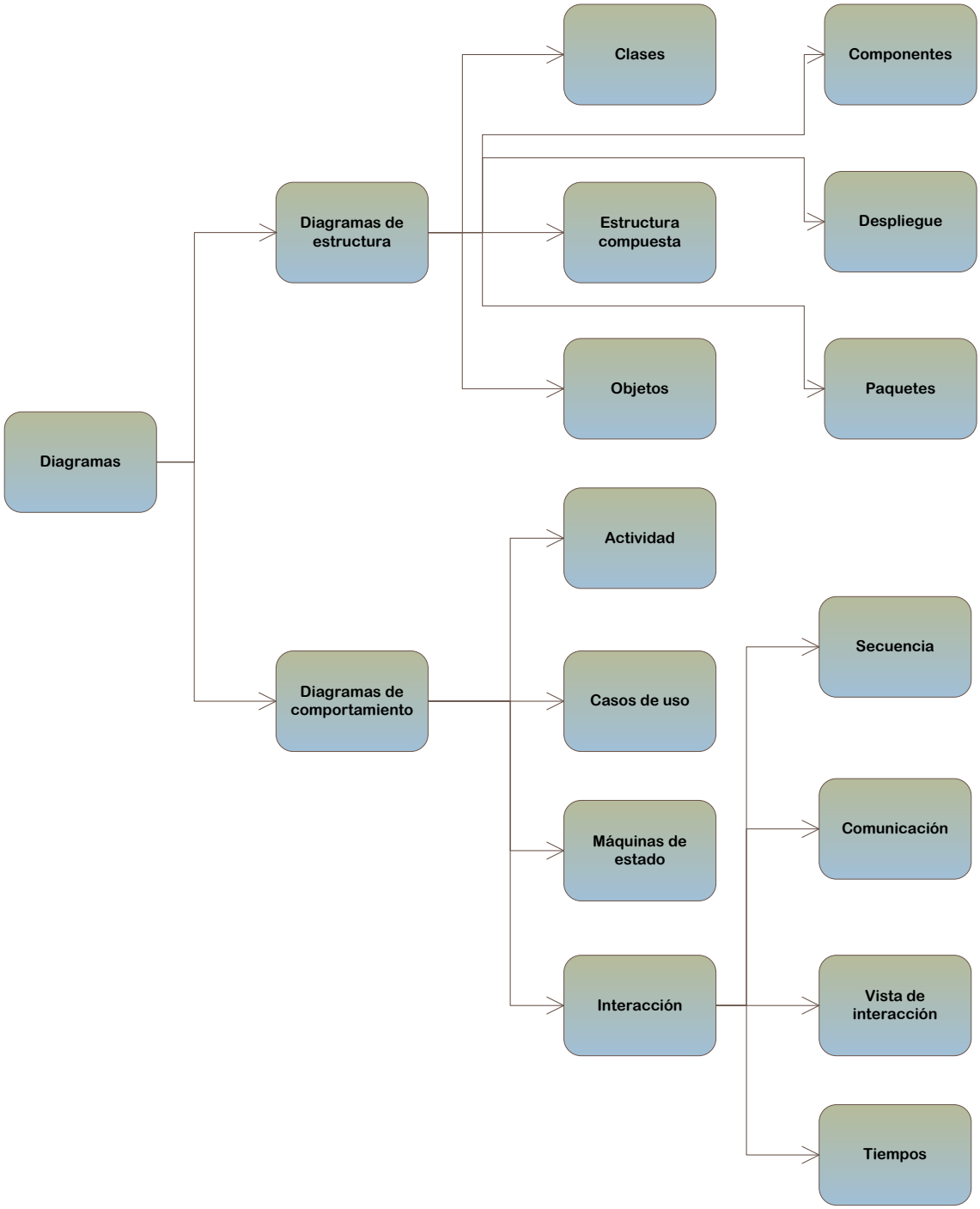


Figura 5.9: Diagramas de UML 2.0

Diagrama de componentes

Representa los diferentes componentes software de los que está formado el sistema con su estructura y dependencias. De ahí se deduce que tiene un nivel de abstracción aún más alto que el diagrama de clases, ya que por lo general un componente estará formado por más de una clase. Está considerado un diagrama de prioridad media.

Diagrama de estructura compuesta

Representa la estructura interna de una clase, es decir, muestra un conjunto de elementos interconectados que colaboran en tiempo de ejecución para conseguir un propósito determinado. Se utilizan *partes* que interactúan a través de *puertos*. Es un diagrama nuevo en la versión 2.0 de UML y está considerado un diagrama de prioridad baja.

Diagrama de despliegue

Representa la distribución de componentes en una aplicación. En muchas ocasiones se mezcla la configuración hardware con la configuración software del sistema y con las asociaciones que hay entre todos los componentes. Está considerado un diagrama de prioridad media.

Diagrama de objetos

Representa una instantánea del sistema en un momento determinado. Generalmente se centra en un conjunto determinado de objetos con los valores de sus atributos y los enlaces que hay entre los objetos. En ocasiones se utilizan varios diagramas de objetos para representar como se quiere que el sistema vaya evolucionando a través del tiempo. Muchas veces se utilizan para proporcionar ejemplos y para realizar casos de prueba del diagrama de clases. Está considerado un diagrama de prioridad baja.

Diagrama de paquetes

Representa cómo un sistema se descompone en agrupaciones lógicas y cómo las agrupaciones se relacionan entre sí. Generalmente se busca que entre los diferentes paquetes haya el mínimo acoplamiento posible para facilitar el mantenimiento de las aplicaciones. Está considerado un diagrama de prioridad baja.

Diagrama de actividades

Representa las actividades y las acciones que describen los diferentes flujos de trabajo que hay en las aplicaciones. Está considerado un diagrama de prioridad alta.

Diagrama de casos de uso

Representa la funcionalidad que ofrece el sistema y los actores que pueden utilizarla. También representa relaciones entre funcionalidades y relaciones entre actores. Está considerado un diagrama de prioridad media.

Diagrama de máquinas de estado

Representa los diferentes estados por los que pueden pasar los elementos de un sistema. Hay que tener en cuenta que el estado de un componente no sólo depende de una determinada entrada, sino que también depende del estado que tuviera anteriormente. Está considerado un diagrama de prioridad media.

Diagrama de secuencia

Representa la colaboración que se produce entre diferentes objetos mediante una secuenciación de mensajes para un determinado caso de uso. Está considerado un diagrama de prioridad alta.

Diagrama de comunicación

Representa, al igual que el diagrama de secuencia, la comunicación en tiempo de ejecución entre objetos. Sin embargo, el diagrama de comunicación está más enfocado a visualizar la colaboración que el momento temporal. Antes de la versión 2.0 de UML, este diagrama se llamaba diagrama de colaboración. Está considerado un diagrama de prioridad baja.

Diagrama de vista de interacción

Representa la cooperación existente entre otros diagramas de interacción para mostrar el flujo de control. Es una variación del diagrama de actividad, siendo un diagrama nuevo en la versión 2.0 de UML. Está considerado un diagrama de prioridad baja.

Diagrama de tiempos

Representa el comportamiento de los diferentes objetos dentro de una escala temporal, es decir, muestra una representación visual de los cambios de estado de

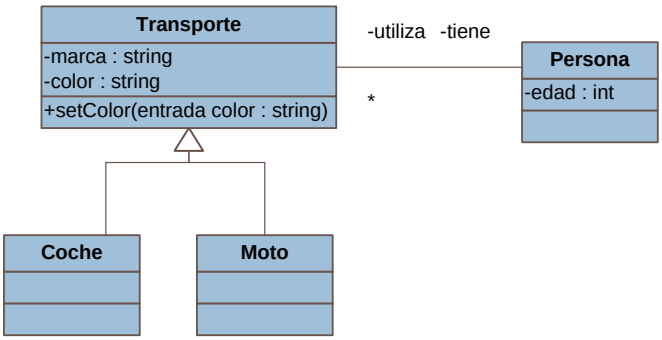


Figura 5.10: Diagrama UML (ejemplo)

los objetos a través del tiempo. Es un diagrama nuevo en la versión 2.0 de UML. Está considerado un diagrama de prioridad baja.

5.6.3. UML 2.0 Object Constraint Language

UML 2.0 OCL (Object Constraint Language) [OMG, 2003b] es un lenguaje formal, declarativo, utilizado para especificar restricciones semánticas no ambiguas en los elementos de los modelos UML. Hay que destacar que OCL puede utilizarse sin ir necesariamente ligado a UML y que puede emplearse en niveles diferentes al M1. Básicamente lo que se hace con OCL es *navegar* por los elementos de un diagrama.

En la Fig. 5.10 puede verse un sencillo diagrama UML, que junto con los dos fragmentos de código que se muestran a continuación (Cod. 5.1 y 5.2), dan lugar a un ejemplo de OCL que cubre el uso de los diferentes tipos de expresiones que se pueden emplear: invariantes, precondiciones y postcondiciones.

```
//*************************************************************************
/*Quien utiliza un medio de transporte tiene que ser mayor de edad*/
Context Transporte
inv: self.tiene.edad >= 18

/*Quien utiliza una moto tiene que tener más de 20 años*/
Context Moto
inv: self.tiene.edad >= 20

/*No hay nadie que pueda utilizar más de 4 medios de transporte diferentes*/
Context Persona
inv: self.utiliza->size <= 4

/*No hay nadie que tenga dos medios de transporte BMW*/
Context Persona
inv: self.utiliza->select(t|t.marca = "BMW")->size <= 1

/*Todos los medios de transporte de una persona son de la marca SEAT*/
Context Persona
inv: self.utiliza->forAll(t|t.marca = "SEAT")
```

```
/*Existe un coche de la marca MERCEDES*/
Context Coche
inv: Coche.allInstances()->exists(t|t.marca = "MERCEDES")
//*****************************************************************************
```

Código fuente 5.1: Archivo `OCL-ejemplo-inv.cs`

```
//*****************************************************************************
/*Para cambiar el color de una moto sólo si estaba pintada de blanco*/
Context Moto::setColor(color:string)
pre: color = "blanco"
post: self.color = color

/*Para cambiar el color de un coche siempre*/
Context Coche::setColor(color:string)
pre: None
post: self.color = color
//*****************************************************************************
```

Código fuente 5.2: Archivo `OCL-ejemplo-pre-pos.cs`

5.6.4. UML 2.0 Diagram Interchange

UML 2.0 Diagram Interchange [OMG, 2003a] es una extensión de XMI (véase sección 5.9) para incluir información gráfica de los modelos UML. Facilita, entre otras cosas, la representación gráfica de diagramas mediante el formato SVG (Scalable Vector Graphics).

5.7. Perfiles UML

Los perfiles UML (*profiles*) son el mecanismo estándar, exclusivo de UML, para extender y configurar los elementos básicos de UML. Existen tres formas de extensión a través de los perfiles UML:

- Estereotipos (*Stereotypes*). Nuevos elementos de construcción.
- Valores etiquetados (*Tagged Values*). Atributos propios de un estereotipo.
- Restricciones (*Constraints*). Anotaciones semánticas que se hacen a determinados elementos del modelo mediante el lenguaje OCL.

Se puede definir un perfil UML como una extensión del metamodelo original UML en el que nuevos conceptos aparecen a partir de otros ya definidos previamente, como lo son *class*, *operation*, *attribute* o *association*. De ese modo, se pueden crear nuevos modelos bien formados, diferentes a los que UML ofrece por defecto.

5.8. Meta-Object Facility

OMG ha especificado MOF [OMG, 2005a], el núcleo central de MDA, que es un lenguaje de metamodelado[4] orientado a objetos mediante el cual se pueden definir los componentes y la estructura de los metamodelos, por ejemplo UML. Como características de MOF se podrían citar las siguientes:

- Está capacitado para ser utilizado con cualquier tipo y paradigma de modelo que se desee.
- Se pueden añadir nuevos tipos de metamodelos en cualquier momento.
- Proporciona los mecanismos para poder describir todos los tipos de sistemas de información.
- Permite relacionar diferentes tipos de metadatos.
- Para definir el lenguaje MOF se utiliza el mismo MOF porque al estar en el nivel más alto de la arquitectura de cuatro capas, se considera que no hay nada más que pueda definirlo.
- MOF se describe con la sintaxis concreta de UML.

Elementos de modelado de MOF

Aunque hay muchos más, los elementos de modelado básicos de MOF[5] son:

- *Class*. Sirve para modelar conceptos que serán entidades en los metamodelos. Por ejemplo las clases, atributos o asociaciones de UML, o incluso conceptos del propio MOF como clases o propiedades.
- *Association*. Sirve para modelar relaciones binarias entre elementos.
- *Package*. Sirve para agrupar conceptos similares.
- *DataType*. Representa a los tipos de datos primitivos (String, Integer, etc.).

[4]También se dice que es un lenguaje de meta-metamodelado porque sirve para definir lenguajes de metamodelado

[5]MOF utiliza como elementos de modelado algunos de los elementos de UML. Sin embargo, dichos elementos tienen un valor semántico diferente porque están en otra capa

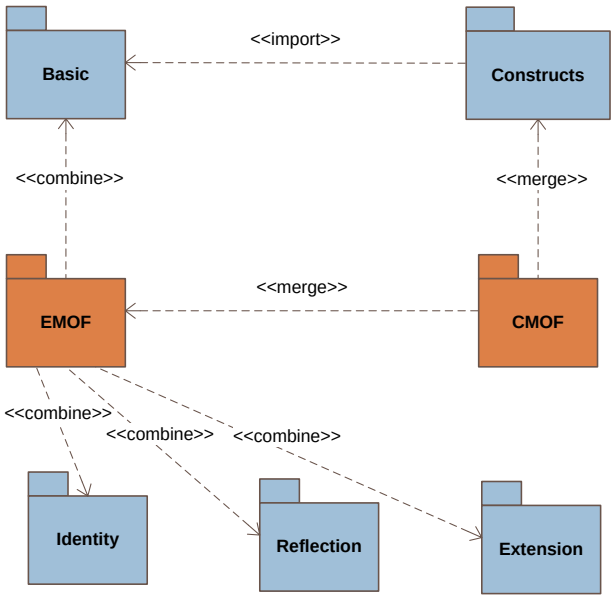


Figura 5.11: EMOF y CMOF

EMOF y CMOF

En la última versión de MOF existen dos meta-metamodelos, el EMOF (Essential MOF) en el que se reduce la expresividad a cambio de obtener sencillez y el CMOF (Complete MOF), que es más complejo pero tiene toda la expresividad. Básicamente, EMOF deriva del paquete *Basic* de la infraestructura UML mientras que CMOF extiende al paquete *Construct* (Fig. 5.11).

5.9. XML Metadata Interchange

MOF XMI (XML Metadata Interchange) [OMG, 2005b] es una especificación basada en XML para compartir metadatos MDA. Actualmente XMI es la base para conseguir la interoperabilidad entre herramientas MDA. Como ventaja principal de XMI se puede citar que se pueden almacenar todos los modelos basados en MOF utilizando un mismo esquema. Sin embargo, tiene inconvenientes tales como que su formato es muy difícil que sea leído por humanos o que muchas de las herramientas existentes que trabajan con XMI no se ajustan a las especificaciones de la recomendación.

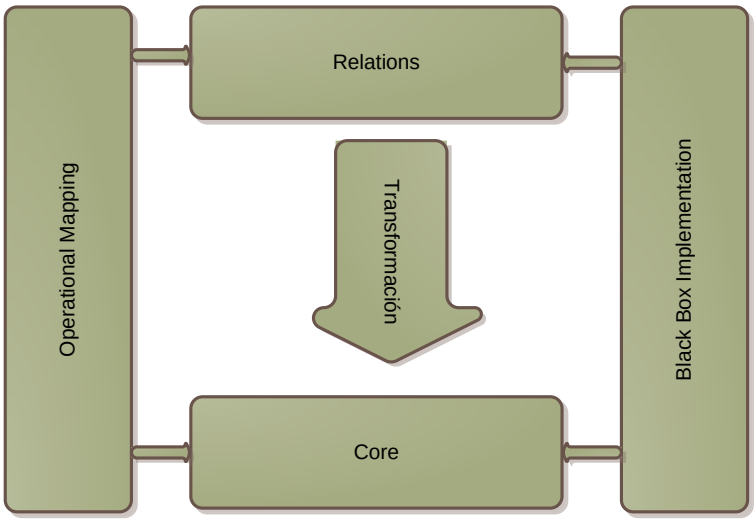


Figura 5.12: Arquitectura de QVT

5.10. Query-View-Transformation

MOF QVT (Query-View-Transformation) [OMG, 2005c] es un estándar de OMG que se ha creado para permitir realizar la transformaciones entre modelos derivados de MOF.

QVT utiliza OCL para realizar consultas y define tres lenguajes de dominio específico *Relations*, *Core*, *Operational Mappings*, que están organizados en una arquitectura en capas.

Relations y *Core* son dos lenguajes declarativos que están en dos niveles diferentes y para los que existe un mapeo entre ellos. Por otra parte, *Operational Mappings* es un lenguaje imperativo que extiende a los dos lenguajes anteriores.

Además, el mecanismo denominado *BlackBox* sirve para dar facilidades de transformaciones expresados en otros lenguajes como por ejemplo *XSLT*. En la Fig. 5.12 puede verse la arquitectura básica de QVT.

QVT sólo sirve para realizar transformaciones de modelo a modelo basándose en MOF pero no sirve para realizar transformaciones de modelo a texto o de texto a modelo.

5.11. Ciclo de vida del desarrollo con MDA

El ciclo de vida del desarrollo de software con MDA tiene las mismas seis etapas que el ciclo de vida del desarrollo de software tradicional (Fig. 5.13), pero con ciertas diferencias:

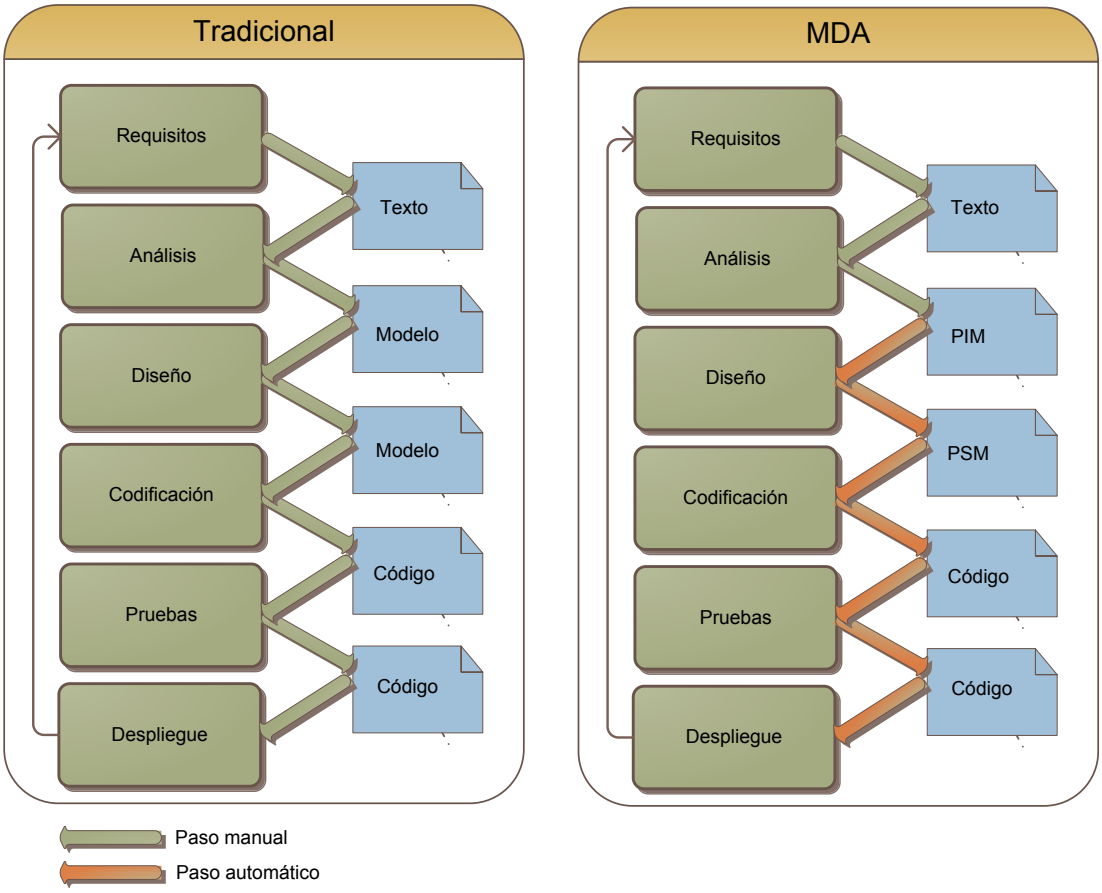


Figura 5.13: Ciclo de vida del desarrollo MDA

- *Requisitos*. La primera etapa es igual en ambos casos y se corresponde con la captura de requisitos de forma textual en un papel. Utilizando la nomenclatura de MDA, esta etapa consiste en construir el CIM.

- *Análisis*. Con el desarrollo tradicional, los requisitos se convierten en diagramas de análisis que básicamente sirven como documentación y que casi nunca están sincronizados con el código debido a las continuas modificaciones que hay que hacer durante el desarrollo y durante el mantenimiento. Con MDA, en el análisis los requisitos se convierten en el PIM, un artefacto de primer orden en el desarrollo, lo cual conlleva muchas ventajas.

- *Diseño*. En esta etapa, con el desarrollo tradicional lo que se hace es obtener de forma manual otros nuevos diagramas, mientras que con MDA lo que se hace es convertir el PIM a PSM de la forma más automatizada posible.

- *Codificación.* Mediante el desarrollo tradicional es necesario convertir los diagramas a código fuente de forma manual, mientras que con MDA la idea es que el código se genere de forma automática a partir del PSM.

- *Pruebas.* Durante el desarrollo tradicional, es necesario emplear una cantidad muy grande de tiempo para realizar pruebas. En cambio, con MDA muchas de las pruebas se pueden evitar o realizar de forma automática, lo que es otra ventaja adicional.

- *Despliegue.* Con el desarrollo tradicional, el despliegue se suele realizar manualmente. Sin embargo, la herramienta MDA podría ayudar automatizando parte o todo el despliegue. En ambos casos se puede utilizar herramientas de integración continua (véase capítulo 4) que ayudarían a realizar el despliegue del software.

5.12. Conclusiones

MDA es el estándar de la industria de software en lo referente al trabajo con la aproximación MDE. A su vez, dicha propuesta de OMG, está basada en otros estándares como MOF, UML, XMI, OCL o QVT. Todas estas tecnologías ofrecen un inmenso abanico de posibilidades que aún no está, ni mucho menos, completamente explotado. Sin embargo, restringirse a las limitaciones que en muchas ocasiones tienen los estándares hace que la productividad de desarrollo se vea afectada.

Por ese motivo, en el siguiente capítulo, se verá otra iniciativa MDE que apuesta más por la productividad, que por la interoperabilidad, portabilidad y reusabilidad que pueden ofrecer los estándares.

CAPÍTULO 6

Factorías de Software

Las SF (Software Factories o Factorías de Software) [Greenfield and Short, 2003] son una iniciativa MDE para elevar el nivel de abstracción y aumentar así el grado de automatización del desarrollo de software cuando dicho software tiene unas características, funcionalidades y arquitectura comunes. Su objetivo primordial es evitar las causas conocidas que producen los problemas tradicionales en el desarrollo de software.

Hay que apreciar que el concepto SF no se refiere a una fábrica en la que se desarrolla software al igual que se haría en las fábricas de coches. Se refiere a una aproximación que eleva el nivel de abstracción para así mejorar el desarrollo de software, que tiene su origen en Microsoft, pero cuyas ideas son independientes de las tecnologías que se utilicen.

En este capítulo se tratarán los pilares sobre los que se fundamentan las SFs y su arquitectura básica. También se desglosan algunas de las principales SFs que existen en la actualidad, así como los principales componentes software que permiten su construcción.

⁎ ⁎ ⁎ ⁎

6.1. Los cuatro pilares de las SFs

Existen cuatro pilares, relacionados entre ellos (Fig. 6.1) [Lenz and Wienands, 2006], en los que se basan las SFs para organizar el desarrollo de software.

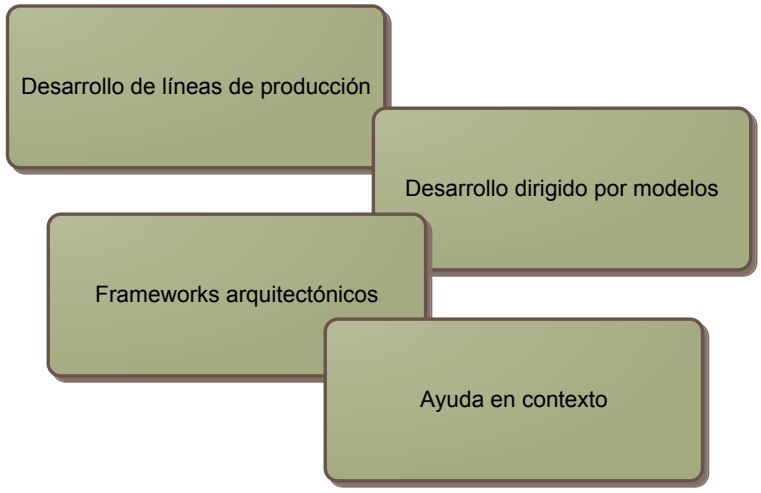


Figura 6.1: Los cuatro pilares de las Software Factories

6.1.1. Desarrollo de líneas de producción

Según [Clements and Northrop, 2002]:

> ❝*Una línea de producción software es un conjunto de aplicaciones que comparten un conjunto gestionado de características que satisfacen las necesidades específicas de un segmento particular, ya que están desarrolladas desde un conjunto de elementos centrales de una manera determinada.*❞

Merece la pena destacar que, ya en 1976, había definiciones de líneas o familias de productos software [Parnas, 1976].

Alrededor de las líneas de producción hay tres conceptos centrales que son:

- *Alcance*. Describe los productos o soluciones que pueden realizarse con la línea de producción.

- *Variabilidad*. Identifica las características comunes y variables de los productos descritos en el alcance.

- *Extensibilidad*. Identifica los llamados *puntos de extensión*, que pueden ser utilizados para añadir nuevas características (no descritas en el alcance) a un producto de la línea de producción.

6.1.2. Frameworks arquitectónicos

Debe existir un framework base que implemente las características comunes de la línea de producción para que de ese modo exista un modo uniforme de organizar todo el desarrollo.

El framework debería utilizar las mejores prácticas conocidas de desarrollo, patrones de diseño y otros conceptos ligados a los frameworks. Por ejemplo, proporcionar formas para realizar la autenticación, autorización, registro de eventos o acceso a datos. Más adelante, en este capítulo, se aborda la Enterprise Library [Microsoft, 2008, Fenster, 2006], una buena base para crear un framework bajo la plataforma .NET

6.1.3. Ayuda en contexto

Las guías contextualizadas son también muy importantes, pues son una forma de facilitar la utilización de las SFs. Pueden ser de diferentes tipos, como por ejemplo:

- Artículos publicados con información útil.
- Ejemplos de código para ver cómo se realizan tareas.
- Páginas de ayuda tipo *how to.*
- Plantillas de código sobre las que basar los desarrollos.
- Patrones de diseño aptos para ser empleados con el framework base.

En lugar de recomendar un sitio en el que buscar y leer las guías cuando lo desee el desarrollador, una de las características de las guías de las SFs es que normalmente son contextualizadas, es decir, van apareciendo automáticamente en el entorno de desarrollo en el momento en el que pueden ser de utilidad.

6.1.4. Desarrollo dirigido por modelos

Al igual que en MDA (véase capítulo 5) se utilizan modelos para generar artefactos, las SFs también lo hacen. La principal diferencia en ese aspecto es que en MDA se confía principalmente en lenguajes estándar (como UML) para guiar *todo* el proceso de desarrollo y las SFs utilizan lenguajes de dominio específico únicamente para *determinadas partes* del desarrollo.

6.2. Arquitectura general de las SFs

Las SFs pueden verse desde dos puntos de vista diferentes. Por una parte, el punto de vista del creador de la SF (*author's view*), y por otra parte, el punto de vista de la persona que utiliza la SF (*consumer's view*).

Para el creador (*author's view*), una SF es una colección de elementos software que se utilizan como base para generar aplicaciones de una determinada línea de producción, es decir, aplicaciones que comparten características, funcionalidades y arquitectura. La SF define la línea general que el desarrollador de software (el usuario de la SF) tiene que seguir.

Para el usuario (*consumer's view*), la SF pasa a formar parte de su entorno de desarrollo y le ayuda a crear miembros de la línea de producción de una manera más eficiente.

En la Fig. 6.2 puede verse la arquitectura general de las SFs desde sus dos posibles puntos de vista.

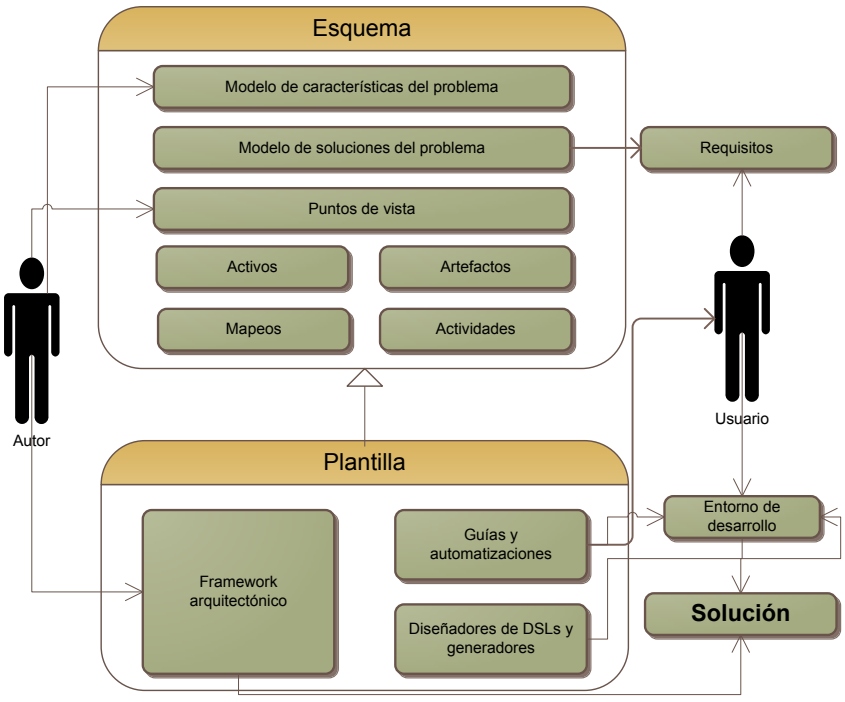


Figura 6.2: Arquitectura general de las Software Factories

6.2.1. Esquema de la Software Factory

El esquema de una SF es un modelo interpretado tanto por humanos como por herramientas, que describe productos, flujos de trabajo necesarios para crear los productos y activos (formados por uno o más artefactos) utilizados en los flujos de trabajo. Todo ello, para una familia de productos software de un determinado dominio [Greenfield et al., 2004].

El esquema está organizado en diferentes puntos de vista que describen al sistema desde diferentes perspectivas y cómo las preocupaciones de los diferentes *stakeholders* son atajadas (p.e., estructura de implementación, paquetes de componentes, distribución física del sistema, comportamiento en tiempo de ejecución).

Para construir el sistema completo hay que combinar todos los puntos de vista [IEEE, 2000]. Cada uno de los puntos de vista tiene una descripción de sus activos más importantes (p.e., herramientas, componentes, librerías, patrones). Además, también se definen actividades que son necesarias para producir los artefactos.

Lo que diferencia a las SFs de otras metodologías, es que las demás definen un proceso con un conjunto preestablecido de puntos de vista que intentan dar solución a todos los problemas de cualquier dominio. Las SFs apuestan por definir procesos específicos para cada posible familia de productos.

6.2.2. Plantilla de la Software Factory

La plantilla de la SF puede considerarse como una instanciación de su esquema. Es una colección de los activos definidos en los diferentes puntos de vista del esquema. Hay de varios tipos:

- *Librerías y frameworks*. Son componentes de software reutilizables, empleados por la arquitectura de la línea de producción. La librería más utiliza en las implementaciones de referencia es la Enterprise Library (véase sección 6.4).
- *Activos guía*. Sirven para agilizar el desarrollo. Por ejemplo, páginas de ayudas, patrones de diseño, *how-tos*, plantillas, *scripts* de construcción, etc.
- *DSLs y diseñadores*. Permiten elevar el nivel de abstracción de los desarrollos software. Un ejemplo podría ser el WWF (Windows Workflow Foundation) [Shukla and Schmidt, 2006] que permite crear modelos de *workflows* que se pueden ejecutar en un motor de *workflows*.
- *Modelos de características*. Dichos modelos describen el alcance de la SF por medio de una lista de características comunes y variables de los productos que se pueden crear con ella.

6.3. Algunas Software Factories

6.3.1. Smart Client Software Factory

La SCSF (Smart Client Software Factory)[1] está considerada como una de las cuatro implementaciones de referencia de las SFs. Se utiliza para facilitar la creación de aplicaciones de escritorio.

6.3.2. Mobile Client Software Factory

La MCSF (Mobile Client Software Factory)[2] también está considerada como una de las cuatro implementaciones de referencia de las SFs. Está especialmente pensada para facilitar la creación de aplicaciones móviles.

6.3.3. Web Client Software Factory

La WCSF (Web Client Software Factory)[3] es otra de las cuatro implementaciones de referencia de las SFs. Sirve para crear aplicaciones Web.

6.3.4. Web Service Software Factory

La WSSF (Web Service Software Factory)[4] es la cuarta implementación de referencia de las SFs. Facilita la creación de servicios Web.

6.3.5. Application Block Software Factory

La Application Block Software Factory [Newton, 2007] es una SF que permite el desarrollo de nuevos Application Blocks personalizados, utilizando guías y ayudas para facilitar el proceso de desarrollo. En las próximas líneas se mostrará qué son los Application Blocks.

6.3.6. Computer Games Software Factory

La Computer Games Software Factory [Furtado et al., 2007] demuestra la aplicabilidad de las SFs en el mundo real. En el trabajo citado se presenta una línea de

[1] http://www.codeplex.com/smartclient/
[2] http://msdn.microsoft.com/en-us/library/aa480471.aspx
[3] http://www.codeplex.com/websf/
[4] http://www.codeplex.com/servicefactory/

producción para crear videojuegos, identificando la arquitectura y los activos utilizados (un DSL, validadores y generadores de código). El objetivo principal del trabajo es mostrar el aumento de productividad gracias a elevar el nivel de abstracción.

6.4. Enterprise Library

La Enterprise Library [Microsoft, 2008,Fenster, 2006] es un componente software muy importante debido al masivo uso que se hace de él tanto en las principales SFs como en otros desarrollos (p.e., en la arquitectura común de las familias de productos software). En general, la Enterprise Library se puede utilizar para cualquier tipo de aplicaciones sean o no, dirigidas por modelos. Se puede afirmar que la Enterprise Library nace como necesidad para ayudar a los desarrolladores software a crear código mucho más compacto, reusable y de calidad.

6.4.1. El origen de la Enterprise Library

Todas las aplicaciones tienen multitud de componentes en común que se hacen y rehacen continuamente cada día en múltiples desarrollos. Puede existir alguna causa que justifique implementar alguno de estos componentes desde cero, pero en prácticamente todos los casos es mucho mejor utilizar componentes creados por expertos, que además ya han sido probados en otros desarrollos. De ese modo, se conseguirá un ahorro muy significativo, tanto en tiempo como en dinero.

Microsoft Patterns and Practices

Microsoft ha creado el Microsoft Patterns & Practices Group[5] para ayudar a los desarrolladores a hacer software de más calidad y en menos tiempo mediante el uso de tecnologías y herramientas de Microsoft.

Se ofrecen fundamentalmente cuatro herramientas, todas ellas de forma gratuita, para todo aquel que las quiera utilizar:

- Guías.
- Implementaciones de referencia.
- Software Factories.
- Application Blocks.

[5] http://msdn.microsoft.com/en-us/practices/default.aspx

Application Blocks

La Enterprise Library está formada principalmente por Application Blocks, que son implementaciones de los patrones y guías de mejores prácticas propuestos por el Microsoft Patterns & Practices Group.

Son componentes *Open Source* que se pueden utilizar para ayudar a resolver problemas comunes a muchos de los desarrollos de software que se realizan diariamente.

Como puntos en común se puede decir que en el diseño de los Application Block se buscan fundamentalmente cuatro características:

- Que sean fáciles de utilizar.
- Que sean consistentes en su diseño.
- Que se integren con facilidad.
- Que tengan puntos de extensión.

Los primeros Application Blocks desarrollados fueron creados para el .NET Framework 1.0. Posteriormente, el Microsoft Patterns & Practices Group junto con la empresa Avanade[6] realizaron el primer conjunto de Application Blocks que formaron parte de lo que se denominó Enterprise Library 1.0 (también llamado Enterprise Library Application Blocks). Antes de esta versión se habían creado otros Application Blocks, pero al ser los primeros no se llegó a cumplir plenamente con las cuatro características que en ellos se buscaban. En esta primera versión se eliminaron alguno de los Application Blocks originales y se añadieron otros nuevos. Esto se debe a que el Patterns & Practices Group se propone que todas las versiones del Enterprise Library sean genéricas para cualquier tipo de arquitectura software, sin estar ligadas a ninguna arquitectura en particular, añadido a que también se ha decidido dejar fuera de los componentes principales (los que aparecen en la Enterprise Library) alguno de los componentes que no se consideraban como prioritarios. Además de los componentes mencionados, también se incluye la Enterprise Library Configuration Tool (también llamada *configuration console*) para facilitar la configuración de los diferentes componentes que se proporcionan.

En los últimos años, y de la mano de las nuevas versiones del .NET Framework, han aparecido varias nuevas versiones de la Enterprise Library. La última versión es la Enterprise Library 5.0 para .NET Framework 4.0, de abril de 2010, cuyos principales componentes se verán en este capítulo.

[6] http://www.avanade.com/

6.4.2. Diseño de la Enterprise Library

La Enterprise Library incorpora un conjunto de *mejores prácticas* en su diseño. A continuación se enumeran algunas de ellas:

- Utiliza el Enterprise Library Core para dar una funcionalidad común.
- Posee formas unificadas para el nombrado y versionado de los diferentes componentes.
- Realiza instrumentación en los diferentes componentes.
- Provee de pruebas unitarias realizadas durante la fase de diseño.

En la Fig. 6.3 se muestra un esquema general[7] de la Enterprise Library junto con sus Application Blocks y las dependencias que existen entre ellos.

Pese a que existen varias formas para crear instancias de los diferentes Application Blocks que pueden ser utilizadas en los desarrollos software (p.e., utilizando las clases *Factory* proporcionadas o mecanismos creados por terceras partes), el modo más común de hacerlo es a través del componente Unity.

El componente Unity permite crear instancias de todo tipo de objetos utilizando el Dependency Injection Pattern [Fowler, 2009b]. Así, se pueden crear automáticamente instancias de objetos en los que existan dependencias.

Es importante señalar que todos los Application Blocks dependen del *kernel* o *Core* de la Enterprese Library. Para crear instancias de objetos es necesario hacerlo a través de *contenedores*.

Cómo muchas de las tareas que realizan los diferentes Application Blocks son comunes a todos ellos e incluso son útiles para aplicaciones realizadas sin la Enterprise Library, se ha creado un componente especial denominado *Enterprise Library Core* para agruparlas de forma externa. El *Core* está formado por tres elementos principales:

- Common Assembly. Agrupa funcionalidades comunes a todos los Application Blocks y se utiliza para reducir la dependencia de unos Application Blocks con otros.
- Instrumentation. La mayoría de los Application Blocks contienen instrumentación (contadores de rendimiento y registro de eventos) que pueden ser utilizados gracias a este componente. Por defecto está desactivado.

[7] En la figura se respetan los nombres originales en inglés

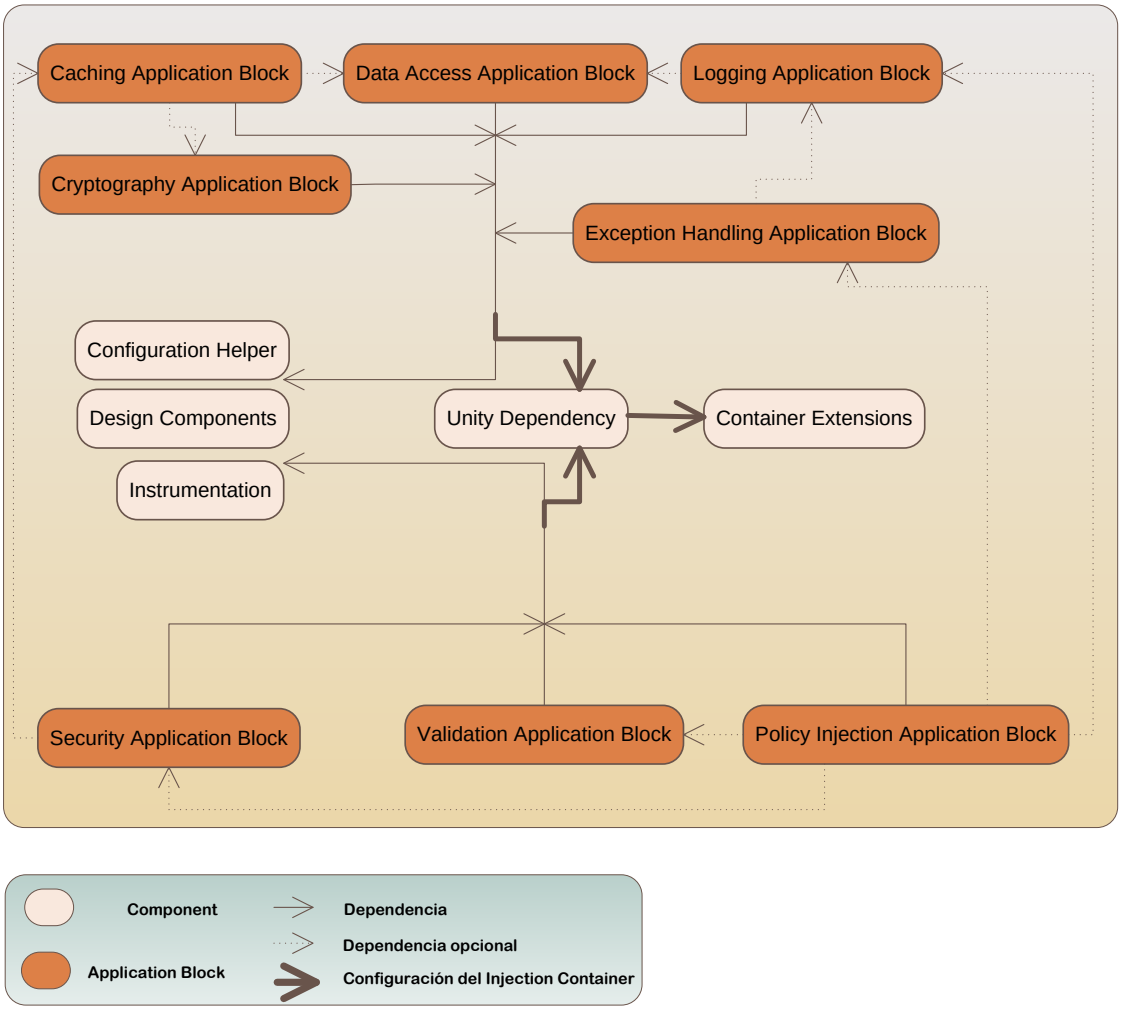


Figura 6.3: Estructura general de la Enterprise Library

- Configuration Helper Classes and Design-time Components. Proporciona un modo estandarizado, sencillo y extensible de tratar con archivos de configuración de aplicaciones. Además, existen componentes que permiten cambiar la configuración de los Application Blocks sin modificar los archivos de configuración XML.

Data Access Application Block

Típicamente, las aplicaciones tienen un capa de datos que contiene una capa de almacén de datos (que puede ser una base de datos, un archivo de texto o cualquier otro medio que sirva para almacenar información) y una capa de acceso a datos. Debido a que hay muchos tipos de almacenes de datos y muchos tipos de accesos a datos, hay que aprender a utilizarlos para cada caso concreto, lo que puede suponer

un problema.

El Data Access Application Block sirve para realizar el acceso a los datos de forma uniforme y proporciona una gran cantidad de facilidades.

Caching Application Block

El uso de una memoria caché, con moderación, puede acarrear interesantes beneficios [Smith, 1982]:

- Aumento del rendimiento debido a que la comunicación con el almacén de datos no será tan intensa[8].
- Aumento de la escalabilidad de la aplicación gracias a que se puede evitar hacer continuamente nuevas consultas a la base de datos.
- Posibilidad de que las aplicaciones funcionen de manera *offline* debido a que la información estará ya guardada en el caché.

El Caching Application Block aparece porque en casi todas las aplicaciones se trabaja con información que se guarda de manera persistente en algún tipo de almacén de datos. Es interesante porque trabajar con las bases de datos tradicionales acarrea un coste adicional, ya que hoy en día leer y escribir en disco es una operación todavía muy lenta.

Hay alternativas como las SANs (Storage Area Networks) que utilizan un buffer en memoria mientras se lee o se escribe información pero son muy caras. También se puede utilizar memoria RAM como disco duro pero aún no es una tecnología madura.

Exception Handling Application Block

Un lenguaje moderno ofrece sistemas para capturar y tratar excepciones, además de permitir al usuario crear sus propias excepciones. Por ejemplo, .NET tiene una clase denominada *System.Exception* que es la base para todo el tratamiento de excepciones. De dicha clase heredan otras dos clases, *System.SystemException*, base para las clases dedicadas al tratamiento de excepciones lanzadas por la plataforma .NET y *System.ApplicationException*, base para las clases que los desarrolladores puedan crear y lanzar en sus propias aplicaciones.

Además, es recomendable y necesario, tener un registro almacenado con información generada con las excepciones para facilitar la localización de la fuente del error

[8]Los accesos a disco se miden típicamente en milisegundos y los accesos a la memoria RAM se miden en nanosegundos

y una rápida corrección en caso de ser necesaria. También, en muchas ocasiones, es necesario advertir o avisar al usuario sobre las excepciones que se están produciendo en el programa, pero hay que tener mucho cuidado con no mostrar información inadecuada que pueda dejar agujeros de seguridad abiertos o que pueda contener información confidencial de cualquier usuario como información bancaria, personal o médica.

El Exception Handling Application Block nace como necesidad debido a que todo el mundo es susceptible de cometer errores cuando se desarrolla o se utiliza software, que sólo son identificables en tiempo de ejecución. Los lenguajes modernos como Java o C# proveen medios para identificar, capturar y tratar dichos errores (p.e., las instrucciones *try*, *catch*, *finally*, y *throw* de C#), pero este bloque da un paso más y ofrece mucha más funcionalidad.

Logging Application Block

El registro de eventos (*logging*) puede ser realizado con diversos fines:

- Instrumentación. La instrumentación consiste en guardar información sobre el estado de *salud* de la aplicación.
- Rastreo y Depuración. Consisten en guardar información en diferentes puntos estratégicos de la aplicación. Con fines de *rastreo* y *depuración* generalmente se guarda menos información que con la instrumentación pero con mayor nivel de detalle.
- Auditorías. Las auditorías suelen ser registros muy grandes de datos que guardan las empresas con información histórica sobre todos los cambios que se han hecho con información sensible (p.e., números de teléfono) durante la ejecución de la aplicación.

Existen múltiples formas de guardar la información registrada:

- Utilizar un fichero de texto (plano, XML, CSV, etc.).
- Guardar la información en un sistema de gestión de bases de datos.
- Enviar la información a una cola de mensajes para ir tratándolos uno a uno.
- Enviar la información por correo electrónico al responsable de mantenimiento.
- Si se está utilizando Windows, utilizar el registro de eventos de Windows o EIF (Enterprise Instrumentation Framework).

- Además, se puede utilizar una combinación de alguna de las formas anteriores dependiendo de la prioridad y del impacto que pueda tener la información. Así, un mensaje informativo puede estar guardado en un fichero XML, pero un mensaje con una excepción muy grave podría enviarse directamente por correo electrónico al responsable de mantenimiento.

Prácticamente todas las aplicaciones de cierta envergadura (incluso las más sencillas) guardan un registro de eventos sucedidos. El Logging Application Block es la apuesta de la Enterprise Library para proporcionar un método común, flexible y eficiente que sirva para guardar información de lo que sucede durante la ejecución de una aplicación.

Cryptography Application Block

La criptografía es un término originado del griego (*krypto* 'ocultar' y *graphos* 'escribir') que tiene como objetivo principal garantizar que la comunicación entre entidades sea secreta, es decir, que sólo dichas entidades puedan averiguar el contenido del mensaje.

Otro objetivo adicional asegurar que tanto el emisor como el receptor son efectivamente quien dicen ser y que el mensaje llega sin manipular a su destino. Puede obtenerse más información sobre criptografía en [Menezes et al., 1996].

Existen varios métodos criptográficos entre los que destacan en algoritmo *hash*, la encriptación simétrica, la encriptación asimétrica o la encriptación híbrida.

El Cryptography Application Block proporciona una forma común de realizar la criptografía para cualquier aplicación que se desarrolle bajo el .NET Framework.

Security Application Block

Existen tres pasos fundamentales (conocidos como la triple A) [Metz, 1999] en seguridad de sistemas informáticos, los cuales se indican a continuación por orden de importancia.

Mediante la *autenticación* se trata de verificar que el usuario es realmente quien dice ser. Como problema típico en las aplicaciones software está el hecho de que la autenticación generalmente no se hace en modo local, sino en modo remoto, lo que puede ocasionar, sobre todo en entornos sin estado como la Web, que los datos del usuario sean comprobados cada vez que realizan una acción, provocando una sobrecarga innecesaria.

Mediante la *autorización* se permite a los usuarios acceder a los recursos del sistema para los que tengan permisos. En el caso de la autorización en las aplicaciones

de software, el problema es que existe una falta madurez para que no se trate de reinventar la rueda una y otra vez.

Mediante las *auditorías* se registran todos los accesos que se hacen a los recursos del sistema por parte de los usuarios, sean autorizados o no. Para realizar auditorías se utilizan los sistemas de registro de eventos.

La seguridad es un aspecto muy importante en prácticamente todas las aplicaciones y sin embargo, es un tema que muchas veces no se trata correctamente debido a las prisas y al incremento constante de complejidad de los nuevos desarrollos, que hace que ni los propios desarrolladores estén preparados.

El Security Application Block aparece para simplificar y estandarizar el modo en el que los usuarios son autorizados a acceder a cierta información sensible y el modo en el que se cachean los perfiles autenticados dentro de la aplicación.

Validation Application Block

La validación es el proceso de asegurar que los datos con los que se trabaja en una aplicación son correctos y no pueden causar problemas de seguridad o corrupción de otros datos. Con la validación de los datos se asegura que los datos son válidos y seguros antes de que sean procesados. Existen muchos métodos utilizados para realizar la validación de los datos:

- Comprobar el formato de la cadena introducida. Por ejemplo, para asegurar que la fecha se introduce correctamente.
- Comprobar el tipo de los datos de la entrada. Por ejemplo, si se pide la edad y se introduce una cadena de texto debería aparecer un mensaje de advertencia.
- Comprobar el rango de los datos. Por ejemplo, el número de mes deberá estar comprendido entre 1 y 12.
- Comprobar el límite del valor. Por ejemplo, para introducir una edad de una persona no se permitirá introducir el valor -1 ni el valor 150.
- Comprobar la presencia del valor. Por ejemplo, cuando se rellena un formulario y se le pide al usuario la dirección de correo electrónico, el sistema comprobará que ciertamente se ha introducido.
- Comprobar la consistencia de los datos. Por ejemplo, no se puede decir que se es una mujer y después especificar que la forma de tratamiento hacia esa persona es "Mr".

- Comprobar la cadena de entrada. Por ejemplo, un número de teléfono móvil de España tiene que estar formado por 9 dígitos y el primero de ellos tiene que ser un 6.

- Comprobar los dígitos de entrada. Se añade un dígito extra calculado en función del resto de dígitos introducidos. Por ejemplo, el último dígito de los ISBN de los libros se calcula así.

- Comprobar la consistencia a través de sistemas. Consiste en comparar un mismo dato a través de dos sistemas diferentes. Por ejemplo, el número de teléfono del mismo nombre de usuario debería coincidir en ambos sistemas.

El Validation Application Block facilita la implementación de reglas de dominio dentro de una aplicación, posibilitando la creación de validadores a los objetos de los tipos de datos que se utilicen en el .NET Framework.

Policy Injection Application Block

Hablar de inyección de políticas es lo mismo que hablar de la programación orientada a aspectos[9], cuya principal meta es la de la separación de las diferentes incumbencias que pueda tener una aplicación.

La separación de incumbencias o aspectos se refiere a separar diferentes módulos de código cuyos objetivos son distintos entre sí y ortogonales a la funcionalidad principal de la aplicación. Una de las ventajas de la programación orientada a aspectos es que la lógica de dominio no se ve mezclada con código que no es específico de la lógica de dominio. Por ejemplo, un método que calcula el IVA (Impuesto sobre el Valor Añadido) de un precio debería únicamente calcular el IVA de ese precio y no debería tener código para hacer *logging*, validaciones, tratamientos de excepciones, controlar la autenticación o controlar la autorización. Todo el código que no se dedique a calcular únicamente el IVA debería estar fuera de ese método separado en diferentes incumbencias.

El desarrollo de software evoluciona constantemente y por esa razón se ha pasado de desarrollar aplicaciones utilizando código máquina a utilizar el lenguaje ensamblador. Después, llegaron los lenguajes procedimentales y los lenguajes orientados

[9]Según quien presentó el concepto de programación orientada a aspectos, Gregor Kiczales: [Kiczales, 1996] 'un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares del programa'

a objetos. El último paso, sin contar con MDE, es la aparición de los lenguajes orientados a aspectos.

El Policy Injection Application Block permite utilizar el paradigma de desarrollo que utiliza aspectos de una manera sencilla.

6.5. MDA versus SFs

No hay una opinión generalizada sobre qué iniciativa MDE es más interesante para el desarrollo de software. De hecho, los mayores expertos en la materia no se han puesto de acuerdo. De entre todas las iniciativas actuales destacan sobre todas las demás dos de ellas. La primera es MDA, por tratarse del estándar reconocido de la industria en lo referente al trabajo con la ingeniería dirigida por modelos. La segunda son las SFs, por ser posiblemente la más empleada en los desarrollos empresariales actuales.

Steve Cook, en su artículo [Cook, 2004] comenta que MDA guarda cierto paralelismo con la promoción y posterior fracaso de las herramientas CASE (Computer-Aided Software Engineering) en la década de los 80 y dice que es probable que MDA también fracase a no ser que se consiga combinar adecuadamente modelos, patrones, frameworks y código dentro de un proceso de desarrollo ágil, ya que de no ser así MDA no será nunca realmente una arquitectura.

El uso de UML como UMLAsBlueprint[10] limita aún más a MDA, ya que pese la extensibilidad que permiten los perfiles UML, la práctica ha demostrado que no es viable realizar transformaciones de cierta complejidad a una plataforma tecnológica específica.

Por su parte, MOF (Meta-Object Facility) es un lenguaje de modelado de dominio específico utilizado para definir lenguajes de modelado como por ejemplo UML, pero el problema es que MOF sólo define los conceptos básicos en un lenguaje y

[10]Martin Fowler, en uno de los libros más importantes sobre UML [Fowler, 2004], distingue entre tres usos principales de UML:

- *UMLAsSketch*. UML ha sido muy exitoso en el sentido de que se puede utilizar para documentar de manera uniforme diferentes sistemas software orientados a objetos, eliminando diferentes tipos de representaciones que no hacían más que dificultar la comprensión
- *UMLAsBlueprint*. En este sentido UML es utilizado como un artefacto de primera mano que se transformará de manera automática o semi-automática en código fuente operativo. Esto significa que el modelo UML debería ser lo suficientemente rico como para que la traducción sea completa, pero la realidad hace ver que no es así, ya que por ejemplo un diagrama de clases UML no soporta campos *static*
- *UMLAsProgrammingLanguage*. Hay quien ve a UML como un lenguaje de programación más con el que construir software. En este sentido UML es muy minoritario

cómo dichos conceptos son almacenados e intercambiados, quedándose lejos de lo que debería hacer: definir los conceptos de un dominio, representar los conceptos del dominio gráfica o textualmente, definir maneras para que los usuarios puedan interactuar con el lenguaje, identificar qué modelos son válidos o no, y especificar cómo los modelos son intercambiados. El trabajo que hay que hacer para conseguir una iniciativa válida de MDE integrándola en el entorno Visual Studio requiere elementos que van mucho más allá que lo que puede proporcionar MOF.

XMI (XML Metadata Interchange) no es la solución ideal para serializar los modelos surgidos de MOF porque hay muchos problemas de incompatibilidades entre las diferentes versiones de MOF, UML, e XMI. Por ello, la mejor solución para serializar un determinado lenguaje de modelado es construir un *schema* específico para ese lenguaje. Así, Steve Cook concluye que Microsoft no soporta MOF por varios motivos:

- No es un estándar estable.
- Ampliar MOF con los elementos que le faltan lo harían más inestable todavía.
- No se puede utilizar para definir las herramientas de Microsoft porque no cubre todas sus expectativas.
- XMI falla en la serialización.

Michael Guttman respondió en [Guttman, 2004] al trabajo escrito previamente por Steve Cook [Cook, 2004] diciendo que los estándares de OMG no son la única iniciativa posible para MDE pero si son una buena solución cuando se busca la interoperabilidad entre diferentes herramientas y entre diferentes plataformas. Además, hay muchas personas que están investigando en dichos estándares e incluso existen muchos casos de éxito en su utilización, como los citados en [Guttman and Parodi, 2006]. Otro aspecto es que tanto UMLAsBlueprint como UMLAsProgrammingLanguage están siendo cada vez más soportados por las herramientas, hecho que Steve Cook ha ignorado en su artículo.

Por otra parte, es evidente que Microsoft hasta ahora ha ignorado la importancia de MDE y pese a las críticas realizadas, Microsoft tiene en cuenta las propuestas de OMG, ya que tanto el propio Steve Cook como otras personas contratadas por Microsoft provienen del OMG. Sin embargo, parece ser que ha reaccionado tarde y prefiere crear otra nueva iniciativa mediante de las SFs, quizá con motivos comerciales.

Por último, Michael Guttman señaló que el motivo por el que las herramientas CASE fracasaron fue principalmente que los diferentes vendedores utilizaron dife-

rentes estándares para solucionar un mismo problema que todos tenían en común y MDA se creó para evitar que vuelva a suceder lo mismo.

Dave Thomas [Thomas, 2004] dice que los generadores de código MDA y UML, usados con moderación son útiles y pueden ayudar, pero están lejos de ser lo que muchos han augurado. Al igual que Steve Cook y Stuart Kent, opina que UML sólo es válido para muchos proyectos relacionados con la informática y las telecomunicaciones pero falla para proyectos en otros dominios.

Steve Cook y Stuart Kent [Cook and Kent, 2008] también criticaron a UMLAs-Blueprint por dos razones:

- UML es muy grande y los APIs que ofrecen para trabajar con los modelos son muy difíciles de utilizar.

- UML no es específico de dominio y pese a que soporta un mecanismo de extensión llamado perfiles UML, que permite añadir más información a los modelos, este hecho no es suficiente para salvar las diferencias conceptuales entre el dominio y UML. Sólo se puede utilizar con éxito UML para MDE cuando exista una correspondencia muy clara entre el dominio y el modelo UML y así, con algunos datos y reglas de validación adicionales sería suficiente para que los modelos sean lo suficientemente precisos para generar código.

Ambos autores están a favor de utilizar DSLs en lugar de UML, utilizando herramientas específicas como las citadas en [Langlois et al., 2007]. Así, proponen la creación de IDEs (Integrated Development Environments o Entornos Integrados de Desarrollo) de dominio específico porque tienen las ventajas del desarrollo dirigido por modelos y los lenguajes de dominio específico.

Steven Kelly y Juka-Pekka Tolvanen [Kelly and Tolvanen, 2008] critican MDA y apoyan la utilización de DSLs. Según ellos, UML es demasiado general y no sirve para generar soluciones prácticas a través de la transformación de modelos. Además, quien utiliza UML para definir dominios tiene que tener un buen conocimiento tanto de UML como del dominio. Gracias a los perfiles UML y a las restricciones OCL se puede obtener más potencia pero está muy lejos de lo que se puede conseguir con un lenguaje de dominio específico, debido, sobre todo, a que en un DSL sólo existen los conceptos estrictamente necesarios para el dominio y por tanto, es más sencillo crear aplicaciones.

6.6. Conclusiones

De todas las exposiciones anteriores, se puede obtener como conclusión que con MDA se consigue aumentar sobre todo, la interoperabilidad, la portabilidad y la reusabilidad de sistemas y que con otras iniciativas, más específicas de un dominio concreto, se aumenta la productividad y la calidad del desarrollo.

Así, las SFs son una de las principales iniciativas MDE existentes en la actualidad, muy fuertemente apoyada por Microsoft. La idea es centrarse en la productividad y en las herramientas antes que en los estándares y en la interoperabilidad. Para ello, se proveen interesantes componentes como la Enterprise Library. Además, las ideas subyacentes a las SFs no están ligadas a ninguna tecnología y podrían implantarse bajo cualquier plataforma.

En el siguiente capítulo se presentará el trabajo TMDE (TALISMAN MDE) [García-Díaz et al., 2010a], cuyo objetivo es unir en una única iniciativa, las ventajas ofrecidas por MDA con las ventajas ofrecidas por las SFs. De hecho, hay una gran cantidad de opiniones contradictorias sobre qué iniciativa MDE es mejor y debe ser empleada.

CAPÍTULO 7

Solución adoptada

Las principales y más referenciadas propuestas MDE son, por un lado, MDA [Miller et al., 2003] y, por otro lado, las Software Factories [Greenfield and Short, 2003].

MDA es seguida principalmente por los defensores de los estándares en el desarrollo de software.

Las Software Factories son el estandarte del resto de iniciativas que abogan por dar importancia a la productividad ofrecida por las líneas de productos y el empleo de herramientas y tecnologías orientadas a plataformas específicas.

En este capítulo se presentará la iniciativa MDE denominada TMDE (TALISMAN MDE) [García-Díaz et al., 2010a], que pretende mezclar las mejores ideas tanto de MDA como de las Software Factories en un único proceso de desarrollo. Así, se consigue la máxima productividad con la máxima posible interoperabilidad, portabilidad y reusabilidad.

* * * *

7.1. Introducción

Debido a la falta de unanimidad y a las peculiaridades de cada proyecto, no es posible saber qué iniciativa MDE es la más adecuada, ya que todas tienen sus puntos fuertes y débiles. Además, en función de la iniciativa empleada, cambian varios aspectos que deben tenerse en cuenta:

- El tipo de modelo que se almacenará en los repositorios. Los modelos MDA se serializan al formato estándar XMI, mientras que con las Software Factories no hay definido ningún tipo estándar de serialización.
- El número de repositorios. Con un desarrollo puro MDA sólo hace falta uno (todo son modelos), mientras que con las Software Factories idealmente hacen falta tres (framework arquitectónico, código específico y modelos).
- Las herramientas de modelado utilizadas para crear los modelos.
- Las herramientas para realizar los parseos de los modelos, sus transformaciones y la generación de artefactos.

Para no tener que elegir entre las diferentes iniciativas y, por lo tanto, entre sus ventajas e inconvenientes, en [García-Díaz et al., 2010a] se ha propuesto una nueva iniciativa que trata de potenciar al mismo tiempo las principales ventajas de MDA y de las Software Factories.

En la Fig. 7.1 puede observarse la arquitectura general de MDA. Como principal peculiaridad se puede decir que con MDA el objetivo es generar todos los artefactos mediante el uso de modelos realizando transformaciones de CIM a PIM y de PIM a uno o varios PSM. El último paso será generar los artefactos específicos para una plataforma.

La gran dificultad radica en que, como muchos autores han dicho, únicamente a través de modelos es muy difícil generar código completo y funcional para los incalculables dominios en los que se puede y se podrá aplicar la informática.

Además, el empleo de metamodelos que deriven únicamente de MOF (el estándar de OMG), limita mucho su empleo debido a posibles carencias en el diseño de la sintaxis abstracta y a que la sintaxis concreta generada no siempre es sencilla de utilizar para personas que no tengan unos conocimientos de informática adecuados. De todas formas, con MDA se alcanza el mayor grado de interoperabilidad y portabilidad gracias al empleo de estándares.

Un importante paso adelante es el empleo de GMF (Graphical Modeling Framework) y TMF (Textual Modeling Framework) bajo la plataforma Eclipse, que

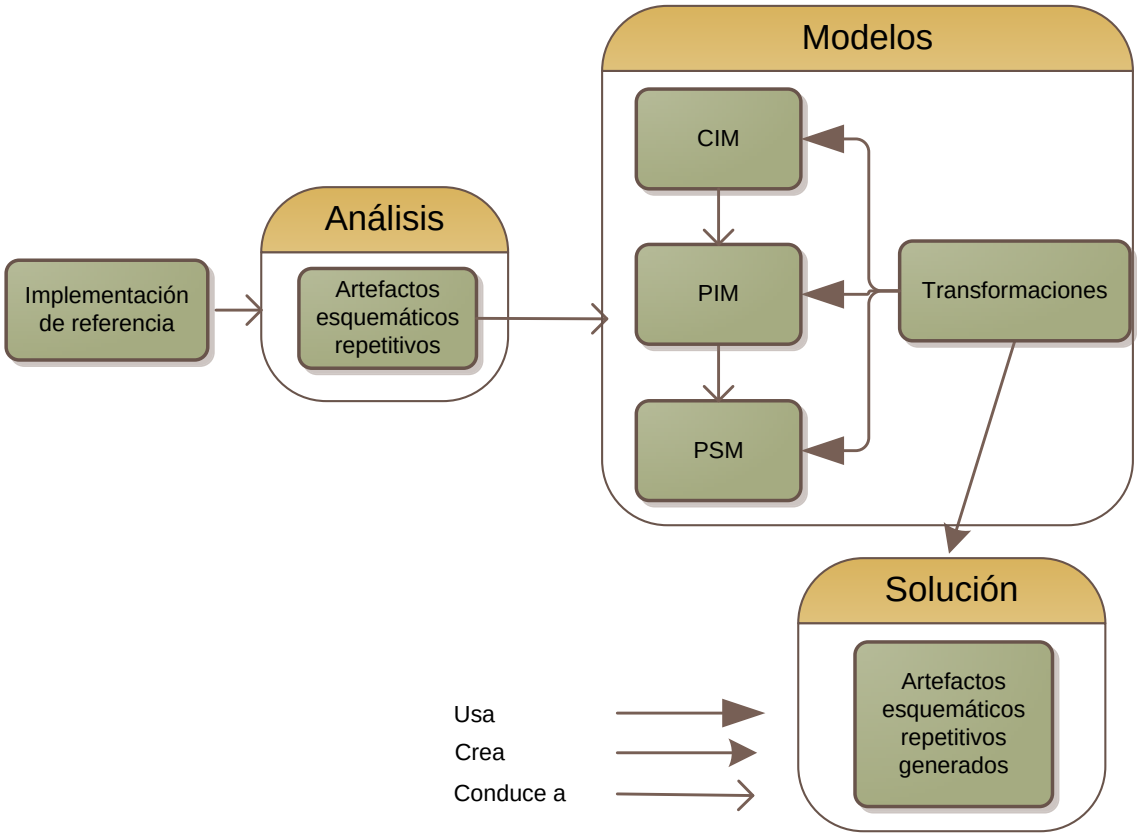


Figura 7.1: Arquitectura global de MDA

permiten crear sintaxis concretas personalizadas para trabajar con sintaxis abstractas diseñadas con EMF (Eclipse Modeling Framework) [Budinsky et al., 2003,Steinberg et al., 2009], mediante el empleo de Ecore (un meta-metamodelo muy próximo a MOF). Sin embargo, dichas herramientas no siguen los principios del estándar MDA en cuanto al desarrollo de software, que apuesta por el empleo de perfiles UML en lugar de por la creación de nuevos metamodelos.

Por otra parte, en la Fig. 7.2 puede observarse la arquitectura general de las Software Factories. Como se ve, con las Software Factories no es necesario realizar diferentes transformaciones mediante el empleo de distintos puntos de vista.

Al no requerir el empleo del meta-metamodelo MOF, se facilita trabajar con otras sintaxis abstractas y por tanto, se puede permitir que los usuarios de las herramientas de modelado trabajen con sintaxis concretas más adecuadas a sus conocimientos, adaptando los elementos del lenguaje a los conceptos que domine el usuario; por ejemplo, si se crea un DSL para trabajar con procesos de fabricación de alimentos es conveniente que los elementos del lenguaje recuerden al usuario los conceptos

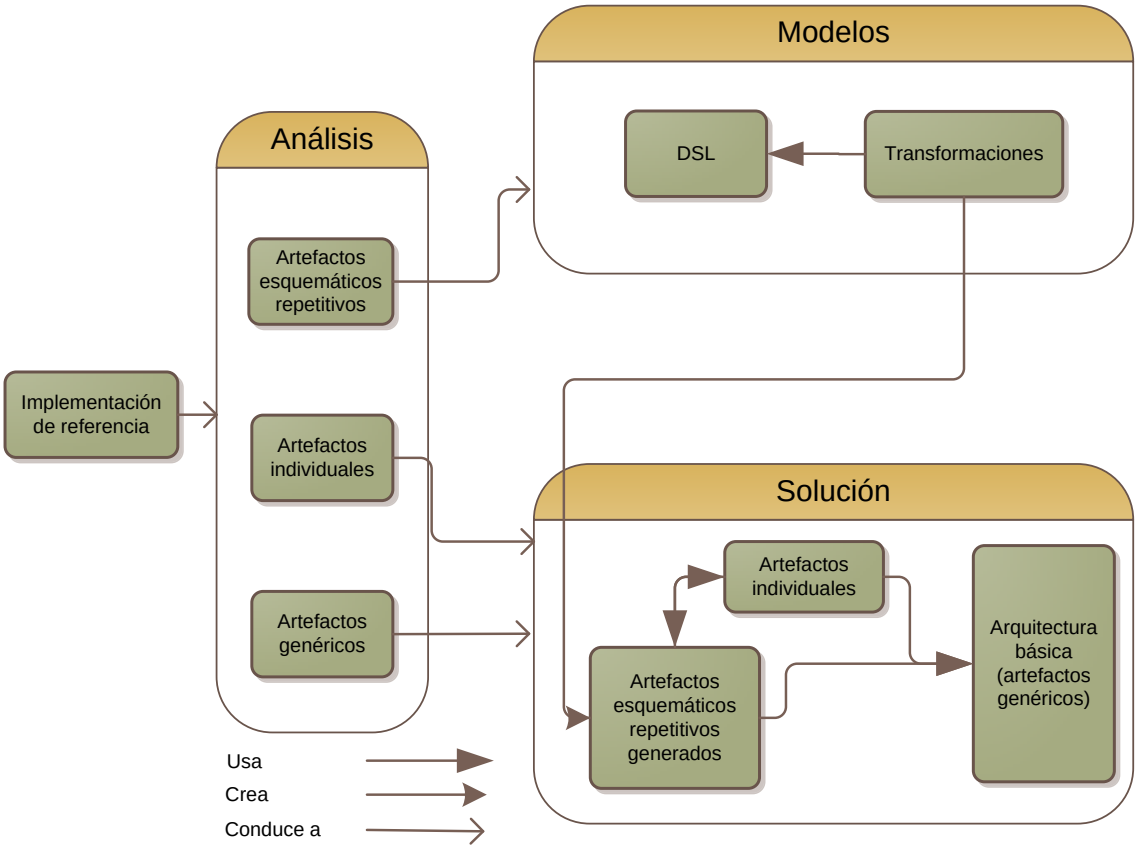


Figura 7.2: Arquitectura global de las Software Factories

que él domina, procurando evitar que el usuario maneje conceptos directamente relacionados con la informática como los empleados en los diagramas UML.

Además, el hecho de evitar las transformaciones CIM, PIM y PSM puede derivar en desarrollos más sencillos. Un detalle importante es que no se busca generar todo el código de las aplicaciones mediante los modelos, ya que la idea es que los modelos sirvan únicamente para modelar la parte variable que, siguiendo un patrón predefinido, cambia de unos desarrollos a otros de una misma línea de producción.

7.1.1. Antecedente

El trabajo presentado en [García-Díaz et al., 2009c] (TALISMAN MDE Framework) es el antecedente al trabajo presentado en este capítulo [García-Díaz et al., 2010a]. La idea es unir lo mejor de MDA y de las SFs en un único framework. Realmente los trabajos de [García-Díaz et al., 2009c] y [García-Díaz et al., 2010a] son diferentes y podrían evolucionar por distintos caminos, aunque las lecciones apren-

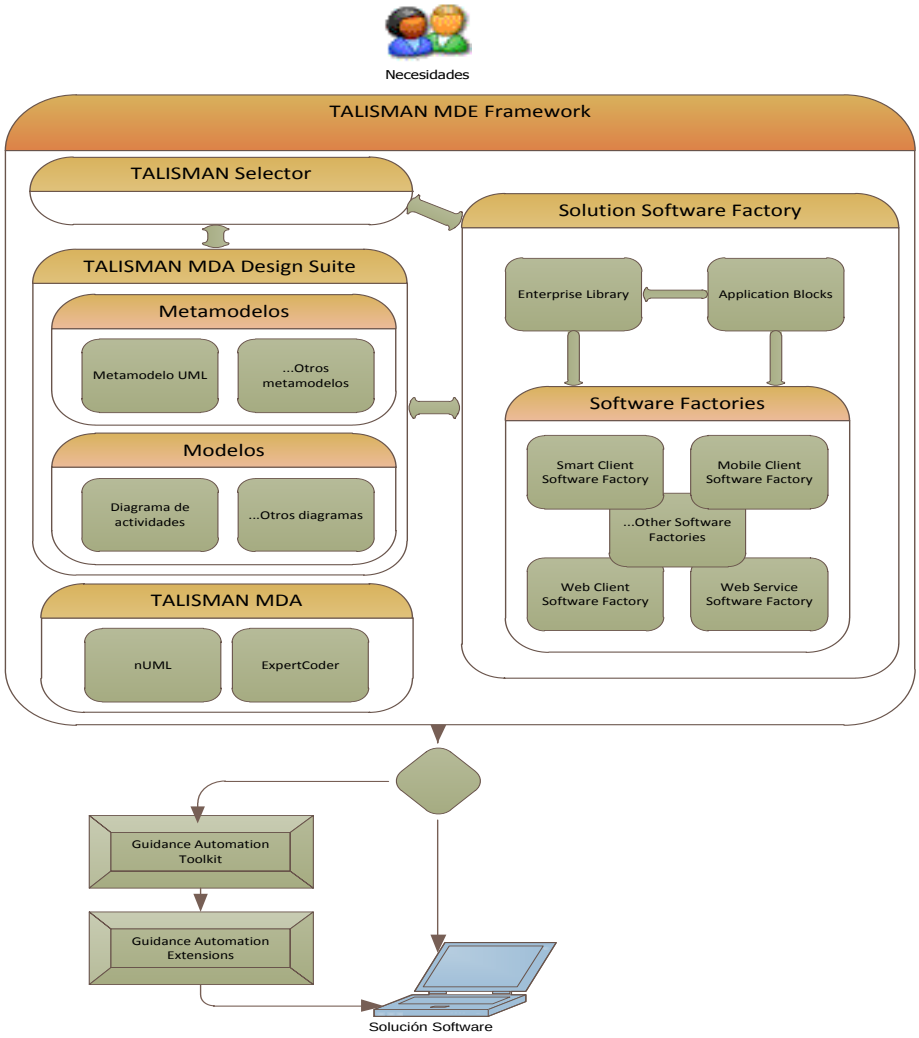


Figura 7.3: TALISMAN MDE Framework. Arquitectura general

didas de un trabajo desencadenaron en la realización del otro. En el primero, la idea es crear un entorno de desarrollo mientras que en el segundo la idea es crear un marco de trabajo conceptual. Así, la propuesta de [García-Díaz et al., 2009c] tiene cuatro componentes fundamentales que pueden verse en la Fig. 7.3:

- TALISMAN Selector. El objetivo de este componente es evitar la tarea repetitiva de tener que crear desde cero los proyectos de software y el código necesario para conectarlos. La mayoría de las soluciones software de cierto nivel de complejidad están formadas por más de un proyecto. El objetivo es proporcionar un DSL con una interfaz gráfica para añadir, eliminar y actualizar la configuración básica de cualquier proyecto que esté incluido en una

única solución.

- Solution Software Factory. El objetivo de este componente es unir las mejores características de las SFs más importantes para diseñar una SF que sea capaz de generar muchos tipos de soluciones software. La idea es que los desarrolladores puedan añadir diferentes tipos de proyectos (p.e., Web, escritorio, móvil) que se integren utilizando una arquitectura común. Así, se evitan las inconsistencias que hay en el diseño de las SFs actuales, ya que han evolucionado de forma diferente a lo largo del tiempo.
- TALISMAN MDA. El objetivo de este componente es ser el responsable de todos los aspectos relacionados con el estándar MDA dentro del entorno de desarrollo. Está formado principalmente por proyectos de código abierto.
- TALISMAN MDA Design Suite. El objetivo de este componente es desarrollar los elementos principales de diversos metamodelos MOF con la idea final de crear diferentes diseñadores de diagramas que puedan estar sincronizados en todo momento con el código fuente de los proyectos incluidos en la solución.

Existen opiniones que defienden que MDE podría no tener éxito[1] si no se cumple con una serie de requisitos como los siguientes:

1. Afrontar todos los objetivos de MDE.
2. Utilizar todas las dimensiones del estándar MDA.
3. Focalizar en la generación de artefactos.
4. Centrarse en lenguajes de dominio específicos.
5. Disponer de herramientas para soportar el proceso.
6. Probar los modelos con facilidad, etc.

Con el trabajo de [García-Díaz et al., 2009c] se cumple, entre otros, con todos los objetivos mencionados, lo que puede servir como apoyo al trabajo realizado.

7.2. Arquitectura utilizada

Trabajos como [Cook, 2004] o [Kelly and Tolvanen, 2008] muestran que únicamente con MDA no es posible crear soluciones completas de una cierta complejidad.

[1] http://www.infoq.com/articles/8-reasons-why-MDE-fails/

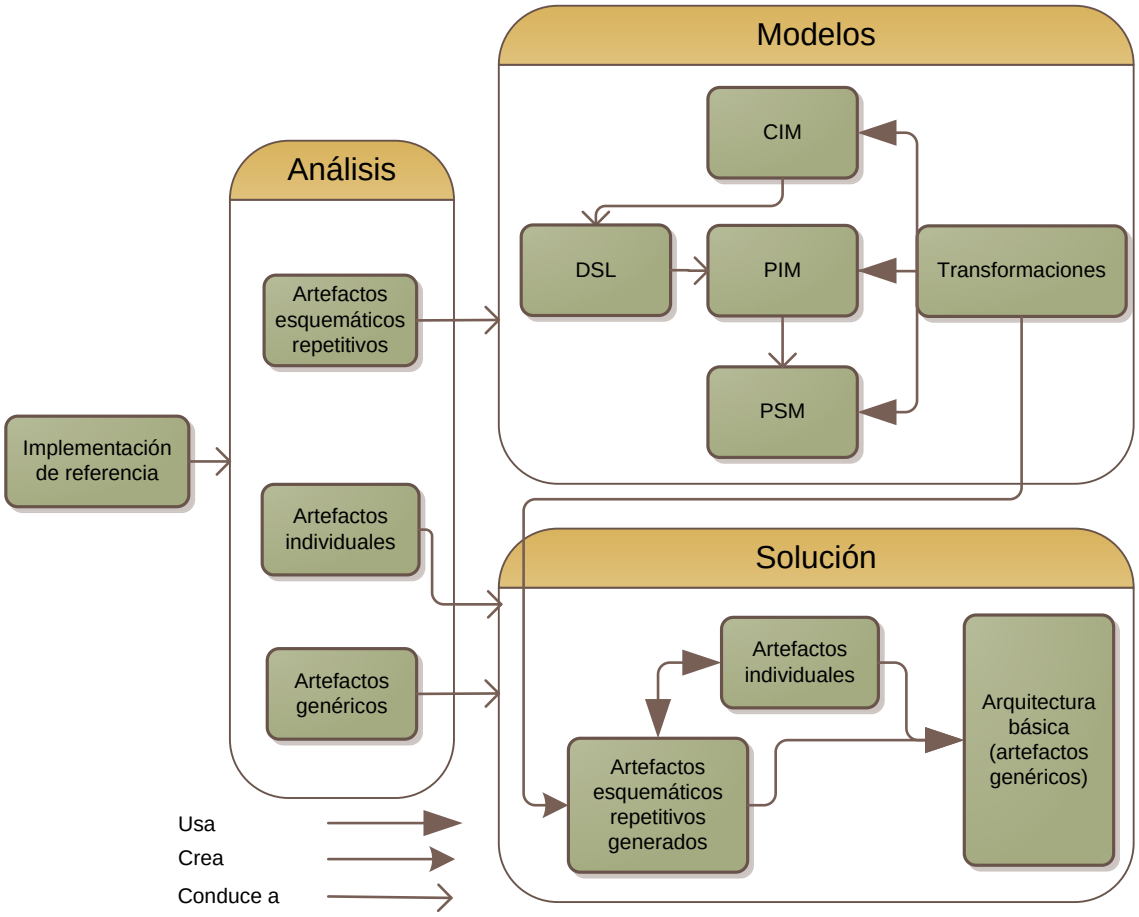


Figura 7.4: Arquitectura global de TALISMAN MDE

Sin embargo, aunque es cierto que otras iniciativas como las Software Factories o AC-MDSD (Architecture Centric Model-Driven Software Development) intentan crear soluciones completas, también es cierto que tienen que pagar el precio de centrarse únicamente en una arquitectura específica, perdiendo la portabilidad y la interoperabilidad ofrecida por MDA [Völter and Stahl, 2006].

Por ejemplo, en [Furtado et al., 2007] se muestra un trabajo que eleva el nivel de abstracción y que incrementa la automatización en la creación de videojuegos gracias a extensiones específicas de Visual Studio y de la plataforma .NET.

No hay ningún generador de código que bajo las directrices de MDA pueda ser capaz de generar artefactos de forma tan eficiente como las Software Factories para un dominio particular. De hecho, algunas de las herramientas más conocidas para crear soluciones MDA (p.e., AndroMDA[2]) muestran casos reales casi exclusivamente

[2]http://galaxy.andromda.org/

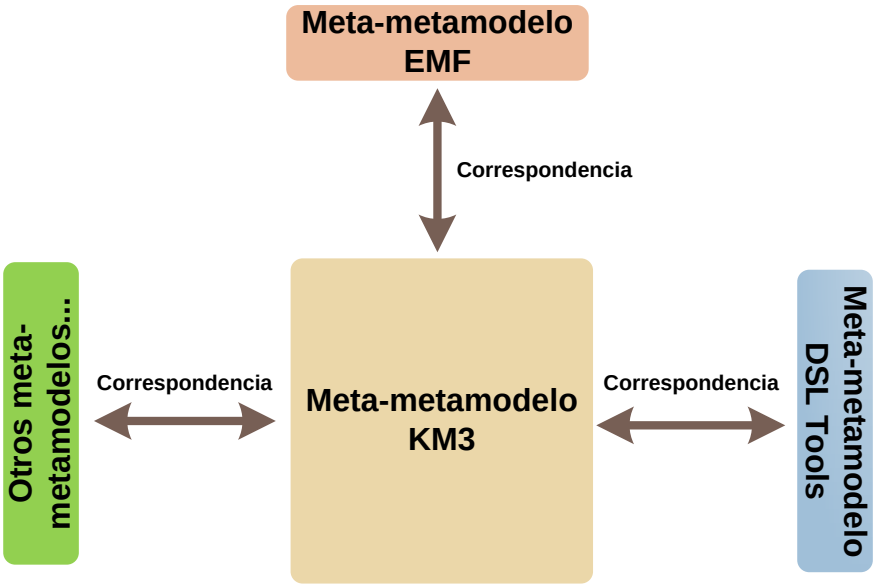


Figura 7.5: KM3 como meta-metamodelo puente

basados en aplicaciones CRUD (Create, Read, Update and Delete). Sin embargo, continuamente se sigue avanzando y dando más potencia a MDA gracias a infinidad de trabajos como [Javed et al., 2007], que muestra como generar casos de prueba a partir de diagramas de secuencia o [Zarras, 2006], que expone como aplicar MDA para la distribución de transparencias.

En la Fig. 7.4 puede observarse la arquitectura general de TMDE (TALISMAN MDE), la iniciativa propuesta en [García-Díaz et al., 2010a]. La idea básica es aprovecharse de las principales ventajas de MDA (portabilidad, interoperabilidad y reusabilidad) y de las principales ventajas de las Software Factories (eficiencia, calidad, abstracción y completud). Se emplean los estándares promovidos por MDA pero también se emplean DSLs no derivados de MOF. Este hecho se consigue gracias a correspondencias o mapeos entre diferentes meta-metamodelos como los propuestos en [Bézivin et al., 2005b] o [Bézivin et al., 2005a], en los que se utiliza un meta-metamodelo llamado KM3 (Kernel Meta Meta Model), que puede ser utilizado como *pivote* para realizar mapeos entre diferentes meta-metamodelos (Fig. 7.5).

De ese modo, se podrían utilizar, por ejemplo, las DSL Tools para crear un DSL específico y con un mapeo entre las DSL Tools y KM3 (ejemplo en Fig. 7.6), se podrían aprovechar todos los demás mapeos ya creados de otros meta-metamodelos a KM3 (p.e., Ecore).

Básicamente, la desventaja de esta iniciativa es que requiere un paso más que MDA pero al mismo tiempo ofrece las ventajas de MDA y de las Software Factories.

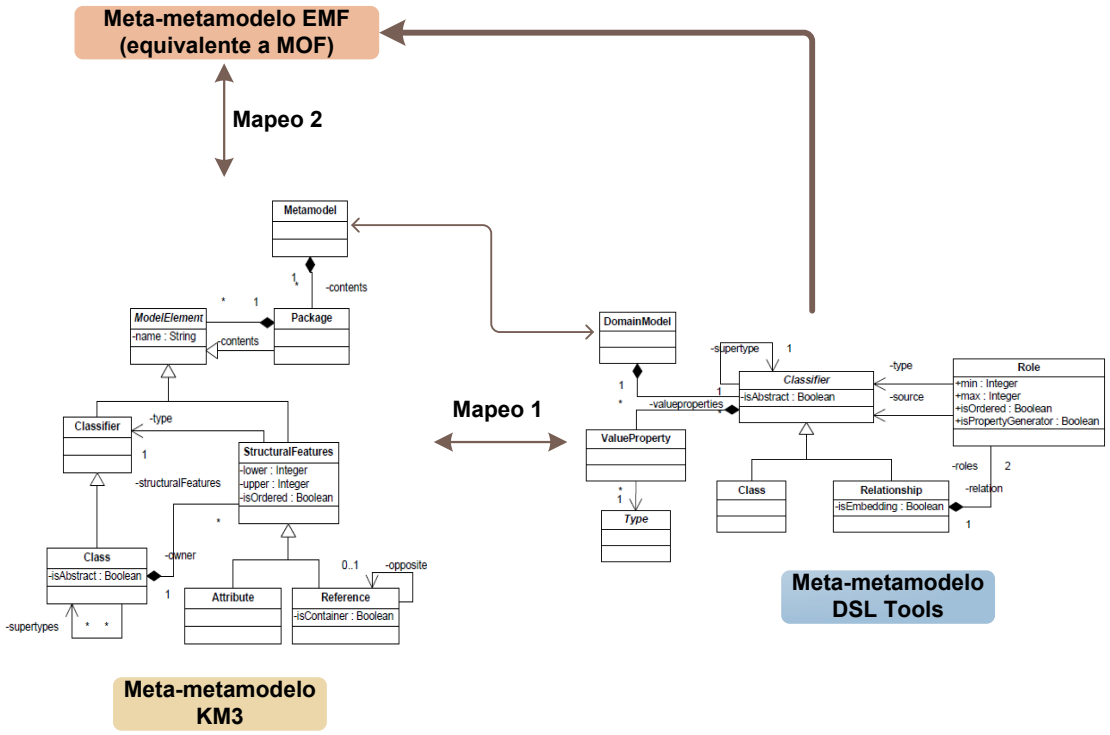


Figura 7.6: KM3 y DSL Tools

La Fig. 7.7 muestra la diferencia entre directamente utilizar UML (perfiles UML en este caso) o utilizar un DSL en un paso previo como se propone en este trabajo. La realidad es que un DSL puede ser mucho más entendible por humanos y tiene un mayor nivel de abstracción. Además, dado que ambas imágenes representan lo mismo, se pueden crear, transparentemente para los usuarios, mapeos utilizando KM3 u otros lenguajes de transformación o plantillas.

La novedad del trabajo es que TMDE tiene todos los beneficios de productividad ofrecidos por las Software Factories pero no todo el sistema está directamente relacionado con una plataforma específica porque los artefactos esquemáticos se abstraen de la plataforma utilizando la arquitectura de cuatro capas MDA [Miller et al., 2003]. Por lo tanto, si en el futuro alguien decide migrar el sistema a otra plataforma, sólo sería necesario cambiar una pequeña parte. Además, basándose en el estándar MOF [OMG, 2005b], se incrementa la interoperabilidad debido a que la integración con otros metamodelos como por ejemplo el ODM (Ontology Definition Metamodel) [OMG, 2005d], que relaciona modelos UML con OWL (Web Ontology Language)[3].

Por otro lado, TMDE tiene la desventaja respecto a las Software Factories del

[3]http://www.w3.org/TR/owl-features/

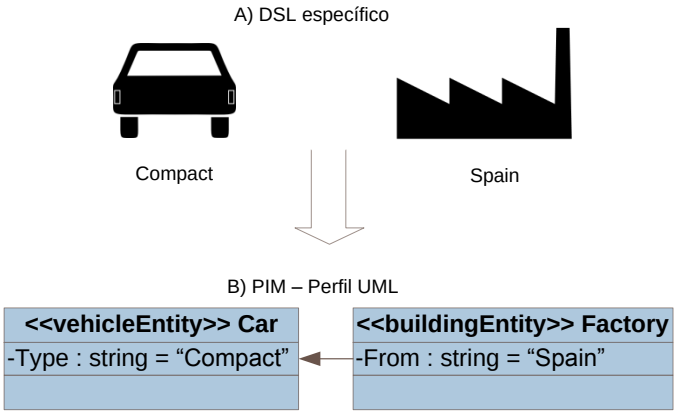


Figura 7.7: DSL versus Perfiles UML

Iniciativa de desarrollo / *Atributos*	Desarrollo tradicional	MDA	SF	TMDE
Interoperabilidad	✗	✓	✗	~
Portabilidad	✗	✓	✗	~
Reusabilidad	✗	✓	✗	~
Eficiencia	✗	~	✓	✓
Calidad	~	~	✓	✓
Abstracción	✗	~	✓	✓
Completud	✓	✗	✓	✓

✓ Característica ofrecida
~ Característica parcialmente ofrecida
✗ Característica no ofrecida

Figura 7.8: Comparativa de los procesos de desarrollo de software

incremento de la complejidad de los generadores debido a la necesidad de utilizar las transformaciones que promociona MDA en lugar de crear DSLs pensados directamente para entornos específicos como por ejemplo hacen las DSL Tools [Cook et al., 2007] o MetaEdit+[4]. Otro aspecto es que desde un punto de vista MDA puro, no todo el sistema es portable, reusable e interoperable.

Dadas las características más destacadas de cada iniciativa de desarrollo de software, se puede hacer una comparación punto por punto (Fig 7.8). Como se ve, dependiendo de las diferentes características, a veces MDA es más fuerte que las Software Factories y en ocasiones ocurre lo contrario. También se observa que TMDE ofrece, posiblemente, los mejores resultados en la comparativa.

- *Interoperabilidad*. MDA, utilizando estándares y puentes entre ellos, permite

[4]http://www.metacase.com/

que los diferentes sistemas puedan interactuar sin una excesiva dificultad. Por ejemplo, el proyecto PRAXIS [Karakoidas et al., 2004] describe la forma mediante la cual se crean mapeos de alto nivel orientados a la interoperabilidad, permitiendo así desarrollar aplicaciones empresariales interoperables utilizando MDE.

- *Portabilidad*. El PIM de MDA es completamente independiente de la plataforma para la cual se generan los artefactos finales. Por lo tanto, para portar un software a otra plataforma sólo es necesario crear nuevamente el PSM y el ISM. [Kleppe et al., 2003] explica cómo se puede incrementar la portabilidad utilizando MDA.

- *Reusabilidad*. Como el PIM es independiente de la plataforma, puede ser utilizado para diferentes desarrollos. Incluso se pueden reutilizar otros mapeos que ya hayan sido creados previamente. En [quan QIAO et al., 2007], los autores mejoran la reusabilidad de las aplicaciones a través de la separación entre modelos y las tecnologías de implementación del sistema.

- *Eficiencia*. Las Software Factories intentan conseguir la máxima productividad creando líneas de productos software para un dominio específico de desarrollo mediante el empleo de DSLs con un alto nivel de abstracción, apoyándose sobre un framework arquitectónico especialmente pensado para una plataforma específica. Por lo tanto, ofrecen una mayor eficiencia en la generación de sistemas software, como explica [Pham et al., 2007].

- *Calidad*. Los frameworks arquitectónicos utilizados por las Software Factories suelen utilizar una serie de *mejores prácticas* para un dominio y una plataforma particular. Así, es mucho más sencillo y rápido conseguir una mayor calidad en los desarrollos. Además, como se describe en [Clements and Northrop, 2002], las líneas de productos software incrementan la productividad y la calidad del software.

- *Abstracción*. El empleo de DSLs no directamente relacionados con un estándar de OMG, aumenta las posibilidades de crear DSLs con un nivel de abstracción mayor debido a que habrá menos restricciones para su construcción. Por lo tanto, se puede decir que las Software Factories pueden tener un mayor nivel de abstracción que MDA [Cook and Kent, 2008].

- *Completud*. Las Software Factories, al igual que los desarrollos tradicionales, crean soluciones completas listas para ser utilizadas. MDA es diferente porque

no siempre es posible generar todo el código de un sistema utilizando mapeos entre CIM, PIM, PSM y ISM. Además, como dice [Greenfield and Short, 2004], MDA únicamente trata con modelos mientras que las Software Factories son una metodología de desarrollo completa.

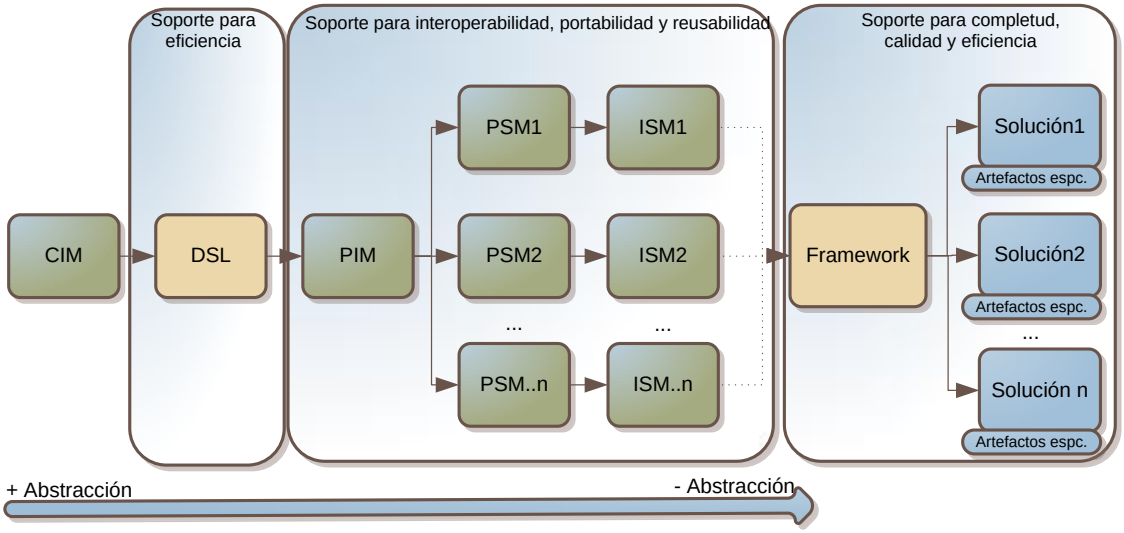


Figura 7.9: Desarrollo con TMDE

La Fig. 7.9 muestra las diferentes características y como se reflejan en TMDE. TMDE ofrece todas las ventajas de las Software Factories y al mismo tiempo las de MDA debido a que se realiza un mapeo entre un DSL y el metamodelo MOF. Así, una gran cantidad del código puede interactuar con los estándares promovidos por OMG, consiguiendo utilizar la enorme cantidad de herramientas que han sido ya creadas para dichos estándares. Sin embargo, y como ya se introdujo, la interoperabilidad, portabilidad y reusabilidad no es tan completa como en el caso de utilizar únicamente MDA debido a que con TMDE se dan esas características únicamente en los artefactos esquemáticos repetitivos. Para el resto de características, TMDE se comporta exactamente igual que las Software Factories. Se utiliza esta estrategia porque MDA tiene algunos inconvenientes:

- El nivel de abstracción de los modelos es menor.
- Utilizando una estrategia MDA pura no se pueden crear aplicaciones muy complejas.
- La falta de un framework arquitectónico reduce el nivel de productividad.

7.2.1. Aspectos relacionados con las Software Factories

El principal objetivo de las Software Factories es el desarrollo de líneas de productos software (véase capítulo 6). Por esa razón, se tienen en cuenta los más importantes conceptos relacionados con ellas, que son:

- *Alcance*. Relacionado con artefactos genéricos, que son implementados con un framework arquitectónico.
- *Variabilidad*. Relacionado con artefactos esquemáticos repetitivos, que son implementados mediante un DSL.
- *Extensibilidad*. Relacionado con artefactos específicos para un determinado desarrollo, que son implementados mediante puntos de extensión utilizados para añadir, modificar o eliminar características.

Estas ideas, adoptadas por TMDE, no están ligadas con ningún entorno de desarrollo y siguen el mismo proceso que el citado en [Greenfield et al., 2004].

7.2.2. Aspectos relacionados con MDA

Además de las Software Factories, no se quiso ignorar la oportunidad de utilizar la filosofía MDA y así, obtener los beneficios de su empleo. Es decir, incremento de la portabilidad, interoperabilidad y reusabilidad de sistemas. Todas estas características son muy difíciles de obtener. Sin embargo, la promesa de MDA es muy interesante y el uso de un estándar como UML es claramente beneficioso. Por ello, con TMDE siempre se realiza un mapeo entre el DSL utilizado y MOF, cualquiera que sea la tecnología empleada para su construcción.

Resumiendo, TMDE utiliza los principios de las Software Factories para proporcionar un desarrollo dirigido por modelos capaz de desarrollar aplicaciones automáticamente para una línea de productos software, pero al mismo tiempo utiliza los principios de MDA para evitar la dependencia total en una plataforma específica, permitiendo que partes del sistema puedan ser generadas a partir de UML o de un perfil UML.

7.3. Caso de estudio

En esta sección se muestra un caso real de un generador de sistemas software de trazabilidad alimentaria desarrollado siguiendo los principios de TMDE, que muestra sus ideas y como podrían ser aplicadas en otros dominios.

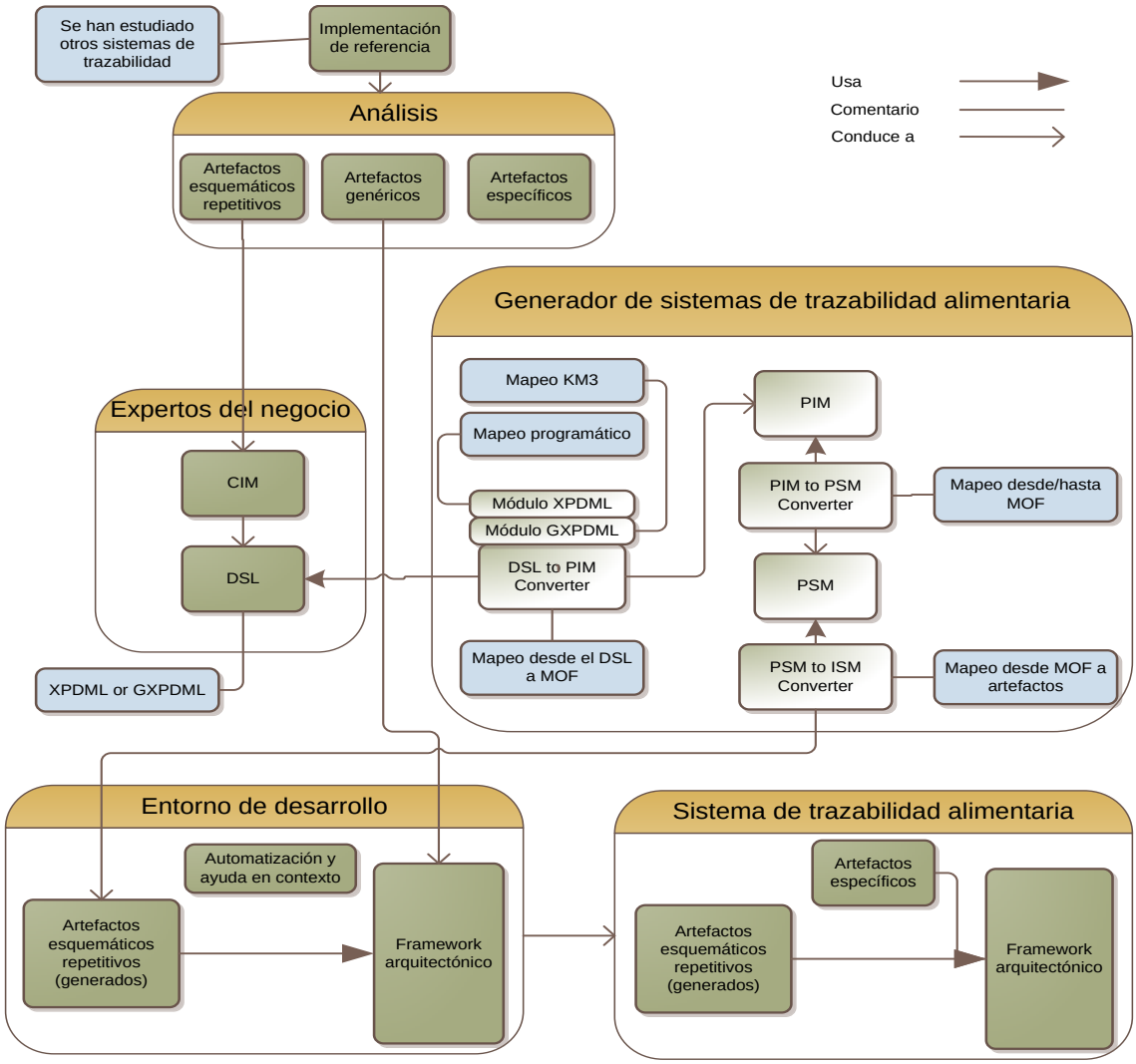


Figura 7.10: Sistemas de trazabilidad alimentaria utilizando TMDE

La Fig. 7.10 muestra el proceso de desarrollo. Inicialmente se han realizado varias visitas a diversas queserías y se han estudiado otros sistemas de trazabilidad alimentaria para tener una base por la que comenzar.

Utilizando un modelo de características [Kang et al., 1990], se han seleccionado las partes que pueden forman el conjunto de artefactos esquemáticos repetitivos y el conjunto de los artefactos que forman parte del framework arquitectónico. Para modelar la variabilidad se ha desarrollado un DSL, que tiene dos variantes: XPDML (eXtensible Process Definition Markup Language) y GXPDML (Graphical XPDML), realizado implementando el metamodelo de XPDML con las DSL Tools [Cook et al., 2007] para obtener así un DSL gráfico. La idea es introducir en los archivos XPDML/GXPDML información independiente de la plataforma y

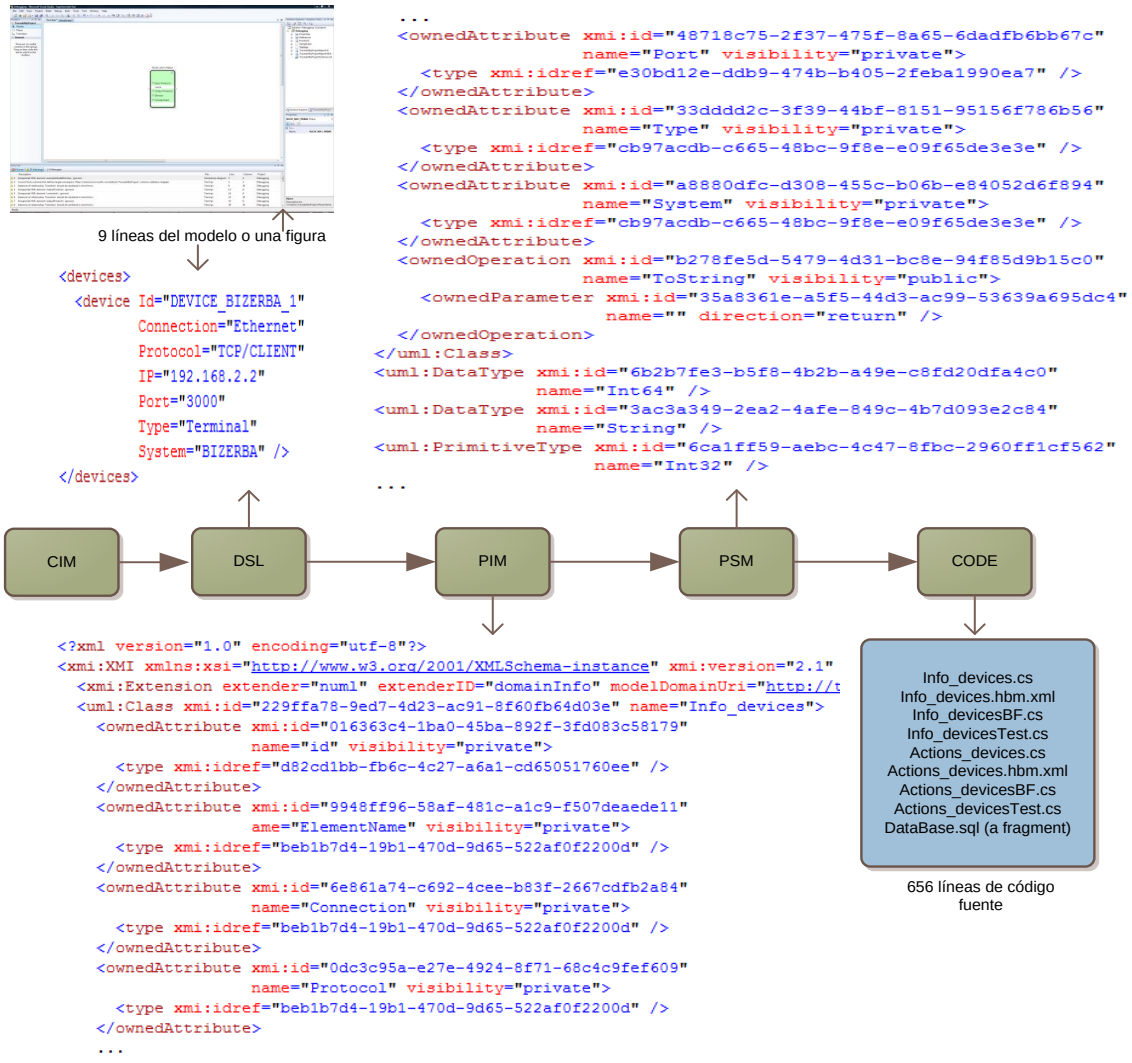


Figura 7.11: Transformaciones en el sistema de trazabilidad alimentaria

empezar el proceso de transformación.

Por ejemplo, el framework arquitectónico realizado para la primera instalación de una quesería del queso Cabrales tenía 166 archivos, más otros 23 de artefactos específicos. Una vez realizadas todas las transformaciones, el número total de archivos fue de 408.

Es muy importante el DSL, ya que utilizando un nivel de abstracción muy alto y una semántica muy cercana al dominio de la trazabilidad alimentaria, hace posible generar artefactos esquemáticos.

Respecto al lenguaje XPDML/GPXML para definir los procesos de fabricación, hay otras alternativas como BPM (Business Process Modeling) [Fernández-Fernández et al., 2008, Fernández-Fernández et al., 2009] pero como se indica en

[García-Díaz et al., 2010a], no son plenamente viables por no tener tan alto nivel de abstracción y por utilizar conceptos diferentes a los de los productores de alimentos. Además, trabajos como [Fernández-Fernández et al., 2010] hacen ver que la notación estándar BPMN (Business Process Modeling Notation) podría ser muy mejorable respecto a su usabilidad.

Con TMDE, hay un número de transformaciones que conducen a la solución. Lo primero es que el archivo XPDML/GXPDML se mapea, reduciendo su nivel de abstracción, al PIM de MDA. Esta es la transformación más compleja, dado que el resto se basan en estándares de OMG y por tanto, son facilitadas gracias a la gran cantidad de herramientas disponibles para ellos como por ejemplo el Eclipse Modeling Project [Gronback, 2009], nUML[5] o ExpertCoder[6].

Cuando la entrada es XPDML, la transformación se hace programáticamente utilizando el archivo XML. Sin embargo, cuando la entrada es GXPDML, se hace uso de la existencia de transformaciones DSL Tools-KM3-Ecore utilizando ATL (Atlas Transformation Language) [Bézivin et al., 2003].

La Fig. 7.11 muestra uno de los casos más simples de transformaciones desde el DSL. Como se observa, se especifican aspectos como el dispositivo hardware, la conexión Ethernet, el protocolo, la IP, el puerto, el tipo y el fabricante. El generador hace algunos mapeos internos desde XPDML/GPDML a XMI [OMG, 2005b]. Este documento XMI se corresponde con el PIM. Posteriormente, el PIM se *decora* con información que depende de la plataforma destino (p.e., el método *ToString*, que sobrescribe el método de la clase *Object* de .NET). El último paso es bastante directo gracias al uso de lenguajes de plantillas como Xpand[7].

7.4. Conclusiones

La mayoría de herramientas actuales para trabajar con MDE están diseñadas para trabajar con metamodelos MOF. Esto hace que las herramientas sean más interoperables, evitando una de las causas que según Steve Cook [Cook, 2004] podrían hacer fracasar a MDA. Estas herramientas incluyen a los generadores de código dirigidos por modelos y a los sistemas de control de versiones para modelos, ambos clave en MDE.

Pese a todo, sigue siendo interesante utilizar iniciativas diferentes a MDA para trabajar con líneas de producción y DSLs con mayor nivel de abstracción, creados

[5] http://numl.sourceforge.net/
[6] http://expertcoder.sourceforge.net/
[7] http://wiki.eclipse.org/Xpand/

con otras herramientas y facilitando así la labor de los usuarios de las herramientas de modelado.

En trabajos como [Bézivin et al., 2005b, Bézivin et al., 2005a] se dan claves para crear puentes entre meta-metamodelos, de tal forma que se pueden crear metamodelos que no sigan el estándar MOF para que sean utilizados por los usuarios. Después, transparentemente para el usuario de la herramienta de modelado, se pueden realizar transformaciones para que los modelos creados puedan ser consumidos por las herramientas que siguen las directrices de OMG, promotores de MDA y MOF.

Otros trabajos como [Tolosa et al., 2009, Tolosa et al., 2010] focalizan en la creación de taxonomías y métricas para ayudar a conocer las características de las transformaciones entre modelos y metamodelos, así como para ayudar a mejorar la calidad de las mismas.

Por todo ello, lo más razonable es la combinación de las dos iniciativas más clásicas, creando una parte del sistema más estática mediante el empleo de las mejores prácticas para una plataforma concreta y creando las partes esquemáticas repetitivas utilizando las capas de MDA a partir de un DSL de alto nivel de abstracción.

Este es el camino para conseguir la máxima productividad con la máxima posible interoperabilidad, portabilidad y reusabilidad. Además, como se observa en [García-Díaz et al., 2010a], la cantidad de código fuente necesario para trabajar con las Software Factories y con TMDE es idéntico.

En resumen, los modelos con los que se trabajará en el resto del trabajo de esta Tesis serán conformes al estándar MOF, bien directamente o bien realizando un mapeo previo. Así, se podrán utilizar todas las herramientas creadas que siguen los estándares, sin descartar el uso de otras que puedan ser de utilidad. TMDE se podrá aplicar en el futuro a trabajos que actualmente se están llevando a cabo como por ejemplo [Palacios-González et al., 2009, Montenegro-Marín et al., 2011, Martínez et al., 2011b].

Bloque IV

Detección de nuevas versiones en los VCSs

Índice del bloque

CAPÍTULO 8

Problemática

El control de versiones (también denominado control de revisiones, control de fuentes, gestión del código fuente o gestión de la configuración del software) da lugar a los VCSs (Version Control Systems o Sistemas de Control de Versiones) y consiste en la gestión de diferentes revisiones de una misma unidad de información.

Generalmente, se utilizan en el desarrollo de software para lograr guardar los diferentes estados por los que pasan las fuentes del desarrollo. Sin embargo, también se utilizan para otros activos como documentos de requisitos o imágenes. Las diferentes revisiones se identifican normalmente por medio del incremento de algún valor o código asociado con la persona que ha hecho el cambio.

Los VCSs suelen ser aplicaciones software independientes, que se pueden integrar en otros sistemas por medio de extensiones (p.e., Subclipse[1]). Sin embargo, existen multitud de aplicaciones que incorporan sus propios VCSs (p.e., Google Docs, MediaWiki, etc.).

En este capítulo se mostrará la importancia que tienen los sistemas de control de versiones en los, cada día, más heterogéneos desarrollos de software. Se presentan las principales características de dichos sistemas, las propuestas que actualmente existen para desarrollos bajo el paradigma MDE y se profundizará en la detección de diferentes versiones de un modelo, aspecto clave que, idealmente, deben incorporar todos los VCSs para trabajar correctamente con MDE[2].

* * * *

[1] http://subclipse.tigris.org/

[2] Para evitar no detectar una nueva versión de un modelo, o detectar como nueva versión de un modelo algo que en realidad no lo es

8.1. Importancia de los sistemas de control de versiones

Como la construcción del software se realiza en varias etapas (típicamente análisis, diseño, desarrollo, pruebas y despliegue), es común que diferentes versiones de software estén desplegadas en diferentes equipos y que los desarrolladores trabajen simultáneamente en nuevas mejoras, en nuevas versiones y en la corrección de errores de determinadas versiones (debido a que con la evolución del software se solucionan algunos problemas pero se crean otros). Por todo ello, es muy importante que se tenga completamente localizado todo el historial de versiones para determinar el contenido de cada una de ellas.

Los desarrolladores podrían guardar copias locales ordenadas por número de versión y trabajar con ellas sin necesidad de un VCS, pero para ello se requiere una gran disciplina. Además, a menudo se producen errores humanos que producen pérdida de información o mezcla de versiones.

Por otra parte, en los desarrollos de software, cada día más grandes, es fundamental el trabajo en equipo, y por ello se hace necesario un sistema capaz de gestionar los cambios de los diferentes miembros del equipo de desarrollo de manera transparente para ellos, así como guardar un registro con información de quién ha hecho cada modificación.

Es evidente que en los últimos años ha nacido un gran mercado para proporcionar soluciones que faciliten el control de versiones y que muchas de la tecnologías utilizadas en dichas soluciones son innovadoras en el sentido de que se están publicando una gran cantidad de artículos en congresos y revistas científicas (p.e., [Oliveira et al., 2005, Altmanninger, 2007]). Incluso OMG ha liberado una especificación para unificar el versionado de artefactos MOF [OMG, 2007a]. Sin embargo, todavía hay una falta de aproximaciones para construir herramientas relacionadas con el versionado de modelos [Altmanninger et al., 2009]. Además, el problema se ve agravado por el hecho de que cada herramienta debe considerar la semántica de cada modelo particular [Schmidt and Gloetzner, 2008].

8.2. Características principales de los VCSs

Todos los VCSs tienen algunos aspectos comunes:

8.2.1. Funcionalidades

Desde sus orígenes con [Tichy, 1985], se han identificado las siguientes necesidades que todos los VCSs han de satisfacer:

- Capacidad para almacenar los diferentes tipos de activos que se van a manejar (p.e., texto, modelos, música).
- Capacidad para realizar cambios sobre activos almacenados.
- Capacidad para registrar todo el historial de cambios de los diferentes activos. Además, es conveniente que exista la opción de volver a un estado anterior en el tiempo.
- Capacidad para generar informes con los cambios entre versiones.

8.2.2. Clasificación

Según la forma en la que se almacena el código, existen principalmente dos tipos de VCSs:

- Centralizados. Se utiliza un repositorio centralizado en el que se guardan todos los artefactos. La mayoría de los VCSs son centralizados.
- Distribuidos. Se utiliza un repositorio distribuido. Se da más flexibilidad pero se dificulta la sincronización al tener a los artefactos repartidos por más de una ubicación física. Trabajos como [Korel et al., 1991] son los inicios de este tipo de sistemas.

8.2.3. Gestión de fuentes

Existen dos aproximaciones para gestionar la forma en la que se modifica la copia local de un repositorio [OReilly et al., 2003]:

- Pesimista. Si un usuario quiere trabajar con un activo, lo tiene que bloquear para que ningún otro usuario lo pueda utilizar. Con esta aproximación se evitan conflictos pero se dificulta el trabajo colaborativo.
- Optimista. Permite que múltiples usuarios trabajen sobre el mismo activo. El sistema se encargará de hacer las uniones pertinentes con las diferentes modificaciones. Es muy flexible pero pueden ocurrir conflictos que han de solucionarse manual o automáticamente, con el consiguiente riesgo a producir errores.

8.3. VCSs para iniciativas MDE

Debido a que, tanto los modelos, como las transformaciones entre ellos, se han convertido en activos de primera mano dentro del proceso de desarrollo software, surge la necesidad de introducir actividades fundamentales de la ingeniería del software (como el control de versiones) dentro de los procesos de desarrollo MDE. Así, se asegura que se cumplen con los principios de mejores prácticas software [Hnětynka and Plášil, 2004].

Los estándares en los que se basan los VCSs recomiendan la identificación de dos elementos:

- UV (Unit of Versioning o Unidad de Versión), que es la unidad atómica de la que se desea guardar el historial de modificaciones.
- UC (Unit of Comparisong o Unidad de Control), que es la unidad atómica utilizada para detectar conflictos cuando dos o más entidades están manipulando un mismo elemento.

Se podría pensar en utilizar VCSs tradicionales para trabajar con modelos; pero lo cierto es que dichos sistemas no son adecuados al estar basados en una UV a nivel de fichero y una UC a nivel de línea de fichero[3], estructuras no válidas para modelos [Oliveira et al., 2005]. El motivo es que son necesarias abstracciones a nivel de grafo debido a que los modelos son representados por éstos [Altmanninger, 2007].

En [Oliveira et al., 2005] se muestra cómo funcionan las UVs y UCs con los VCSs tradicionales (Fig. 8.1). En el caso del texto plano, los VCSs tradicionales funcionan bien porque generalmente un texto plano (p.e., una carta) está contenido en un único fichero (UV), siendo la UC un párrafo, que generalmente tiene una alta cohesión y un relativo bajo acoplamiento con otros párrafos. En el caso de lenguajes orientados a objetos, si se utilizara la línea como UC entonces se estaría utilizando como UC una estructura que no existe en dichos lenguajes ya que por ejemplo un método estaría formado por más de una línea y en una línea se podrían escribir varias sentencias de un lenguaje. Además, una sentencia suele tener alto acoplamiento con otras sentencias. La UV también se correspondería con estructuras que no pertenecen en dichos lenguajes, ya que en un mismo fichero pueden coexistir varias clases diferentes y un paquete generalmente está formado por varios ficheros.

Por último, prácticamente la totalidad de modelos pueden ser serializados a formato XML (p.e., los modelos UML se serializan a formato XMI). Sin embargo,

[3]Los VCSs más sofisticados permiten variar la UC bajo ciertos supuestos pero la UV siempre se trata a nivel de fichero

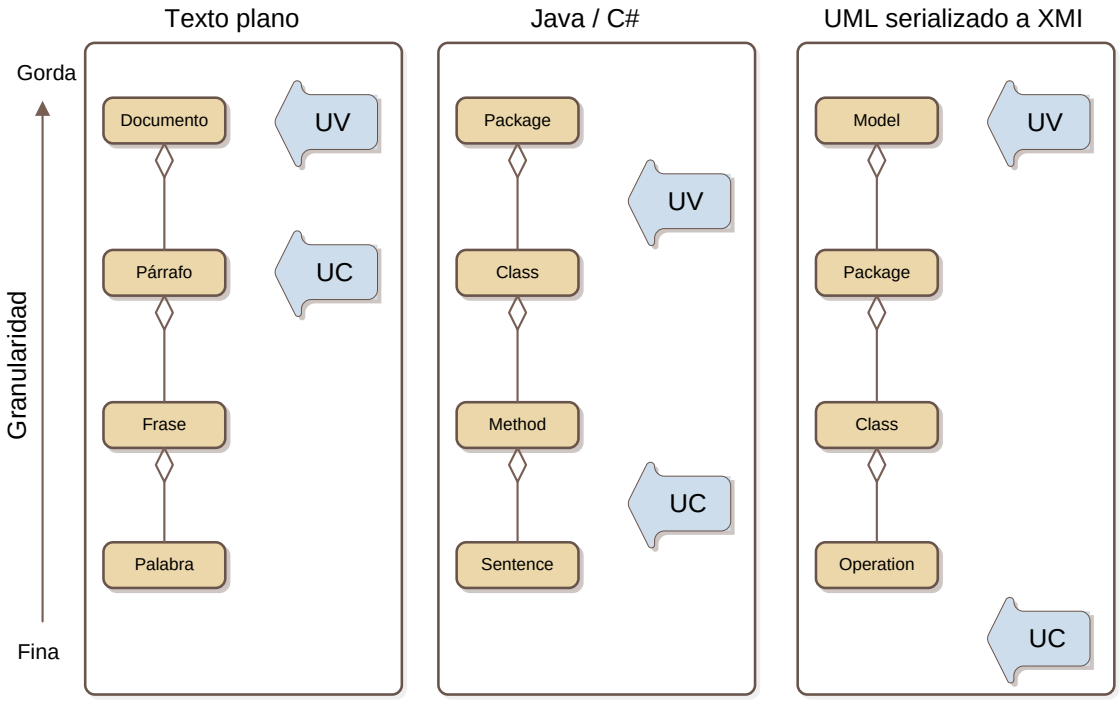


Figura 8.1: UV y UC en VCSs tradicionales

un modelo puede contener cientos o miles de elementos, considerándose todos ellos como la UV. Eso, unido a que la UC se corresponde con un elemento que no existe en dichos modelos serializados (una operación se representa con varias líneas que indican el tipo de retorno, los argumentos, la visibilidad, etc.), hacen que los VCSs tradicionales sean muy poco adecuados para trabajar con modelos: sería mucho más adecuado poder elegir la UV y la UC en función de las necesidades de cada proyecto.

En [Altmanninger, 2007] también se comenta la necesidad de mejorar la detección de conflictos entre modelos (evitar la detección de falsos conflictos sintácticos y detectar conflictos semánticos que no habían sido detectados). Existen diferentes formas de afrontar dicho problema [Harel and Rumpe, 2004], pero destaca sobre todas ellas el trabajo presentado en [Reiter et al., 2007]: se utiliza un procedimiento diferente, en el que la comprobación en el modelo no se hace a nivel sintáctico y sí a nivel semántico mediante la utilización de las llamadas *vistas de interés* que eliminan el *azúcar sintáctico* de los modelos.

Dado que un modelo es la instancia de un metamodelo, una vista semántica de dicho modelo se crea realizando un segundo metamodelo con información semántica a partir del metamodelo original. De ese modo, realizando una transformación con el modelo original, se obtiene un nuevo modelo, es decir, una vista del modelo original.

El ejemplo de la Fig. 8.2 fue obtenido de [Reiter et al., 2007]. En dicho ejemplo

se muestra una versión simplificada del metamodelo de BPEL[4] (Business Process Execution Language), que muestra construcciones con azúcar sintáctico, al mismo tiempo que se ilustran cuatro escenarios en los que dos desarrolladores se disponen a hacer modificaciones concurrentemente en los modelos. En color *naranja* están los modelos según se obtienen del repositorio, en color *rojo* están las modificaciones de un desarrollador y en color *azul* las de otro. Gracias a la vista de interés se detecta un conflicto que de otra forma no se hubiera detectado (caso B). Además, se evita la detección de un falso conflicto (caso C).

8.4. Problemas de los VCSs para modelos con gestión optimista

Cuando se utiliza una gestión de fuentes optimista[5], varios autores pueden modificar simultáneamente un mismo modelo. Esto da lugar a un conjunto de actividades secuenciales que han de ejecutarse para obtener un nuevo modelo a partir de los diferentes cambios realizados simultáneamente. Habrá que *detectar* las diferencias, *resolver* dichas diferencias y posteriormente *mezclar* las diferencias en un único modelo [Reddy and France, 2005].

Los MVCSs (Model Version Control System o Sistemas de Control de Versiones para Modelos) actuales tienen problemas en aspectos relacionados directamente con el ciclo mencionado previamente [Altmanninger et al., 2008, Altmanninger et al., 2009]:

8.4.1. Detección de conflictos

Los conflictos ocurren cuando hay variaciones realizadas al mismo tiempo sobre un modelo dentro de la misma UC. En el mejor de los casos, las variaciones son equivalentes, y por lo tanto no provocan problemas a la hora de unir el trabajo realizado. Sin embargo, ocurre con mucha frecuencia que las modificaciones realizadas sobre el modelo sean contradictorias. Además, en muchas ocasiones se identifican conflictos que realmente no lo son y no menos problemático es no identificar conflictos que realmente sí se han producido. Como ejemplos de conflictos podrían citarse la realización simultánea de cambios sobre un mismo elemento, o cuando se borra un elemento y, al mismo tiempo, se hace un cambio sobre el elemento.

[4] http://www.bpelsource.com/

[5] Los VCSs pesimistas, dada su naturaleza, no tienen los problemas que sí tienen los VCSs optimistas

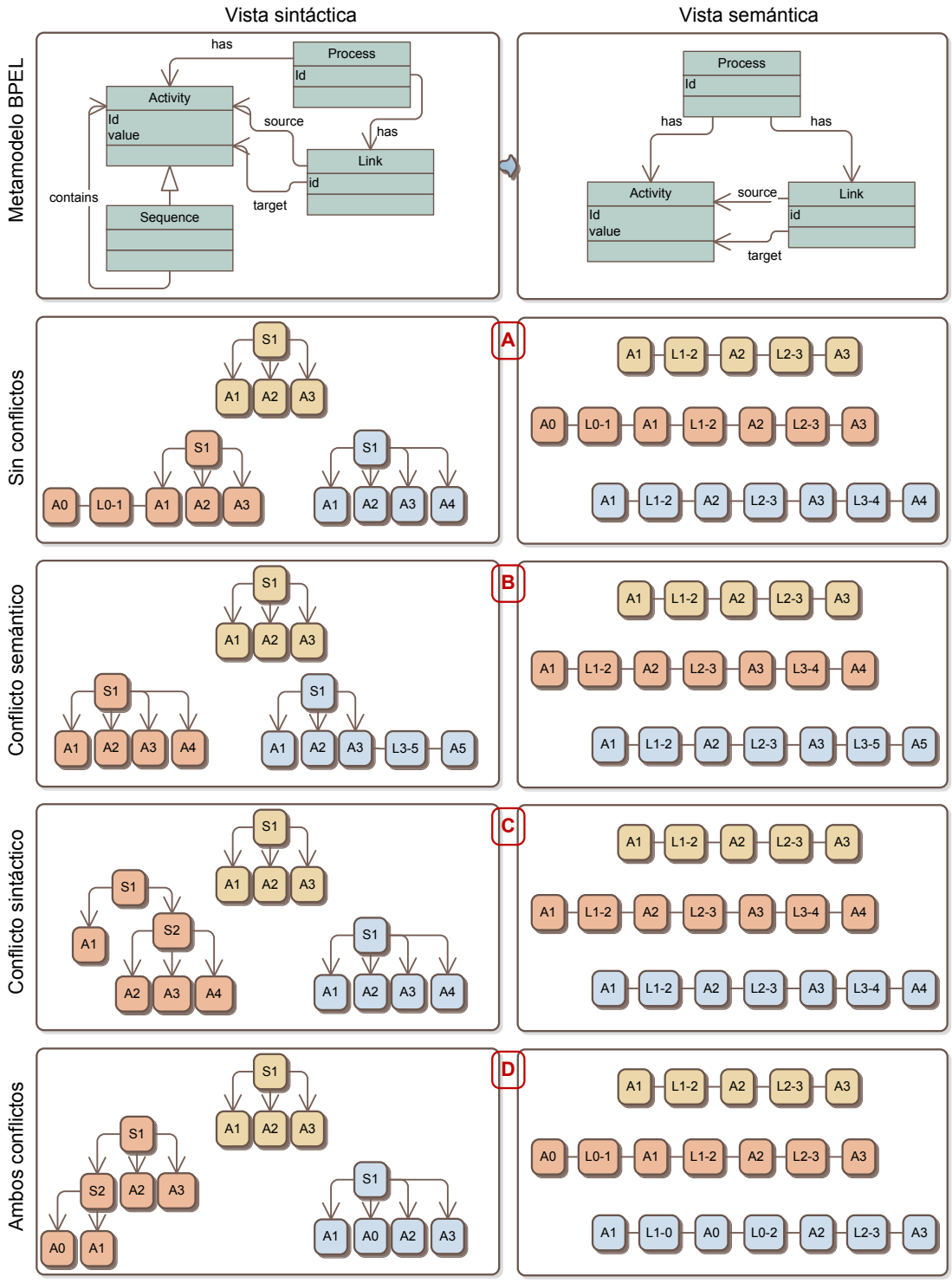


Figura 8.2: Detección de conflictos mediante vistas de interés

8.4.2. Resolución de conflictos errónea

En la mayoría de los casos, la elección de la estrategia de resolución de conflictos tiene que ser realizada manualmente por los usuarios, lo cual es una tarea propensa a errores.

Si el modelo resultante no conforma con el metamodelo del DSL o no cumple con alguna regla de validación (p.e., clases UML que heredan de otras correctamente por separado, pero que una vez unidas crean un ciclo), surgen problemas de inconsistencia sintáctica en la versión mezclada de los modelos tras detectar y resolver los conflictos.

Además de los problemas sintácticos, hay que tener en cuenta los semánticos (p.e., si un modelador introduce una propiedad en una clase de la que heredan otras y el otro modelador introduce una nueva subclase sin ser consciente de que el otro ha introducido una nueva propiedad en la superclase, entonces el modelo mezclado podría ser correcto sintácticamente, pero no semánticamente).

8.4.3. VCSs poco flexibles

Los VCSs para modelos actuales se caracterizan por su poca flexibilidad:

- Existen VCSs muy genéricos que sirven para todo tipo de artefactos pero que se caracterizan por su pobre soporte para la gestión del versionado.
- Existen VCSs con gran soporte para la gestión del versionado pero que están muy sujetos a un determinado tipo de artefacto o a una herramienta específica.

8.5. Elementos a versionar

Dentro del proceso MDE se podría tener versiones de varios elementos, que son clave dentro del proceso de desarrollo de software:

- Las herramientas de generación de código a partir de modelos. Se debe a que la mayoría de herramientas están bajo desarrollo y no sería descartable tenerlas versionadas.
- Los metamodelos que definen al DSL para un dominio determinado.
- Los modelos, instancias de los metamodelos de los DSLs, que contienen la información a partir de la cual se generarán artefactos de más bajo nivel.
- La arquitectura base, que se emplea junto con los artefactos generados a partir de los modelos.

- Los artefactos manuales específicos que pudiera tener un producto individual dentro de una línea de productos software.

Idealmente no habría que tener versiones de los artefactos generados [Völter and Stahl, 2006] debido a que se generan a partir de los elementos que se acaban de citar y por tanto no son una de las fuentes a partir de las cuales se desarrolla el software.

El problema se debe a que en muchas ocasiones hay que integrar nuevo código generado manualmente con el código generado automáticamente, y si no se han separado correctamente ambas partes, habría que tener versiones de todo el código generado porque no sería posible hacerlo de otra forma.

Para realizar el control de versiones hay que tener en cuenta que un cambio de versión de uno de los elementos citados podría acarrear un cambio necesario en otro de los elementos (p.e., si cambia el metamodelo es muy probable que cambie el modelo). Por lo tanto, es conveniente tener almacenadas las referencias de las distintas versiones entre los diferentes elementos.

8.6. Estructura de un proyecto MDE con versiones

En la Fig. 8.3 se pueden observar los elementos que intervienen en un proyecto MDE al que se le aplica un control de versiones.

Típicamente se utilizarían dos repositorios para el proyecto (algunos autores como Martin Fowler [Fowler, 2009a] recomiendan utilizar únicamente uno):

- En uno se introducen los datos meramente de la aplicación (modelos y el código específico que pudiera existir en el desarrollo).
- En el otro se introducen los datos referentes a la arquitectura global de la aplicación (configurador del generador y arquitectura o framework base).

Gracias a los modelos y al configurador, se generan artefactos automáticamente. Posteriormente, el código generado se acopla a la arquitectura base y al código específico del desarrollo.

8.7. Control de versiones cuando se mezclan partes generadas y no generadas

No siempre es posible tener completamente separadas las partes de código generadas de las no generadas. Este hecho acarrea problemas para gestionar las versiones

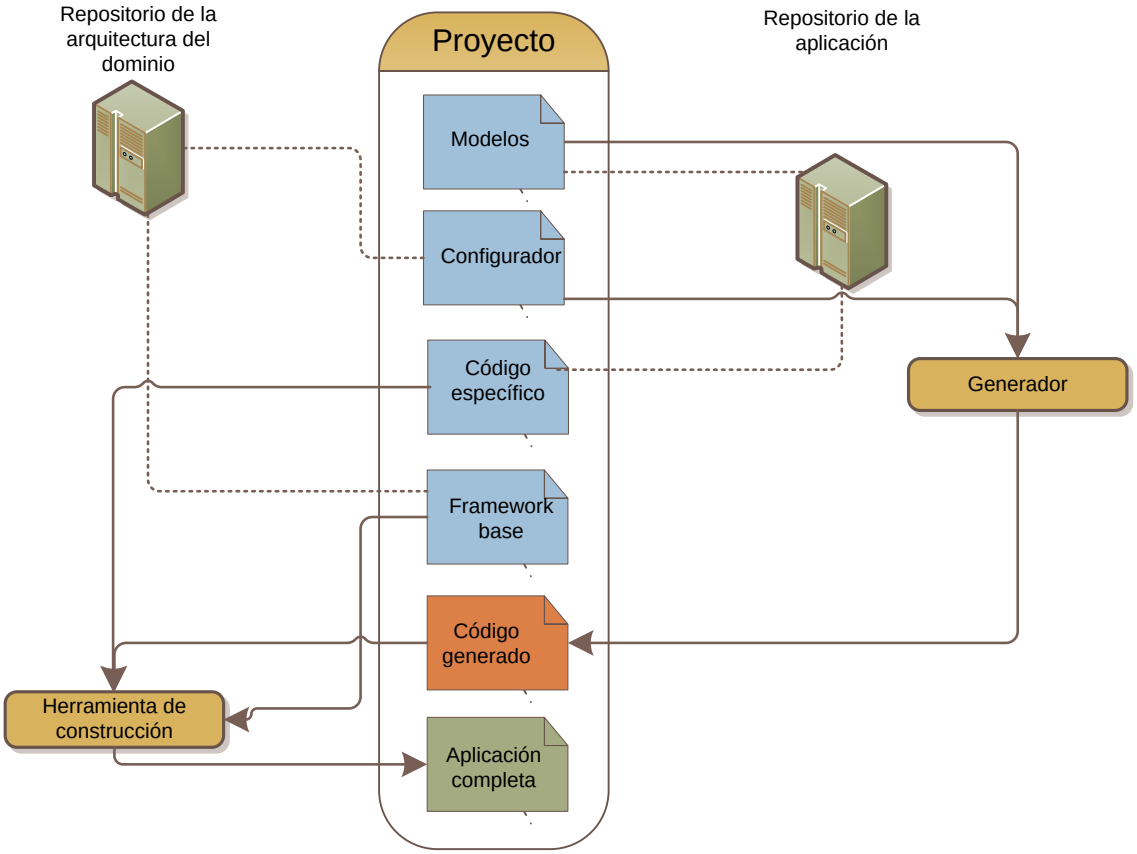


Figura 8.3: Estructura de un proyecto MDE con versiones

aunque idealmente no se tendría por qué realizar un versionado del código generado, pues no es un activo inicial del proceso de desarrollo cuando se trabaja con el paradigma MDE. Cuando se hace necesario unir ambas partes de código, existen mecanismos como las regiones de código protegidas. El proceso consiste en poder definir regiones protegidas para que el generador de código las tenga en cuenta y las preserve gracias a marcas en el código. Estas marcas serán invisibles para el compilador o intérprete de la plataforma en la que se lanzará la aplicación debido a que se realizan con mecanismos como comentarios o atributos en el código fuente.

El uso de regiones de código protegidas tiene como consecuencia la creación de archivos de código en los que se mezclará código generado y código no generado, provocando redundancias en la gestión de versionado porque se versionan elementos que no son activos fuente para el desarrollo. En la Fig. 8.4 se muestra una propuesta [Völter and Stahl, 2006] para reducir las redundancias en el versionado cuando en un archivo se mezclan partes de código generadas y partes de código no generadas.

Las fuentes se almacenan en el repositorio de la aplicación, haciendo distinción

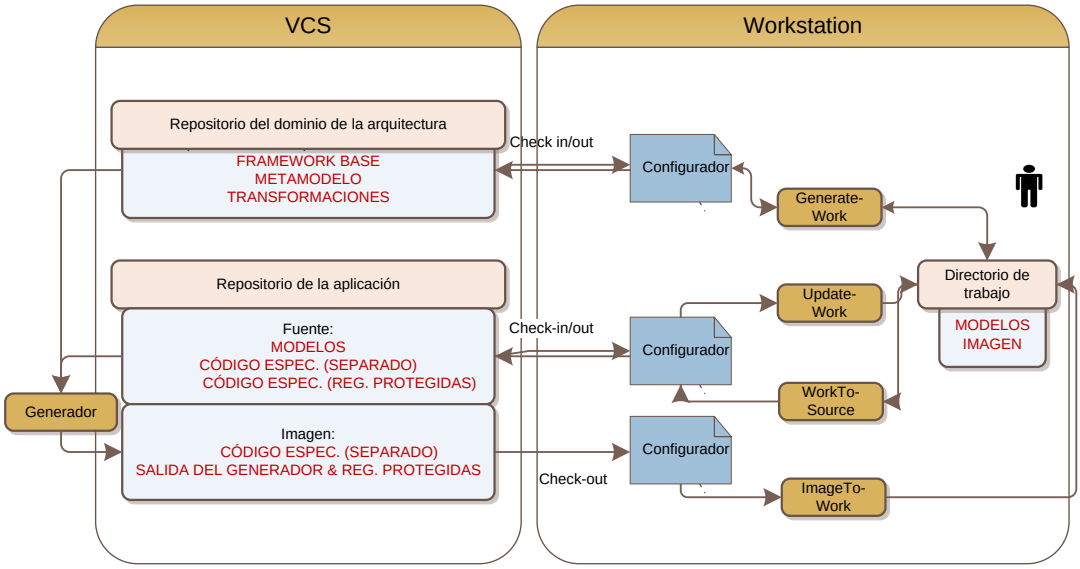


Figura 8.4: Control de versiones con mezcla de código generado y no generado

entre las partes de código específicas generadas por separado y las partes de código específicas que se crean mediante el uso de regiones protegidas.

Frecuentemente se crean imágenes de la aplicación que contienen la parte de código generada y la parte de código creada manualmente. A partir de ese punto, los desarrolladores pueden realizar *check-in* y *check-out* cuando lo deseen para conseguir realizar la sincronización con los repositorios.

El desarrollo se realiza en un directorio de trabajo separado que se comunica con el repositorio por medio de *scripts* que pueden ser los siguientes:

- *GenerateWork*. Produce una imagen completa de la aplicación en el espacio de trabajo del desarrollador.

- *UpdateWork*. Actualiza el código generado manualmente en el espacio de trabajo del desarrollador.

- *WorkToSource*. Introduce en el VCS los cambios que se realizan en el espacio de trabajo del desarrollador.

- *ImageToWork*. Tiene un resultado similar a *GenerateWork*. Actualiza la información del espacio de trabajo del desarrollador.

8.8. Control de versiones en equipos de desarrollo

Los grandes desarrollos de software tienen que ser divididos en módulos para poder ser llevados a cabo con éxito. Este hecho afecta al modo en el que se reparten las diferentes tareas en el desarrollo y como es lógico, también afecta al control de versiones. Hay dos conceptos clave a tener en cuenta para dividir el trabajo:

8.8.1. Partición

Mediante las particiones se describen sistemas parciales o subsistemas que pueden ser integrados mediante el empleo de interfaces (p.e., gestión de usuarios, contabilidad, etc.).

Un ejemplo clásico son dos equipos de desarrollo que necesitan utilizar una misma interfaz; no es buena idea tratar a la interfaz como un elemento independiente en cada herramienta de modelado porque puede dar lugar a fallos de consistencia en el futuro (Fig. 8.5).

En general, nunca es bueno duplicar elementos innecesariamente, ya que existen otras opciones:

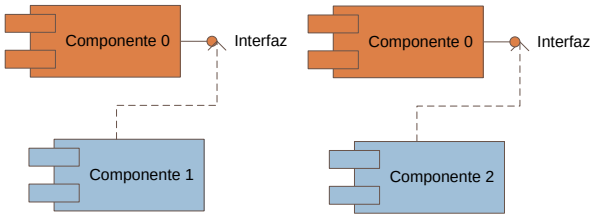


Figura 8.5: Integración con particionado (duplicación de elementos)

- *Integración en el modelo.* Si la herramienta de modelado tiene soporte, es mejor dejar a la interfaz en un único lugar y referenciarla mediante un enlace desde los dos equipos de desarrollo (Fig. 8.6).

- *Integración en el generador con sincronización en el modelo.* Si la herramienta de modelado no da soporte para separar a la interfaz de los modelos, la integración se puede realizar a nivel del generador. Se puede lograr mediante mapeos explícitos o implícitos de elementos (Fig. 8.7).

- *Integración en el generador con referencias.* Otra alternativa es el empleo de referencias (elementos *proxy*). La interfaz sólo está presente en un modelo, el otro modelo simplemente la referencia.

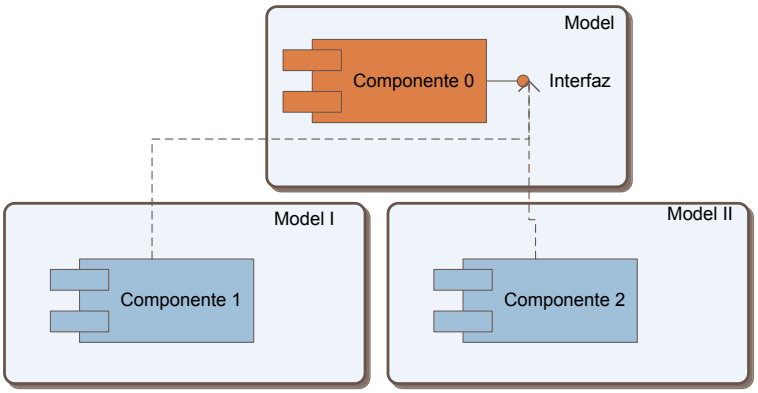


Figura 8.6: Integración con particionado (compartimiento de interfaz)

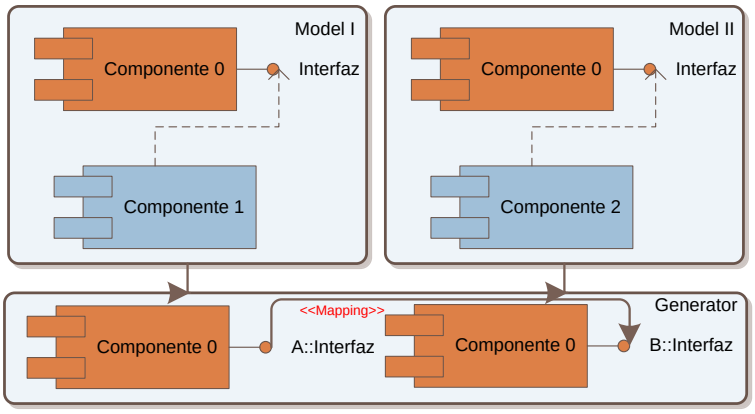


Figura 8.7: Integración con particionado (resolución por el generador)

8.8.2. Subdominio

Mediante subdominios se pueden aislar y separar aspectos técnicos del sistema. Cada subdominio tiene su propio metamodelo (p.e., interfaz de usuario, persistencia, registro de eventos, criptografía, etc.).

8.9. Evolución de los metamodelos

Generalmente, los metamodelos evolucionan con el tiempo y con la experiencia en el dominio de conocimiento; pero es muy importante asegurarse de que mantienen la compatibilidad hacia atrás en el tiempo.

Para lograr la compatibilidad hacia atrás, se puede modificar el generador de artefactos para que soporte todas las versiones de los metamodelos. Hay casos en los que los desarrolladores están trabajando en diferentes versiones a causa de los

necesarios tiempos de transición (no es aconsejable forzar a los usuarios a cambiar la versión de su software en un instante determinado). Se pueden mostrar mensajes de aviso cuando se generan los artefactos, informando de que existe una nueva versión del metamodelo. Existen formas de hacer, lo más sencillo posible, la transición entre diferentes versiones como por ejemplo las citadas en [Völter and Stahl, 2006]:

- Marcando los modelos con la versión del metamodelo que deberá manejar el generador de artefactos.
- Implementando reglas implícitas como por ejemplo, si no existe un atributo utilizar valores por defecto para él.
- Creando nuevos subelementos XML opcionales en el caso de que surjan nuevos atributos. Así, se evita modificar todas las versiones previas.
- Indicando en el atributo XML una clase de un lenguaje de programación que, mediante una interfaz, realice alguna comprobación. Sirve para dar flexibilidad y permitir realizar cambios más fácilmente.
- Utilizando un atributo *deprecated* para marcar elementos ya en desuso. De ese modo, se le puede indicar al generador que hay elementos anticuados que aún se están utilizando.

8.10. Estándar para el versionado de modelos

Debido a la gran importancia que está adquiriendo la gestión del versionado en el software, se mostrará brevemente la especificación que OMG ha presentado: el Meta Object Facility Versioning and Development Lifecycle Specification [OMG, 2007a], que sirve para afrontar el problema de la coexistencia de diferentes versiones de datos MOF. Esta especificación está muy ligada al contexto en el que un usuario hace una petición, debido a que en función a su contexto, los datos se corresponderán con una versión u otra de los artefactos MOF.

En la Fig. 8.8 puede verse un esquema de las dependencias (con otros paquetes definidos por OMG) del paquete del metamodelo creado para afrontar el versionado:

8.11. Detección de versiones de un modelo

Independientemente del uso de una aproximación optimista o pesimista, hay que detectar, como paso previo a los demás, la aparición de nuevas versiones de un modelo. Dicha actividad requiere, al menos, saber los cambios que se han producido

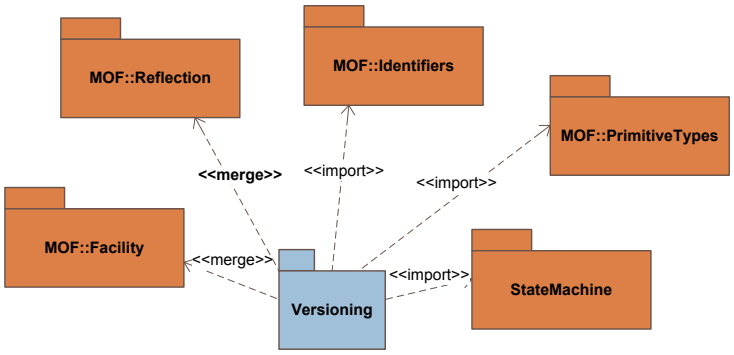


Figura 8.8: MOF Versioning and Development Lifecycle Specification

en un modelo respecto a otro, es decir, requiere conocer las diferencias o variaciones entre los estados por los que pasa un modelo.

Conocer las diferencias entre modelos no es un proceso sencillo [Read and Cornell, 1977] y comprende tres fases [Brun and Pierantonio, 2008] de las que, para detectar nuevas versiones en los MVCs, resulta sobre todo útil la primera de ellas, es decir, el cálculo de las diferencias entre modelos. Las otras dos fases son importantes para otro tipo de actividades, y consisten en la representación de las diferencias entre los modelos y la visualización de tales diferencias en un formato amigable para las personas.

En las próximas líneas se mencionarán los principales contextos en los que se utiliza actualmente el cálculo de la diferencia entre modelos. Posteriormente, se tratarán las técnicas actuales para calcular dichas diferencias.

8.11.1. Ámbitos de aplicación de la diferencia entre modelos

Dentro de la gran cantidad de investigaciones que, relacionadas con MDE, surgen cada año, ha aparecido un número considerable de trabajos enfocados a la necesidad de desarrollar sistemas de control de versiones específicos para modelos [Oliveira et al., 2005]. La idea es suplir las carencias, que respecto a modelos, tienen los sistemas de control de versiones tradicionales [Tichy, 1985].

En trabajos como [Reiter et al., 2007] o [Altmanninger et al., 2008] se trata la gestión de fuentes optimista utilizando modelos, en la que múltiples usuarios trabajan sobre un mismo activo. Por ello, se hace imprescindible el empleo de algoritmos de comparación de modelos para representar sus diferencias y/o realizar uniones entre ellos [Bendix and Emanuelsson, 2008].

Otros trabajos como [Lin et al., 2004] han mostrado la importancia de la repre-

sentación de las diferencias entre modelos como vía para comprobar la corrección de la especificación de transformaciones entre modelos realizadas por humanos. De ese modo, si no hay diferencias entre el modelo esperado tras la transformación y el realmente obtenido, se considerará que las transformaciones están bien definidas.

En [Garcés et al., 2009] utilizan la diferencia entre diferentes versiones de un metamodelo como paso previo para adaptar los modelos que conforman con él.

Adicionalmente, otros trabajos han citado más escenarios en los que calcular la diferencia entre modelos es útil [Selonen, 2007]. Como ejemplos, se pueden citar:

1. La investigación de los patrones de cambio en la evolución del software, explotando las motivaciones subyacentes en ellos y guiando futuras actividades de desarrollo y mantenimiento [Xing and Stroulia, 2005].

2. Detección de similitudes entre diferentes variantes [Mandelin et al., 2006].

3. Soporte para reuso [Maiden and Sutcliffe, 1992].

4. Ahorro de ancho de banda y tiempo en las comunicaciones [Alanen and Porres, 2003].

La creciente importancia de esta materia se ve reflejada en la aparición de herramientas como *Sidiff* [Schmidt and Gloetzner, 2008], un framework para construir herramientas centradas en la diferencia entre modelos.

Otra aplicación del cálculo de la diferencia entre modelos es que se puede considerar como un aspecto clave para generar artefactos de manera incremental en función de los cambios en los modelos (véase capítulo 13).

8.11.2. Cálculo de las diferencias entre modelos

Existen muchos algoritmos que podrían ser utilizados para calcular las diferencias entre distintos modelos, y que pueden ser clasificados según diferentes aproximaciones [Kolovos et al., 2009].

Los primeros algoritmos fueron desarrollados con el objetivo de comparar únicamente archivos de texto (p.e., [Hunt and Mcllroy, 1977, Hirschberg, 1977, Myers, 1986]). Aunque dichas propuestas sirven para su cometido, utilizan una unidad de versión a nivel de archivo y una unidad de control a nivel de párrafo, abstracciones incorrectas para trabajar con modelos [Oliveira et al., 2005] porque utilizan diferentes estructuras de datos.

Desde el punto de vista de los modelos, la aproximación más sencilla es asumir que cada elemento tiene un UUID (Universally Unique Identifier o Identificador Universal Único) estático (p.e., [Alanen and Porres, 2003, Farail et al., 2006, Vernadat

et al., 2006]), que perdura durante todo la vida de los elementos. Así, se permite realizar cálculos rápidos sin necesidad de ningún tipo de configuración. Sin embargo, esta aproximación es muy optimista porque no puede ser utilizada cuando dos modelos se construyen de forma independiente o cuando las tecnologías no soportan el mantenimiento de identificadores únicos. Además, el uso de UUIDs hace aumentar el tamaño de los modelos. Sin embargo, en los casos en los que se puede aplicar, permite la comparación de modelos de manera rápida y sencilla.

El siguiente paso fue el uso de una técnica basada en la asignación de identificadores calculados dinámicamente según ciertos valores de los modelos. Para ello, se utilizan funciones definidas con tal propósito [Reddy and France, 2005]. Esta aproximación permite comparar modelos que han sido construidos de forma independiente pero tiene la desventaja de que no es fácil definir funciones para comparar cada elemento de los modelos para cada problema específico que pudiera existir.

Existen otras aproximaciones que utilizan árboles con hijos ordenados (p.e., [Tai, 1979, Cobena et al., 2001]) o desordenados (p.e., [Zhang and Shasha, 1989, Wang, 2005]). Los árboles son un tipo especial de grafo que no puede contener ciclos. Sin embargo, la naturaleza de los modelos hace que los árboles sean demasiado restrictivos para representarlos.

La última generación de algoritmos están basados en tratar a los modelos como grafos, los cual no es un problema trivial [Khuller and Raghavachari, 1996], y hacer comparaciones utilizando heurísticos mediante la búsqueda de similitudes entre los elementos de los modelos. Para ello, existen principalmente dos grandes grupos de algoritmos:

- Los dependientes del metamodelo.
- Los independientes del metamodelo.

Cálculo dependiente del metamodelo

Los algoritmos dependientes del metamodelo pueden ofrecer mejores resultados al ser mucho más específicos. Existen muchos trabajos sobre ellos, como [Nejati et al., 2007], en el que propusieron una serie de algoritmos para realizar comparaciones utilizando heurísticos a nivel terminológico, estructural y semántico. Este trabajo, se restringe únicamente a diagramas de estado, que describen aspectos comportacionales en el sector de las telecomunicaciones. En [Ohst et al., 2003] se realizó un trabajo que trata la obtención de las diferencias entre versiones de diagramas UML, haciendo mucho hincapié en la representación de dichas diferencias.

La principal desventaja de estos trabajos es que los algoritmos propuestos y las herramientas en los que se implementan son específicos del tipo de modelo empleado en cada caso. La misma desventaja se encuentra en otros trabajos como [Xing and Stroulia, 2005, Selonen, 2007]. El hecho de trabajar con UML o con cualquier otro metamodelo específico como el Ontology Definition Metamodel (ODM) [OMG, 2005d] hace que los algoritmos estén restringidos únicamente al metamodelo elegido. Además, los modelos procedentes de un DSL arbitrario, más generales que por ejemplo los modelos UML, suelen tener otra estructura, sintaxis y semántica [Lin et al., 2007], requiriendo un tratamiento diferente.

Cálculo independiente del metamodelo

Existen varios trabajos cuyo objetivo es mostrar cómo se pueden comparar modelos siguiendo un enfoque independiente del metamodelo utilizado. Por ejemplo, en [Lin et al., 2007] se presentó un algoritmo y una herramienta (DSMDiff) que calcula las diferencias y las similitudes entre dos modelos de un DSL arbitrario. Algunas características de la herramienta realizada son:

- Está basada en el Generic Modeling Environment (GME) [Ledeczi et al., 2001].
- Utiliza *scripts* de editado para representar las diferencias entre dos modelos.
- Examina la similitud estructural en una región específica y localizada, en la que el nodo a comparar es el centro y sus vecinos inmediatos son el borde, restringiendo las capacidades del algoritmo.
- Emplea un heurístico en el que la existencia de un atributo *name* es muy importante en cada elemento del modelo.

Otros interesantes trabajos en este campo son:

1. [Melnik et al., 2002], en el que se presentó un algoritmo utilizable en diferentes escenarios.
2. [Mandelin et al., 2006], en el que se presentó un framework basado en métodos Bayesianos para encontrar las correspondencias de los modelos de forma automática.
3. [Selonen and Kettunen, 2007], en el que se presentó una aproximación muy flexible para inferir correspondencias entre elementos de los modelo.

4. [Treude et al., 2007], en el que se presentó una técnica para computar las diferencias entre modelos un orden de magnitud más rápido que otros algoritmos gracias al empleo de un árbol de búsqueda multi-dimensional.

5. [Rivera and Vallecillo, 2008], en el que se presentó un metamodelo para poder representar diferencias entre modelos mediante el uso de otro modelo.

Una interesante opción es el framework denominado SiDiff [Schmidt and Gloetzner, 2008], que sirve como base para crear herramientas en las que se calcule la diferencia entre modelos. Para cada caso, hay que configurar como se computa la diferencia entre los diferentes elementos de un modelo utilizando pesos para cada propiedad estructural o para la semejanza entre los valores de los diferentes atributos, y un algoritmo que haga el cálculo global. Hay que tener en cuenta que la configuración para cada algoritmo utilizado para comparar modelos depende siempre de la semántica de cada metamodelo, de cómo los modelos se cambian e incluso de las apreciaciones de los usuarios de cuando un cambio es importante o no.

Cabe destacar que entre las implementaciones de herramientas MDE, la que ha tenido una mayor acogida por parte del ámbito empresarial e investigador es el EMP (Eclipse Modeling Project) [Gronback, 2009], proyecto ubicado dentro de la plataforma Eclipse, en el que se implementan gran parte de las especificaciones de OMG, en particular las que tienen que ver con el estándar MDA [Gronback, 2006]. En el EMP, el meta-metamodelo utilizado es el llamado Ecore, tan próximo al meta-metamodelo estándar denominado MOF que es tomado como una implementación del núcleo de MOF, es decir, de EMOF. Otros autores como [Gruschko et al., 2007] justifican el empleo de Ecore diciendo que es el estándar de facto en la industria. En el núcleo del EMP está el EMF [Budinsky et al., 2003, Steinberg et al., 2009]. Según sus autores, la herramienta EMF Compare [Toulmé, 2007]:

> ❝*Ofrece un mecanismo para comparar modelos dentro del EMF y proporciona soporte genérico para cualquier tipo de metamodelo con la intención de comparar y unir modelos. Los objetivos son proporcionar una implementación estable y eficiente para comparar modelos y ofrecer un framework extensible para necesidades específicas.*❞

Además, la salida obtenida está basada en modelos, los cuales proporcionan la posibilidad de hacer manipulaciones posteriores de un modo sencillo. EMF Compare, y por extensión el algoritmo genérico que incorpora, es la herramienta más conocida, con más éxito y utilizada para llevar a cabo comparaciones de modelos.

8.12. Conclusiones

Pese a que los sistemas de control de versiones son herramientas muy útiles en el desarrollo de software, todavía presentan problemas, sobre todo si se quieren utilizar en desarrollos dirigidos por modelos. Esa es la razón por la que existen varias investigaciones que se centran en el trabajo con MVCs.

Sin embargo, y debido a que actualmente todavía existe una carencia de MVCSs finalizados y disponibles, la primera parte del siguiente capítulo se destinará a mostrar aspectos de un prototipo desarrollado que puede simular el comportamiento que debería tener cualquier sistema de control de versiones para modelos real. Si no fuera por el prototipo, no sería posible que una herramienta de integración continua trabajara con un MVCS[6].

Además, se presentará MCTest (Model Comparison Testing Engine), una herramienta que sirve para probar, comparar o medir diferentes implementaciones de algoritmos de comparación de modelos, utilidad que puede ser de gran ayuda para facilitar la creación de algoritmos más eficientes que permitan detectar nuevas versiones de un modelo con una mayor fiabilidad.

[6]Todas las herramientas de integración continua trabajan en base a la información disponible en uno o varios repositorios, típicamente gestionados por un sistema de control de versiones

CAPÍTULO 9

Solución adoptada

Pese a que existen varias propuestas para afrontar la necesidad de sistemas de control de versiones para modelos (p.e., [Oliveira et al., 2005, Murta et al., 2008, Altmanninger, 2007]), todavía no existe ninguna herramienta completa y disponible que permita trabajar con dichas propuestas. Por lo tanto, los sistemas de integración continua disponibles, pese a sus mecanismos de extensión, no pueden trabajar con un sistema de control de versión que gestione correctamente las diferentes versiones de los modelos.

En este capítulo se mostrará, en primer lugar, la solución adoptada y la arquitectura utilizada para ofrecer una alternativa frente a la carencia actual de herramientas MVCSs. Estas herramientas permiten mejorar la forma en la que la integración continua se realiza cuando se trabaja bajo el paradigma MDE.

Debido a que el mayor impedimento para el desarrollo de sistemas reales MVCSs es que los algoritmos que emplean no funcionan de forma óptima, en segundo lugar, se presentará MCTest (Model Comparison Testing Engine) [García-Díaz et al., 2011b], herramienta que permite probar, comparar o medir algoritmos de comparación de modelos, aspecto que puede ser fundamental para el desarrollo de las herramientas MVCS en un futuro cercano.

* * * *

9.1. Introducción

Para alcanzar el propósito de este trabajo es necesario disponer de un MVCS, por tanto, se ha desarrollado un prototipo que simula el comportamiento, en cuanto a la unidad de versión (UV), de las propuestas realizadas por los autores más relevantes en este ámbito. La idea es, basándose en los problemas citados en [Oliveira et al., 2005], permitir decidir en cada caso la UV utilizada para lograr la máxima flexibilidad en la realización de pruebas. Los modelos se guardarán en un repositorio que será *observado* por la herramienta de integración continua. Así, en el momento en el que se detecte una nueva versión del modelo, se lanzará el proceso de construcción.

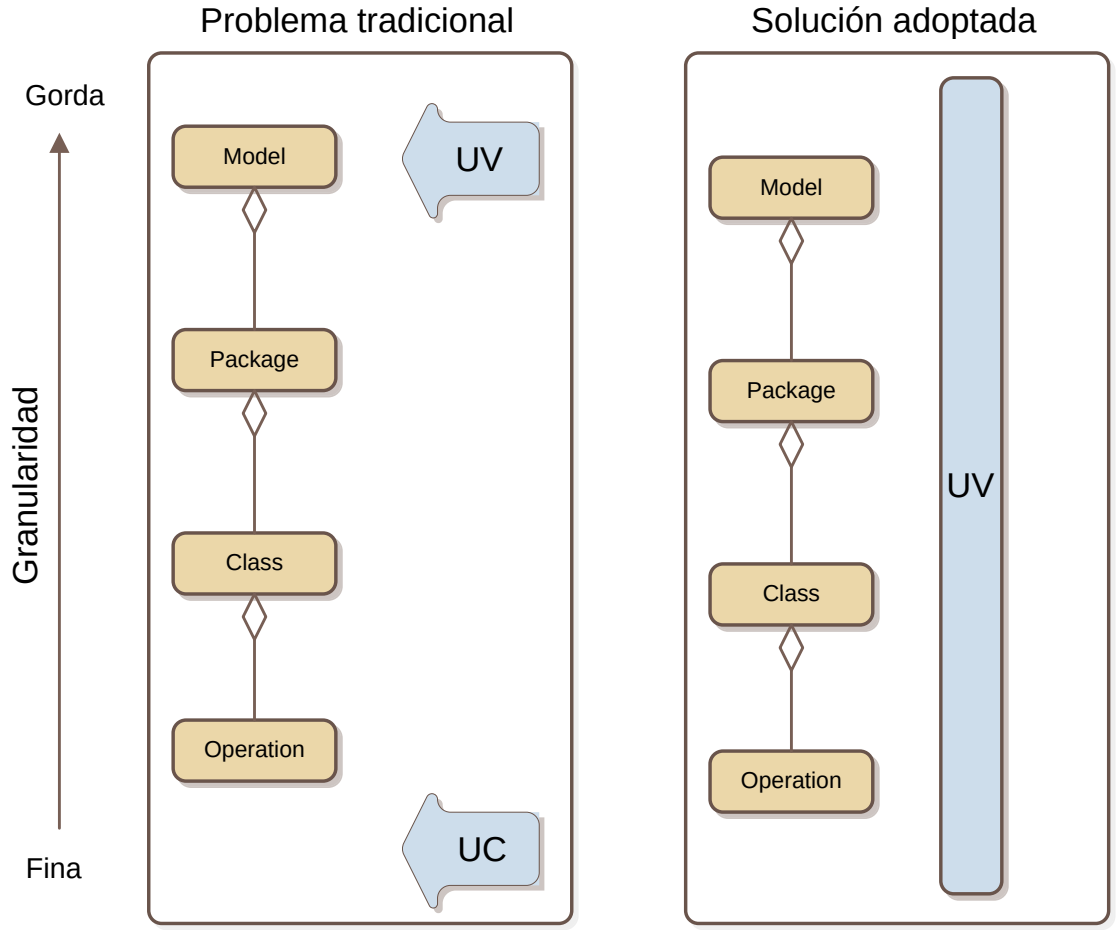


Figura 9.1: UV y UC en modelos

La Fig. 9.1, que se basa en la Fig. 8.1, muestra que el prototipo desarrollado permite configurar la UV libremente para cada caso. Así, se consigue la máxima potencia en cuanto las posibilidades de interacción con la herramienta de CI. Por otra parte, no se ha considerado la UC (tratada ampliamente en trabajos como [Reiter

et al., 2007]) porque está orientada al trabajo colaborativo cuando se utiliza una gestión de fuentes optimista [OReilly et al., 2003], característica fuera del alcance de la investigación, y que por tanto no ha sido incluida en el prototipo.

9.2. Arquitectura utilizada

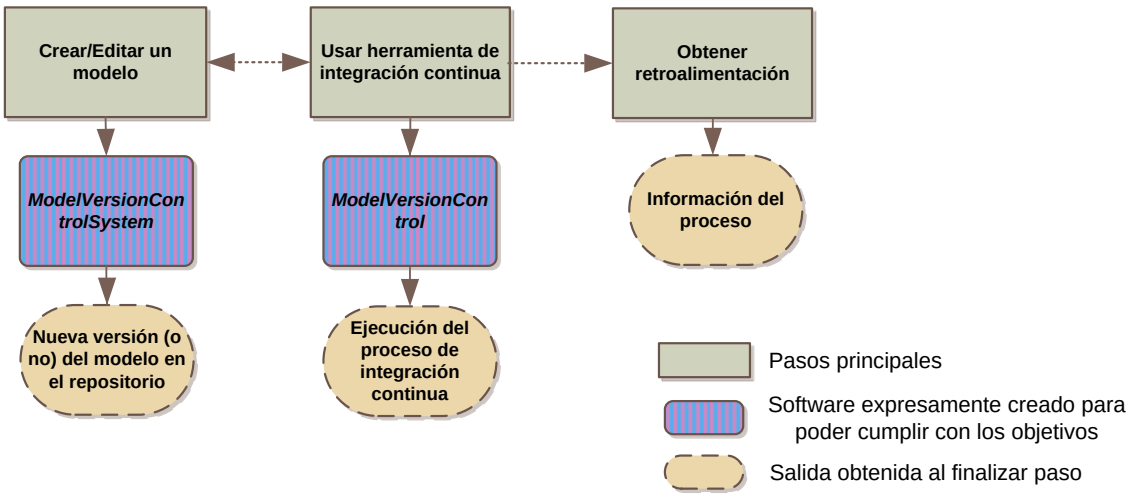


Figura 9.2: Pasos para relacionar CI con MVCS

En la Fig. 9.2 pueden verse los tres pasos fundamentales para la interacción de la práctica CI (Continuous Integration) con los MVCSs (Model-Version Control Systems).

9.2.1. Crear/Editar un modelo

Debido a que TMDE (véase capítulo 7) promueve el uso de metamodelos derivados de MOF y la implementación de MOF más fiel es Ecore, que abarca al subconjunto EMOF, se utilizan modelos Ecore para trabajar con el repositorio.

Estos modelos pueden ser manipulados de varias formas, entre las que destacan:

- Manualmente.
- Con la herramienta en forma de árbol que proporciona el Eclipse Modeling Framework.
- Con un editor textual.
- Con un editor gráfico.

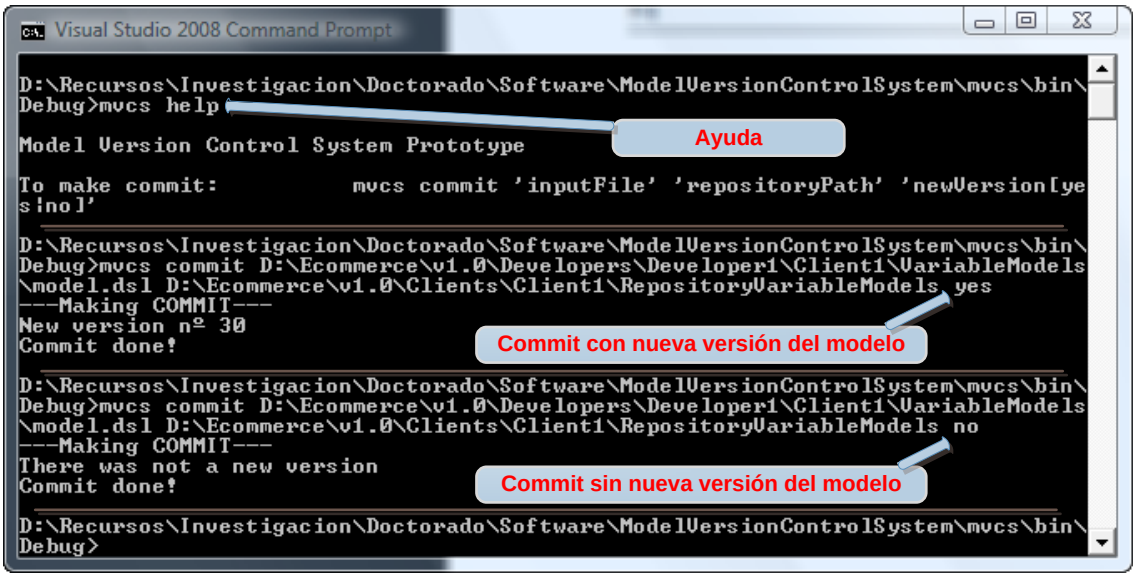


Figura 9.3: Consola para trabajar con *ModelVersionControlSystem*

Sea cual sea la forma de trabajar con el modelo y el entorno de desarrollo, los cambios de dicho modelo se insertarán en un repositorio, debido a que son artefactos de primera mano en el desarrollo del software [Tichy, 1985].

El prototipo desarrollado, llamado *ModelVersionControlSystem*, permite definir cuándo los cambios en un modelo del repositorio conducen a una nueva versión de dicho modelo dentro del repositorio. La Fig. 9.3 muestra la consola de trabajo y la facilidad para especificar si un cambio ha producido una nueva versión del modelo. Por otra parte, la Fig. 9.4 muestra el aspecto del repositorio después de 30 versiones de un mismo modelo.

9.2.2. Usar herramienta de integración continua

Uno de los aspectos desarrollados es el soporte para *ModelVersionControlSystem* por parte de las herramientas de CI. Para realizar este trabajo se ha elegido la herramienta Cruise Control (CC), por ser posiblemente la más utilizada, por disponer de su código fuente y por ofrecer mecanismos de extensión (véase sección 4.7). Esta herramienta verificará periódicamente si hay nuevas versiones de un modelo en el repositorio ya que, de haberlas, lanzará el proceso de integración continua. La Fig. 9.5 muestra la forma en la que se ha extendido la herramienta de CI para dar soporte al trabajo con *ModelVersionControlSystem*.

Por un lado, se ha modificado el archivo de configuración del servidor *ccnet.config* para indicar que el MVCS utilizado para trabajar con el repositorio en el que se

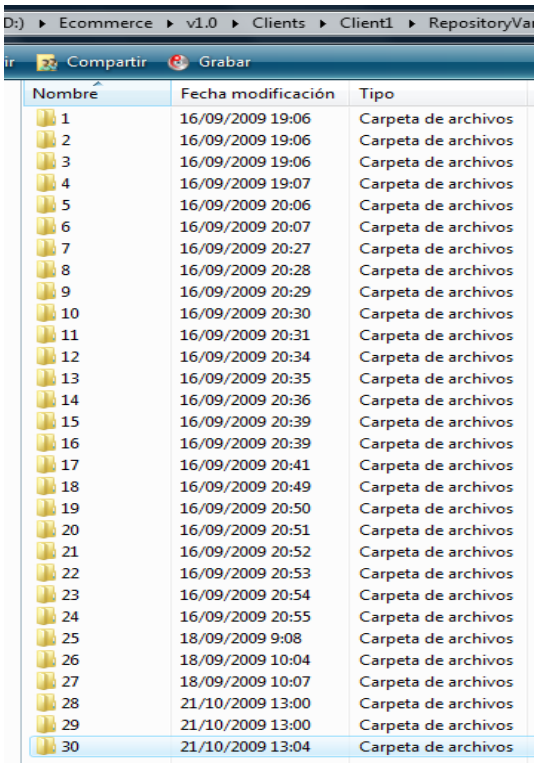


Figura 9.4: Un repositorio utilizado con *ModelVersionControlSystem*

encuentran los modelos es *ModelVersionControlSystem*.

Por otro lado, se ha creado una extensión para CC denominada *ModelVersionControl*, que hace de conexión entre CC y el MVCS, verificando periódicamente (cuando el disparador de la herramienta de CI se ejecuta) si hay nuevas versiones del modelo en el repositorio.

Los disparadores en la integración continua

Un elemento destacado de la integración continua, aunque olvidado por la literatura científica, es el disparador, que inicia el proceso de construcción del software a partir de las fuentes de desarrollo, generalmente *preguntando* al sistema de control de versiones si se ha producido algún cambio en las fuentes de desarrollo.

En el trabajo [García-Díaz et al., 2010b] se expone la posibilidad de mejora de dicho componente software y se abre el camino hacia una investigación que podría ser aplicada a otros ámbitos relacionados con la informática. Para ello, se ha implementado un prototipo que muestra, para un caso de estudio, los resultados obtenidos cuando se utilizan los disparadores de integración continua actuales.

Hay que destacar que aún existen pocos trabajos científicos relacionados con CI

Figura 9.5: Cruise Control y *ModelVersionControlSystem*

como por ejemplo [Holck and Jorgensen, 2004]. Por ello, los trabajos centrados en los disparadores de las herramientas de CI son prácticamente inexistentes. Sin embargo, aún así, hay varios tipos de disparadores que no necesitan intervención manual para ser ejecutados. Algunos de los más conocidos son:

- *Dependency trigger*. Este disparador se emplea para que una construcción comience después de otras de las que depende.

- *Interval trigger*. Con este disparador la etapa de construcción comienza cada un intervalo de tiempo especificado tras el último ciclo de integración. Generalmente, se configura para realizar la construcción únicamente si ha habido un cambio en los repositorios de las fuentes del software.

- *Multiple trigger*. Este disparador se utiliza para permitir utilizar múltiples disparadores simultáneamente.

- *Schedule trigger*. Este disparador permite especificar los días de la semana y la hora en la que comenzará la etapa de construcción.

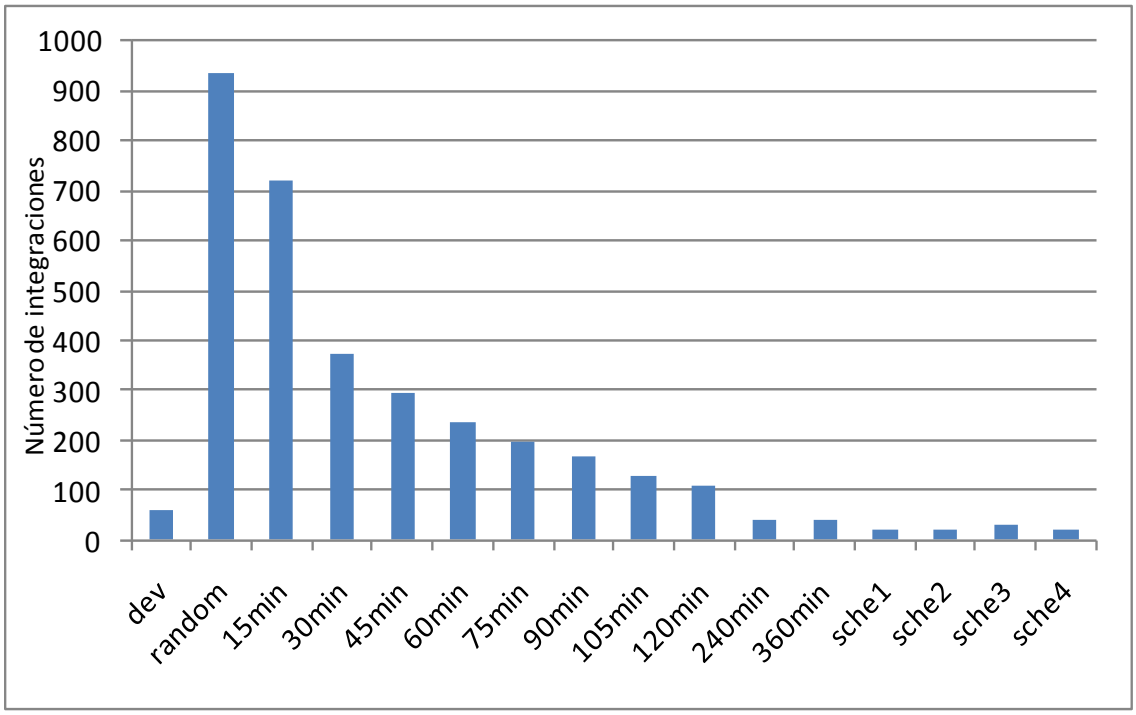


Figura 9.6: Número de integraciones iniciadas con cada configuración

- *Startup trigger*. Este disparador se ejecuta siempre que se inicia la herramienta de integración continua.
- *URL trigger*. Este disparador lanza la construcción cuando una determinada URL (Uniform Resource Locator) cambia. Para ello, se realiza una comprobación cada un cierto intervalo de tiempo. Así, se evita aumentar la carga que sobre los sistemas de control de versiones pueden ocasionar los disparadores por intervalos.

Para mostrar el problema al que se pretender dar solución, se ha realizado un estudio durante un mes en un proyecto ficticio realizado por cuatro personas que trabajan de lunes a viernes de 8:30 a 15:30 (los martes y miércoles también se trabaja por las tardes de 18:00 a 20:00). Hay que tener en cuenta que durante ese mes hay cuatro fines de semana y un miércoles festivo. Para obtener los resultados que se mostrarán a continuación se ha construido un prototipo que funciona a modo de simulador.

En el estudio se revela que el equipo de desarrollo realiza 60 modificaciones en el repositorio gobernado por el sistema de control de versiones. Así, un hipotético disparador óptimo y automático tendría que realizar únicamente 60 intentos de construcción de las fuentes del software y además hacerlo instantáneamente después

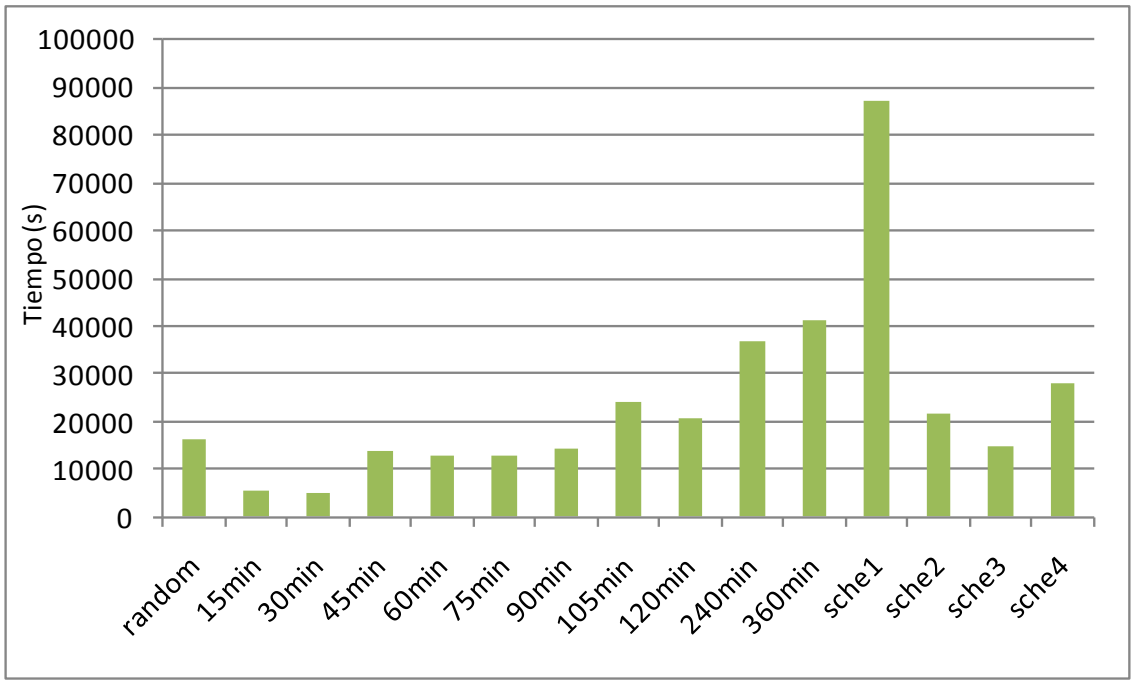


Figura 9.7: Tiempo (s) sin realizar integraciones con cada configuración

de los cambios realizados por los desarrolladores. De ese modo, se ahorrarían ciclos de CPU y el software estaría integrado de la forma más temprana posible.

Utilizando las tecnologías que actualmente disponen las herramientas de integración continua, se han utilizado disparadores por intervalos junto con un filtro que permite que se disparen únicamente las horas supuestamente laborables (cada 5, 15, 30, 45, 60, 75, 90, 105, 120, 240, 360 minutos). Lo mismo se ha hecho con disparadores planificados en horas significativas del trabajo (a las 8:30, 15:30, 15:30 y 20:00, 20:00 horas)[1].

En la Fig. 9.6 se pueden observar las modificaciones realizadas por el equipo de desarrollo a lo largo de los 30 días. Como indica la figura, las configuraciones que mejor se adaptan, es decir, las que más se ajustan al número de modificaciones reales realizadas (60) son las planificadas y las que se hacen cada, como mínimo, dos horas.

Sin embargo, los datos mostrados en la Fig. 9.6 se muestran opuestos a los de la Fig. 9.7. En la Fig. 9.7 se puede observar que con los disparadores cada 30 y 15 minutos son con los que el software está menos tiempo sin integrar, ayudando así a prevenir errores tan pronto como sea posible.

Los datos exponen que si se desea que el software se integre rápidamente cuando

[1]Hay que tener en cuenta que varios días se ha trabajado tiempo extra, realizándose modificaciones en el repositorio tras la hora en la que se ejecuta el disparador

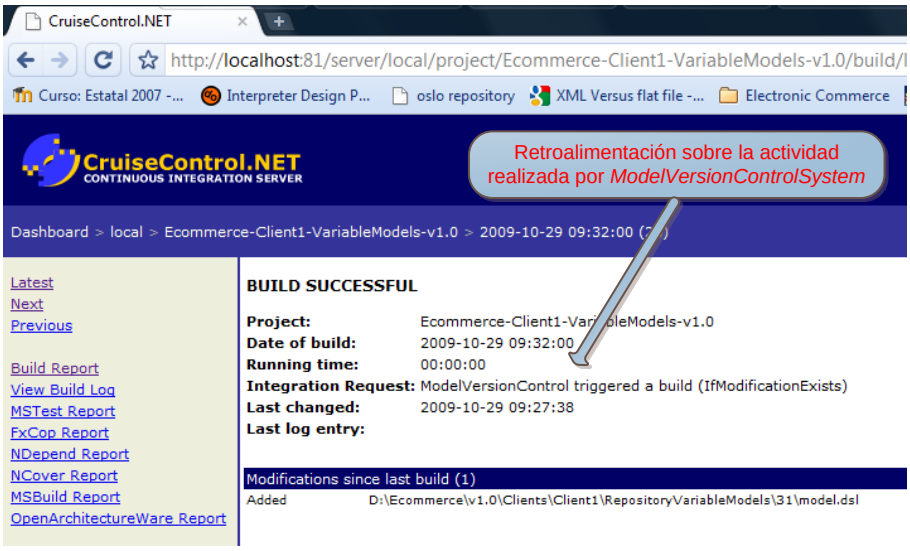


Figura 9.8: Retroalimentación obtenida por la herramienta de monitorización

los desarrolladores realizan cambios en las fuentes, entonces es necesario sobrecargar más el sistema. Lograr un equilibrio entre los valores obtenidos en ambas gráficas podría significar una mejora de la calidad en el desarrollo, ya que cuanto primero se integre el software, tanto mejor [Fowler, 2009a] (véase sección 19.5).

9.2.3. Obtener retroalimentación

Además del servidor, CC tiene una herramienta de monitorización que sirve para obtener retroalimentación sobre las acciones que se van produciendo. En la Fig. 9.8 puede observarse la información obtenida cuando se hace un cambio en un repositorio gobernado por *ModelVersionControlSystem*.

9.3. MCTest

Debido a la creciente importancia de MDE y a los cambios experimentados por los sistemas software durante todo su ciclo de vida, el cálculo, la representación y la visualización de las semejanzas y diferencias entre dos versiones de un mismo modelo se está convirtiendo en algo cada día más necesario y útil.

En esta sección se muestra la necesidad de una mejora en los algoritmos de cálculo de las similitudes entre modelos y se ofrece un framework para probar diferentes implementaciones [García-Díaz et al., 2011b], reduciendo el esfuerzo necesario para medir, comparar o crear nuevos algoritmos. Para mostrar las necesidades de mejora y el framework desarrollado, se han creado diferentes modelos que conforman con el

metamodelo de un lenguaje de dominio específico. Posteriormente, se han comparado dichos modelos con la herramienta EMF Compare, perteneciente al Eclipse Modeling Project, framework de referencia en cuanto a la ingeniería dirigida por modelos.

El MM (Model Management o Gestionado de Modelos) busca proporcionar un modo para representar relaciones entre modelos utilizando un conjunto de operadores. Por ejemplo, *merge* se utiliza para crear uniones entre modelos de acuerdo a sus relaciones [Nejati et al., 2007]. Otro ejemplo son *match* y *diff*, que son muy interesantes porque se utilizan para calcular las relaciones entre modelos, siendo la base para cualquier otra operación entre ellos. Son extremadamente útiles en muchos aspectos (véase sección 8.11.1).

El problema de determinar las relaciones entre modelos es implícitamente complejo e involucra tres pasos fundamentales [Brun and Pierantonio, 2008]:

1. El algoritmo de cálculo.

2. La representación en un formato que permita una fácil manipulación informática.

3. La visualización en un formato amigable para las personas.

MCTest se centra en el primer y más importante paso, es decir, en el algoritmo de cálculo de las relaciones entre modelos. Dicho cálculo es una tarea muy compleja que no tiene una solución óptima para todos los casos, ya que depende de cada problema específico [Kolovos et al., 2009]. En particular, en esta primera versión de MCTest, se ha abordado el estudio del operador *match*, generalmente realizado como paso previo a *diff*. MCTest permite evaluar diferentes algoritmos de comparación de modelos de forma rápida y efectiva, utilizando una interfaz gráfica para mostrar los resultados.

Las siguientes subsecciones están dedicadas a profundizar sobre la herramienta MCTest. Primero, se describirá el caso de estudio. Posteriormente, se presentará la arquitectura principal de la herramienta y por último, se mostrará una evaluación de la herramienta a través de su empleo con el caso de estudio.

9.3.1. Caso de estudio

La Fig. 9.9 muestra la gramática de un DSL desarrollado utilizando Xtext [Efftinge and Völter, 2006]. Con Xtext, a partir de cada gramática específica se generan diferentes artefactos. Por ejemplo, un IDE (Integrated Development Environment o Entorno de Desarrollo Integrado) totalmente integrado en la plataforma Eclipse, un *parser* para procesar las instancias del lenguaje o el metamodelo Ecore que se

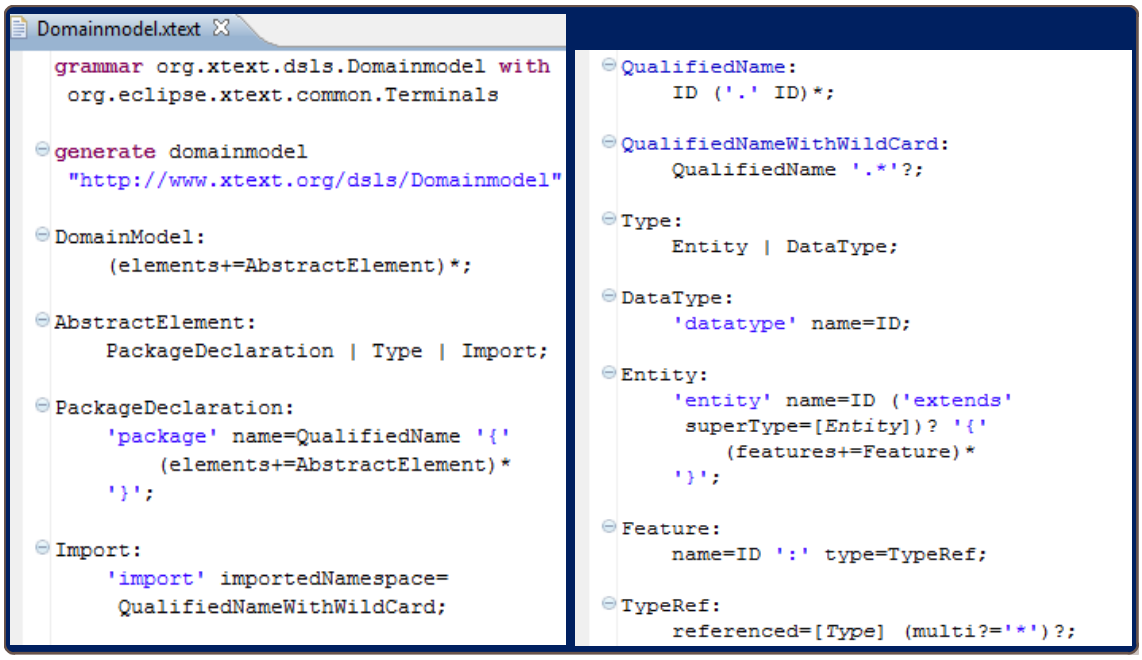


Figura 9.9: Gramática de un lenguaje de dominio específico

corresponda con la gramática. De esa forma, siguiendo los principios de MDE, los desarrolladores utilizan el IDE creado para modelar, creando, de forma transparente para ellos, instancias del metamodelo inferido.

Lo habitual es que todos los modelos sufran una evolución a lo largo del tiempo, que conlleva la aparición de nuevas versiones de los modelos. En este punto es donde es interesante calcular las semejanzas entre diferentes versiones de cada modelo, con el objetivo de realizar posteriores acciones de una forma óptima (véase sección 8.11.1).

En [Brun and Pierantonio, 2008] se describe que EMF Compare surge en 2006 por la necesidad de un motor de comparación de modelos dentro de la plataforma Eclipse. Las capacidades y posibilidades de esta herramienta son:

1. Está integrada dentro del EMP.

2. Ofrece soporte para cualquier metamodelo Ecore.

3. Incorpora heurísticos predefinidos para comparar modelos.

4. Ofrece mecanismos de extensión.

Estas características hacen que haya sido elegida para llevar a cabo las tareas necesarias en el caso de estudio y para probar la herramienta MCTest.

Figura 9.10: Pruebas realizadas

La evaluación llevada a cabo se basa en realizar cambios en un modelo para determinar si EMF Compare funciona de manera correcta. En una gran cantidad de pruebas, dicha herramienta funcionó sin problema utilizando el algoritmo genérico que incorpora por defecto. Sin embargo, la Fig. 9.10 muestra ejemplos para los cuales no se obtuvo un resultado óptimo:

1. En el primer caso, en lugar de detectar el nuevo tipo de datos *SInteger*, se detecta que ha habido un cambio del tipo de datos *Integer* al tipo de datos *SInteger*, y después que se ha insertado un nuevo elemento cuyo tipo de datos es *Integer*.

2. En el segundo caso, en lugar de detectar que se ha cambiado la multiplicidad del tipo de datos *Speaker*, se detecta que ha añadido un elemento nuevo.

3. En el tercer caso, en lugar de detectar un cambio de nombre de *Speaker* a *Presenter*, se detecta que un elemento es eliminado y que posteriormente se añade otro.

4. En el cuarto caso, en lugar de detectar un cambio de nombre de *Session* a *SessionInfo*, también se detecta que un elemento es eliminado y que posteriormente se añade otro. Sin embargo, la herramienta funcionó correctamente para la misma prueba realizada con otros identificadores (*Conference* y *ConferenceInfo*).

5. En el quinto caso, en lugar de detectar que se han movido los tres tipos de datos a un nuevo paquete, se detecta que dichos tipos de datos son elementos nuevos.

6. En el sexto caso, en lugar de detectar el nuevo paquete, se detecta que todos los elementos del modelo son nuevos.

7. En el último caso, en lugar de detectar el cambio de nombre de un elemento y la incorporación de un nuevo paquete, se detecta que todos los elementos del modelo son nuevos.

Sin embargo, no parece excesivamente complicado que se hubieran encontrado las soluciones óptimas para los casos previos:

1. En el primer caso, parece lógico pensar que si sólo hay un cambio en el modelo y si dicho cambio está causado por un elemento que además tiene un nombre diferente al del resto de elementos, dicho elemento debería ser un nuevo elemento que se ha añadido al modelo.

2. En el segundo caso, si sólo se ha cambiado un atributo *multi* de una instancia determinada de un tipo *TypeRef*, parece que lo más realista es pensar que ha habido un cambio y no que hay un nuevo elemento.

3. En el tercer caso, únicamente cambiando un nombre de *Speaker* a *Presenter* y manteniendo toda la estructura interna, tiene sentido pensar que solamente se ha producido un cambio. Además, con herramientas como Rita.WordNet[2], una librería Java que permite acceder a la ontología para el idioma inglés más conocida (WordNet [Fellbaum, 1998]), se puede hacer uso de una aproximación lingüística diferente a la clásica tipográfica. Así, en lugar de buscar por coincidencias exactas entre los nombres de varios elementos, se podrían considerar las correlaciones que podrían existir entre palabras. Rita.WordNet es un sistema distribuido gratuitamente que permite, entre otras cosas, medir la similitud semántica o la relación que existe entre dos conceptos, en este caso los nombres de las instancias.

4. En el cuarto caso, ocurre un fenómeno muy similar al del tercer caso. La única diferencia es que en este caso, el algoritmo de comparación de modelos ha decidido correctamente que *ConferenceInfo* no es un nuevo elemento pero ha fallado en determinar que *SessionInfo* sí lo es. El caso exitoso se debe al mayor número de letras que tiene *ConferenceInfo* exactamente iguales a nombre original respecto a *SessionInfo*, es decir, el tamaño de la variable es mayor, lo que provoca que el algoritmo tenga más opciones para hacer cálculos correctos.

5. En el quinto caso, no habiéndose realizado ninguna modificación en los elementos que se mueven, parece lógico pensar que los elementos se han movido al nuevo paquete y no que se han eliminado para posteriormente crear clones de ellos.

6. En el sexto caso, ocurre un caso muy parecido al anterior. Sin embargo, en esta ocasión parece incluso más evidente porque la cantidad de elementos que se mueven es mucho mayor.

7. En el último caso, sucede una mezcla de los casos anteriores. En esta ocasión se mueven elementos a un paquete y se renombra el otro.

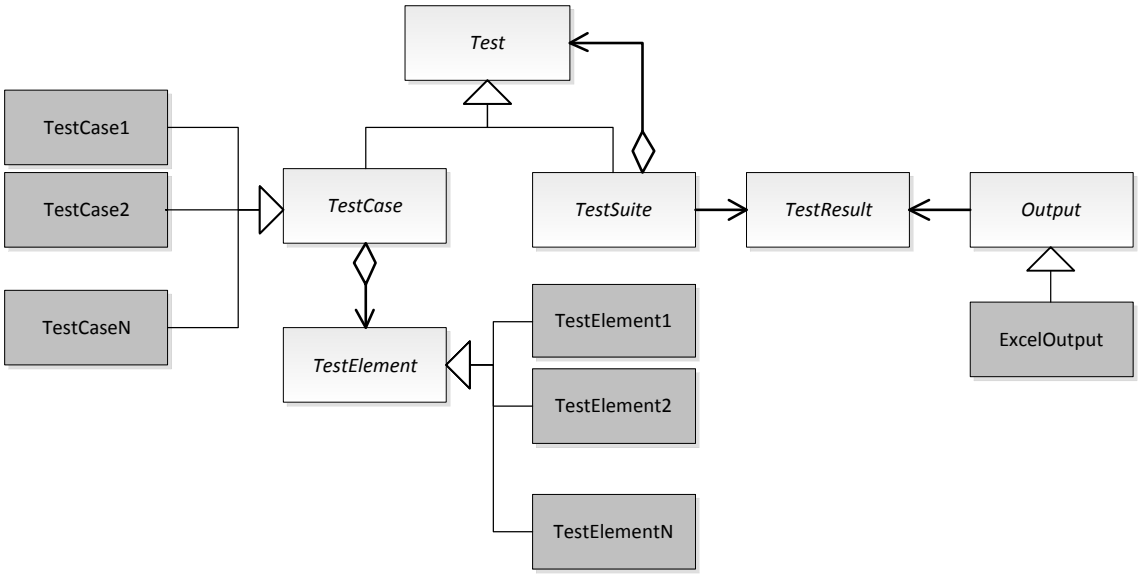


Figura 9.11: Arquitectura básica de MCTest

9.3.2. Arquitectura propuesta

La Fig. 9.11 muestra, de forma resumida, la arquitectura general de MCTest. Una prueba global consiste en uno o en varios conjuntos de prueba, los cuales funcionan como clasificadores para las pruebas concretas, de forma similar a como trabajan las herramientas de la familia xUnit [Beck, 1994]. Sin embargo, las herramientas xUnit están destinadas a probar software realizado mediante el empleo de lenguajes de programación tradicionales.

La arquitectura de MCTest está inspirada en el diseño de JUnit[3], posiblemente la herramienta más conocida, utilizada y extendida para probar software desarrollado con el lenguaje de programación Java.

La diferencia es que JUnit es una herramienta diseñada para realizar principalmente pruebas unitarias en software desarrollado, con el objetivo de comparar el resultado obtenido con el resultado esperado a través del empleo de *asserts* insertados de acuerdo al criterio de la persona encargada de hacer las pruebas. Por contra, MCTest, está diseñada para realizar pruebas en algoritmos de comparación de modelos, con el objetivo de comparar el resultado obtenido con el resultado esperado de acuerdo al criterio de la persona encargada de dicha tarea. Por ejemplo, el experto en el dominio particular de conocimiento para el que se haya construido modelos, sería la persona más indicada para decidir cuando dos modelos son equivalentes,

[2]http://www.rednoise.org/rita/wordnet/
[3]http://www.junit.org/

aunque manejen datos diferentes.

Del mismo modo que los modelos elevan el nivel de abstracción si se comparan con los lenguajes de programación tradicionales, MCTest hace pruebas en un nivel de abstracción superior si se compara con las herramientas existentes anteriormente. El objetivo ya no es probar código fuente, sino probar algoritmos de comparación de modelos. O lo que es lo mismo, comparar los modelos obtenidos tras la aplicación de los algoritmos. La idea de MCTest es cubrir un vacío en cuanto a las herramientas de comparación para permitir mejorar los resultados obtenidos por los diferentes algoritmos.

Un caso de prueba funciona como contenedor en el que se introducen elementos. En este caso, en lugar de utilizar *asserts*, los usuarios de MCTest han de extender la clase abstracta denominada *TestCase* para indicar cuál es la comparación óptima entre dos modelos (el primero sería el origen y el segundo sería el destino). A través de dicha comparación óptima, se podrían estudiar tantos algoritmos de comparación de modelos como se deseen mediante el empleo de elementos de prueba. Para crear un elemento de prueba, habrá que extender la clase abstracta denominada *TestElement* e implementar o utilizar en dicha clase el algoritmo deseado. El motor de ejecución de MCTest realizará entonces las tareas necesarias, comparando el resultado obtenido con el esperado en cada caso.

En esta primera versión de MCTest, cada elemento de prueba es equivalente a un algoritmo de cálculo de semejanzas entre modelos. Es decir, cada caso de prueba tiene asignado uno o más algoritmos cuyos resultados son comparados con el resultado óptimo. Un elemento de prueba encapsula al algoritmo subyacente, haciendo que todos los elementos puedan ser intercambiados y utilizados libremente, con cualquier otro caso de prueba y sin ninguna dependencia. Además, cada caso de prueba utiliza la misma interfaz para los dos, tanto para el resultado óptimo como para el resultado obtenido, empleando la misma estructura de datos, independientemente del verdadero origen de cada elemento de prueba (p.e., EMF Compare, algoritmos de otras herramientas, algoritmos en desarrollo, trabajos de investigación, etc.).

Con este diseño desacoplado, se pueden crear grupos de prueba de forma independiente a los algoritmos que existen o que existirán en el futuro. Así, se pueden crear colecciones de prueba (casos de prueba) que pueden ser extendidas con la incorporación de nuevos algoritmos (elementos de prueba). Este proceso es análogo a las pruebas regresivas realizadas por las herramientas xUnit, de gran valor en la ingeniería del software [Kung et al., 1996]. Además, cada elemento de prueba puede ser ejecutado en cada caso de prueba, haciendo que el número de combinaciones a probar aumente exponencialmente sin ningún esfuerzo adicional.

Creación de programas de prueba

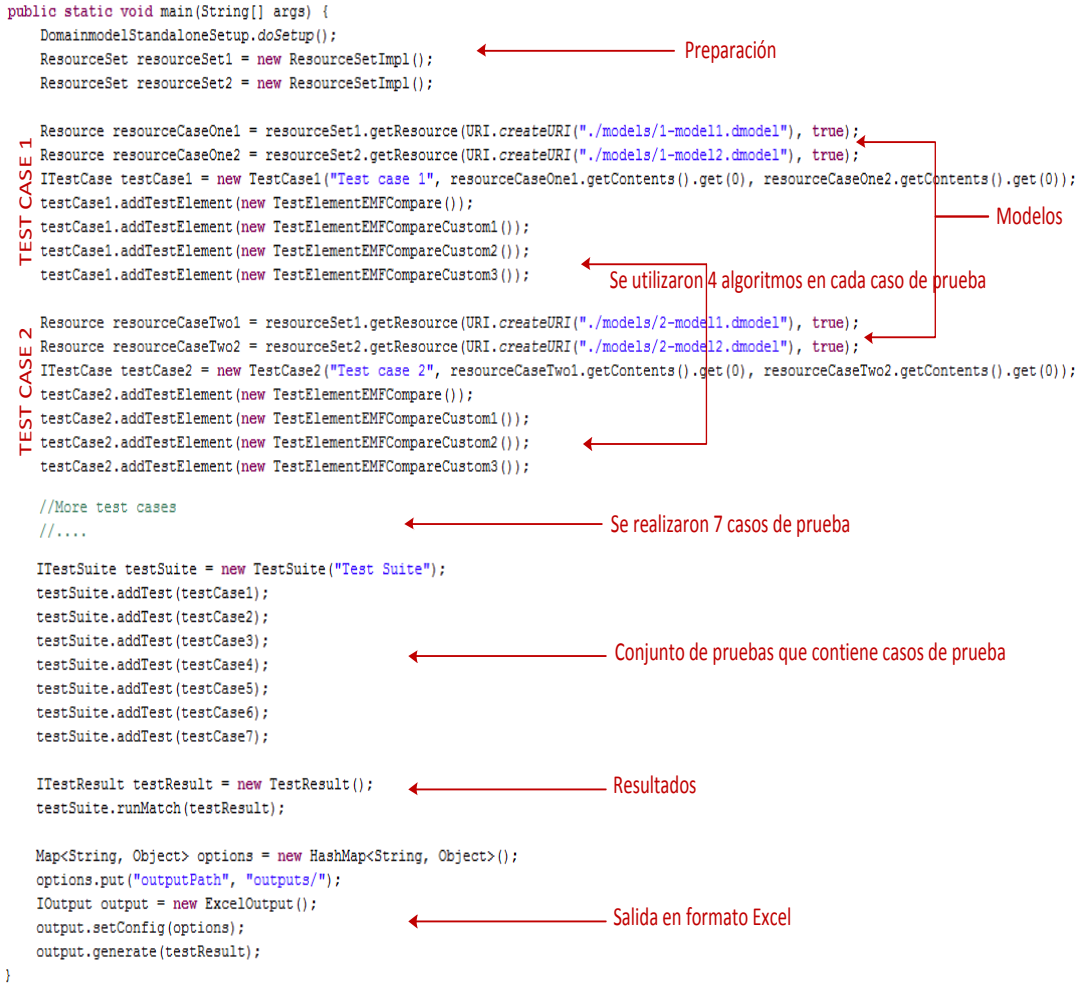


Figura 9.12: Programa desarrollado utilizando MCTest

La Fig. 9.12 muestra el código fuente necesario para crear una prueba completa, que como se puede observar es muy compacto. La primera línea se utiliza para inicializar los componentes necesarios para trabajar con el DSL mostrado en el caso de estudio (Fig. 9.10). Las siguientes dos líneas se utilizan para crear conjuntos de recursos utilizados por EMF para trabajar con los modelos software.

En la Fig. 9.12, sólo se muestran dos de los siete casos de prueba, pero el proceso para crear casos de prueba con MCTest es siempre idéntico, lo que hace que sea una herramienta sencilla de utilizar. Por ejemplo, en el primer caso de prueba se cargan en memoria dos modelos, los cuales se corresponden con los dos modelos del primer

caso de la Fig. 9.10. Posteriormente, se crea una instancia para el caso de prueba al que se le pasan como parámetros los dos modelos cargados y un nombre significativo. El nombre de la clase utilizada para el primer caso de prueba es *TestCase1*, siendo la clase en la que se define la solución óptima que se espera tras la comparación de los modelos. Para conseguirlo se implementa el método abstracto *public IMatchModel getOptimalMatchModel()*. El siguiente paso consiste en añadir elementos de prueba con el objetivo de evaluarlos a través del uso del motor de ejecución de MCTest. En la Fig. 9.12 se observa que en este ejemplo se comparan los mismos algoritmos en los dos casos de prueba mostrados, pero no tendría por qué ser así. Además, en el segundo caso de prueba los modelos cargados en memoria son otros. Dichos modelos se corresponden con el segundo caso de la Fig. 9.10.

Con los casos de prueba definidos, lo siguiente consiste en crear conjuntos de prueba para distribuirlos o clasificarlos. Para mantener el ejemplo sencillo, en la Fig. 9.12 sólo se muestra un conjunto en el que se incluyen todos los casos de prueba. Además, la interfaz denominada *ITestResult* contiene la estructura de las clases en las que se almacena toda la retroalimentación obtenida tras el proceso. Dichas clases, por defecto, no proporcionan una interfaz amigable para el usuario de la herramienta. Por esta razón, se utiliza la interfaz denominada *IOutput* para decorar la retroalimentación obtenida con alguna tecnología diferente a la programática. Por defecto, MCTest ofrece una salida en formato Excel[4], que muestra toda la información relevante. MCTest podría ofrecer la retroalimentación utilizando otras tecnologías de manera muy sencilla.

Retroalimentación obtenida

Al igual que se hace en herramientas ampliamente difundidas como SiDiff [Schmidt and Gloetzner, 2008], la retroalimentación ofrecida se basa en una tabla de correspondencias, que consisten en pares de referencias entre los elementos de los modelos que se considera que hay una relación, así como el valor de dicha relación. Inicialmente, MCTest ofrece diferente información de salida, fácilmente adaptable y ampliable:

1. La estructura interna de los dos modelos que se comparan en cada caso de prueba.

2. El tamaño de los modelos que se comparan.

3. El resultado óptimo de la comparación de ambos modelos.

[4]http://office.microsoft.com/en-us/excel/

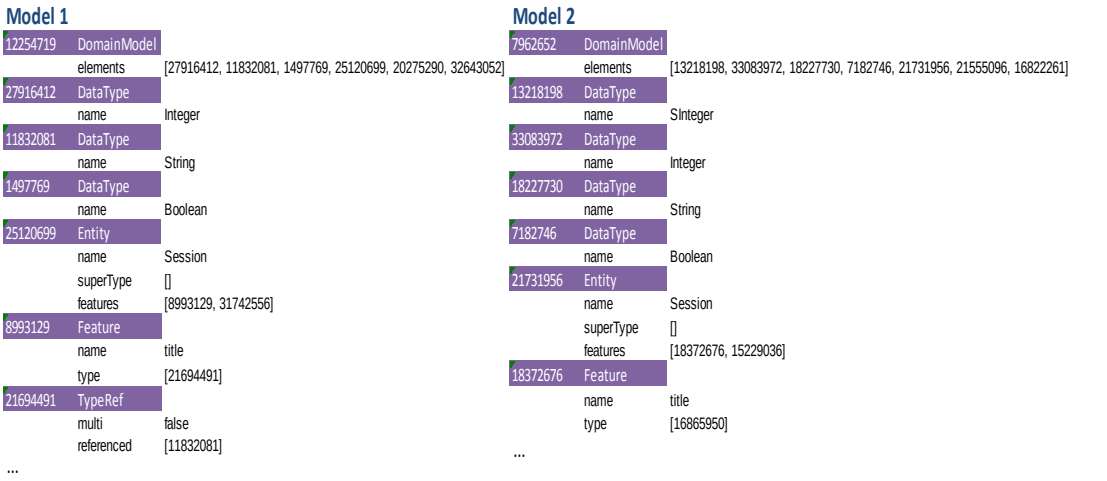


Figura 9.13: Fragmento de la representación de un modelo

4. Los resultados obtenidos para cada algoritmo probado.

5. Los resultados obtenidos comparados con el óptimo para cada algoritmo probado.

6. Los aciertos y fallos de cada algoritmo probado.

7. El porcentaje de acierto de cada algoritmo.

8. El tiempo empleado por cada algoritmo.

9. Una tabla resumen para cada caso de prueba.

La Fig. 9.13 muestra un fragmento de la visualización en formato Excel de los dos modelos de entrada que ofrece MCTest a partir de los modelos del primer caso mostrado en la Fig. 9.10. En el primer nivel de identación se muestran todos los elementos de los modelos, que se corresponden con el metamodelo inferido del DSL, identificados por el código Hash obtenido al ir procesándolos con el lenguaje de programación Java. Los atributos y referencias de cada elemento de la gramática están en el siguiente nivel de identación. Se diferencian porque los atributos tienen asignados un valor concreto mientras que las referencias apuntan al código Hash de otros elementos del modelo. Aunque la Fig. 9.13 no muestra toda la información de los modelos, se puede observar que la estructura mostrada se corresponde perfectamente con la gramática mostrada en la Fig. 9.9, que es, a su vez, utilizada por Xtext para generar el metamodelo correspondiente para el DSL.

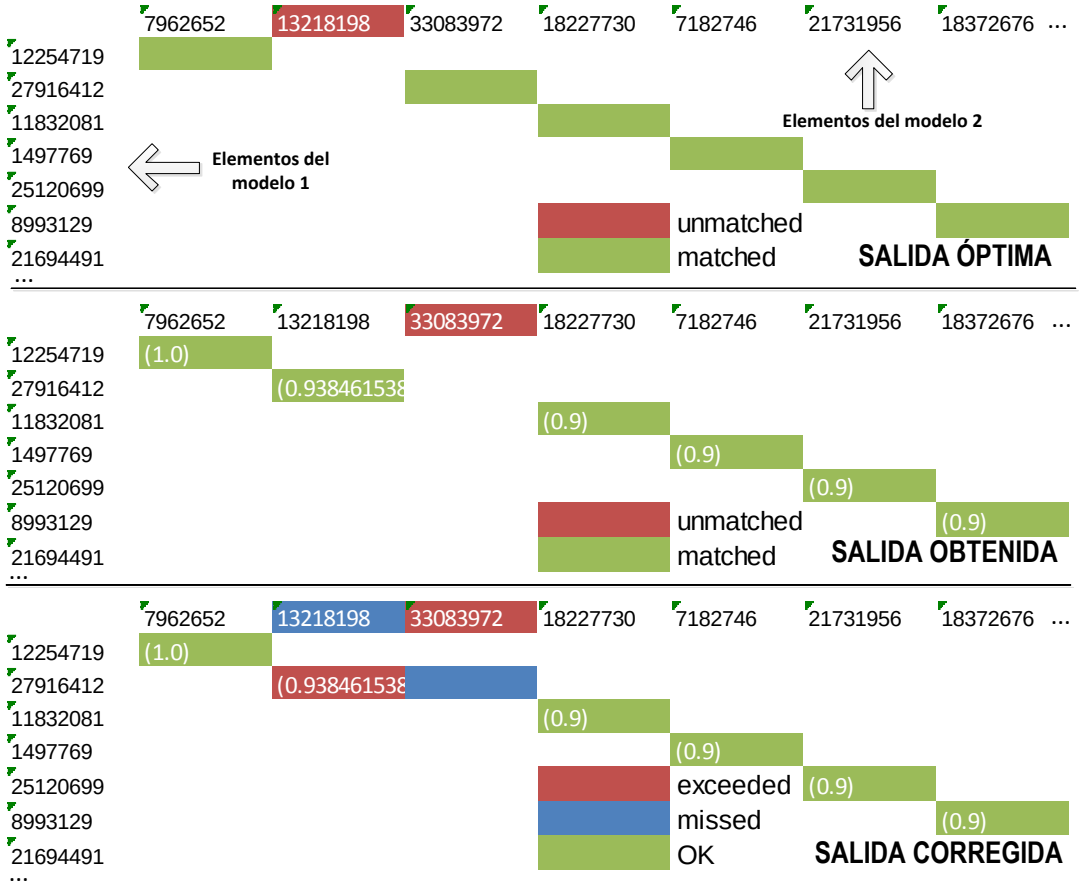


Figura 9.14: Fragmento de las tablas obtenidas en la salida

En la hoja Excel también se inserta automáticamente otra interesante información como la mostrada en la Fig. 9.14. Para cada algoritmo que se prueba en cada caso, se pueden visualizar tres tablas. La primera tabla es inferida automáticamente por MCTest a partir de la implementación óptima especificada. La segunda tabla es inferida automáticamente a partir de los resultados ofrecidos por el algoritmo de comparación de modelos de cada caso. Y por último, la tercera tabla también se obtiene automáticamente, en esta ocasión mediante la comparación de la primera y segunda tabla.

Una casilla marcada como *matched* representa una relación entre ambos modelos. Por ejemplo, en la salida óptima o esperada se representa una relación entre el elemento del primer modelo que tiene un Hash con valor 12254719 y el elemento del segundo modelo que tiene un Hash con valor 7962652. Cuando la casilla está marcada como *unmatched* (por encima del código de algún elemento de uno de los modelos) significa que esos elementos no aparecen en el otro modelo. Así, el elemento del

segundo modelo cuyo Hash es 13218198 no se corresponde con ninguno de los elementos del primer modelo. Dicho de otra forma, en este caso es un elemento nuevo que se ha añadido. Los valores numéricos que están dentro de las casillas representan el grado de semejanza que el algoritmo de comparación determina que existe entre los elementos de los dos modelos. El rango de valores utilizados comprende desde 0 hasta 1, con 1 como sinónimo de exactitud.

De las tres tablas, la que ofrece una información más relevante, en cuanto a la calidad de cada algoritmo, es la última. Las casillas marcadas como *OK* representan aciertos del algoritmo que se está probando respecto a la salida óptima especificada. Las casillas marcadas como *exceeded* representan afirmaciones incorrectas realizadas por el algoritmo que se está probando. Por último, las casillas marcadas como *missed* especifican las casillas en las deberían ir las afirmaciones correctas que el algoritmo no ha conseguido especificar, siempre comparadas con la salida óptima.

9.3.3. Evaluación

Para mostrar el funcionamiento de la herramienta MCTest, se ha realizado la siguiente prueba:

- Siete casos de prueba. Se ha creado un caso de prueba para cada uno de los problemas mostrados en el caso de estudio de la Fig. 9.10. A cada caso de prueba se le ha asignado la solución óptima esperada para que sea comparado con cada elemento de prueba.

- Cuatro elementos de prueba. Se ha extendido la clase *TestElement* para crear elementos de prueba que utilizan como base el algoritmo genérico que incorpora la herramienta EMF Compare. Después, se han creado otros tres elementos de prueba, que han sido configurados cambiando algún parámetro del algoritmo genérico. Los cuatro elementos de prueba han sido asignados a cada uno de los casos de prueba, obteniendo un total de 28 pruebas sin necesidad de realizar ningún tipo de configuración adicional.

Adaptabilidad de EMF Compare

La herramienta EMF Compare es altamente configurable. Por ejemplo, permite añadir nuevos heurísticos o eliminar alguno de los ya existentes. También, permite realizar modificaciones sobre ellos, hecho que podría ocasionar resultados muy diferentes en cuanto a las similitudes detectadas entre elementos.

Los heurísticos básicos están gestionados por clases que extienden a la clase abstracta *AbstractSimilarityChecker*. El heurístico principal es *StatisticBasedSimi-*

larityChecker, que hace comparaciones utilizando heurísticos estadísticos, siendo la opción empleada por múltiples herramientas que realizan comparaciones de forma genérica e independiente del metamodelo, tratando a los modelos como grafos. *DistinctEcoreSimilarityChecker* es una especialización de *StatisticBasedSimilarityChecker*, cuyo objetivo es trabajar con varios metamodelos al mismo tiempo. Tanto *StatisticBasedSimilarityCheckerd* como *DistinctEcoreSimilarityChecker* pueden ser decorados por otros heurísticos que los hacen ajustarse más a determinados requisitos. Así, *XMIIDSimilarityChecker* y *EcoreIDSimilarityChecker* sirven para realizar comparaciones utilizando un atributo identificador cuando está disponible, facilitando mucho el proceso de forma equivalente a como lo harían trabajos como [Alanen and Porres, 2003].

Los *checkers* se encargan de decidir si un elemento hace *match* con otro y en qué grado lo hace. Independientemente de si el *checker* básico es *StatisticBasedSimilarityChecker* o *DistinctEcoreSimilarityChecker*, hay una serie de parámetros que están directamente insertados dentro del código fuente y que podrían modificarse. Algunos de ellos son:

- El umbral general utilizado para determinar si un elemento de un modelo es similar a otro (por defecto 0.96). Todos los valores van de 0 (totalmente distintos) a 1 (idénticos), y vienen determinados principalmente por los resultados de los heurísticos.

- El mínimo número de elementos estructurales que un elemento debe tener para que se realice la comparación de dichos elementos con respecto a otros. Para ello se emplea un criterio basado en el contenido de los elementos (por defecto 5).

- El umbral para el que se considera que un elemento es casi igual a otro (por defecto 0.999999).

- Otros umbrales utilizados para considerar si dos elementos son similares o no (p.e., basados en el contenido, en las relaciones, en el número de atributos).

- Los pesos de los criterios de comparación basados en el contenido, en el nombre o en la posición de los elementos.

Además de añadir o eliminar heurísticos, también se podrían cambiar cada uno de los parámetros listados e incluso cambiar el algoritmo completo de comparación, que por defecto es el llamado *GenericMatchEngine*. Para ello, únicamente habría

que implementar la interfaz *IMatchEngine* y crear un algoritmo que haga uso de las estructuras de datos, técnicas, filtros o heurísticos que se consideren oportunos.

Todo lo anterior abre la puerta a una enorme cantidad de combinaciones que podría ser estudiadas, para por ejemplo mejorar el porcentaje de acierto del algoritmo de *match*, reducir tiempos de procesamiento, realizar estudios empíricos de determinados algoritmos o tratar de justificar el hecho de haber escogido un valor para un parámetro respecto a otro.

Dadas las innumerables combinaciones que pueden realizarse con EMF Compare, de la gran cantidad de otras herramientas que existen o que podrán existir en el futuro para comparar modelos, y de la evolución que puede existir en cuanto a las estructuras utilizadas para definir los propios modelos y/o metamodelos, MCTest puede facilitar la realización comparaciones y la obtención automática de resultados para que puedan ser visualizados, comprendidos, almacenados y tratados de forma inmediata.

Elementos de prueba

El primer elemento de prueba utiliza el algoritmo con los heurísticos que por defecto incorpora EMF Compare. Inicialmente se emplea el *XMIIDSimilarityChecker* decorando a *EcoreIDSimilarityChecker*, que a su vez decora a *DistinctEcoreSimilarityChecker*. No se ha cambiado nada de este algoritmo, por lo que el resultado obtenido sería el mismo resultado que obtendría cualquiera que hiciera las mismas pruebas con EMF Compare.

El segundo elemento de prueba utiliza un heurístico creado específicamente para la prueba, que se ha denominado *NoSimilarSimilarityChecker*. La idea es que siempre que el algoritmo pregunte al heurístico si dos elementos son similares, el heurístico responderá que no y que además no tienen absolutamente nada que ver.

El tercer elemento de prueba utiliza también un heurístico creado para la prueba, llamado en esta ocasión *SimilarSimilarityChecker*. La idea es la contraria al segundo elemento de prueba. En este caso, siempre que el algoritmo pregunte al heurístico si dos elementos son similares, el heurístico responderá que sí y que además son exactamente iguales.

Tanto el segundo como el tercer elemento de prueba pueden considerarse dos heurísticos pseudo-aleatorios que además deberían ejecutarse con una gran rapidez.

Por último, el cuarto elemento de prueba utiliza, al igual que el primer elemento de prueba, el algoritmo que por defecto incorpora EMF Compare. Sin embargo, el motor de ejecución permite realizar ciertas configuraciones básicas mediante una estructura de datos que contiene un *Map* con pares *<String, Object>*. En este caso,

se ha configurado para que en lugar de *DistinctEcoreSimilarityChecker* utilizara *StatisticBasedSimilarityChecked*, especialmente diseñado para los casos en los que sólo se utiliza un metamodelo.

Resultados

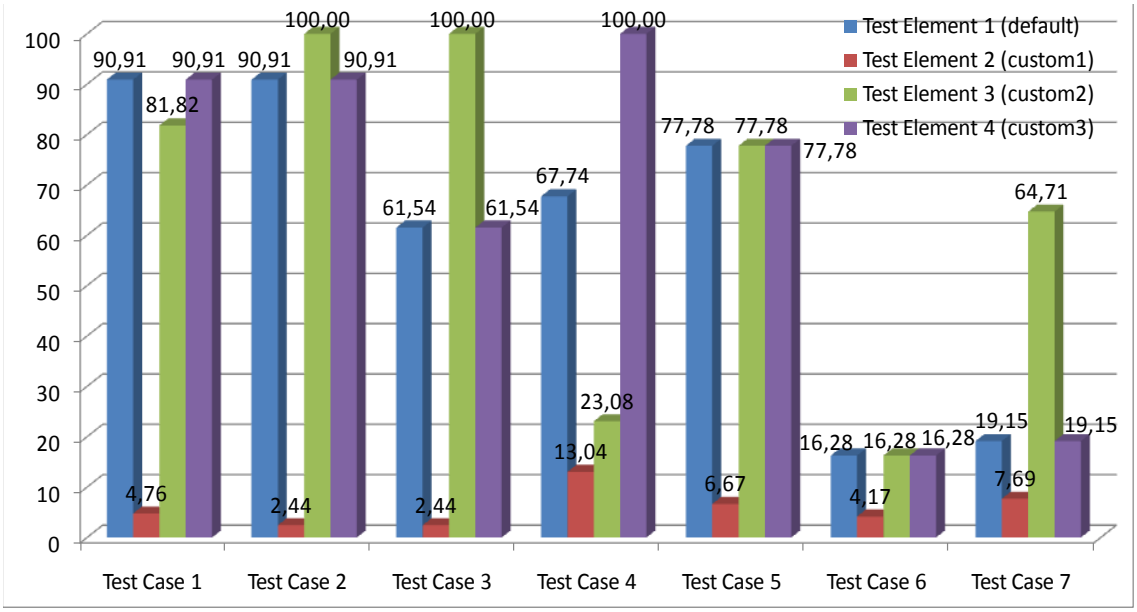


Figura 9.15: Porcentaje de éxito

La Fig. 9.15 muestra los resultados obtenidos para las 28 pruebas realizadas (siete casos de prueba con cuatro elementos de prueba) en cuanto al porcentaje de acierto de los algoritmos utilizados. Claramente, el segundo algoritmo es el que peor se comporta. Sin embargo, no es posible indicar claramente cuál es el mejor de los otros tres algoritmos ya que, dependiendo del caso, unos algoritmos se comportan mejor que otros.

La Fig. 9.16 muestra los resultados obtenidos para las 28 pruebas realizadas en cuanto al tiempo empleado por los algoritmos para realizar los cálculos necesarios. Sorprende el hecho de que el tercer algoritmo, un algoritmo pseudo-aleatorio, aún siendo mucho más rápido que el algoritmo que incorpora por defecto EMF Compare, pueda rivalizar con él en cuanto al porcentaje de acierto en prácticamente todos los casos de prueba. Este hecho revela que es posible mejorar el algoritmo genérico de EMF Compare.

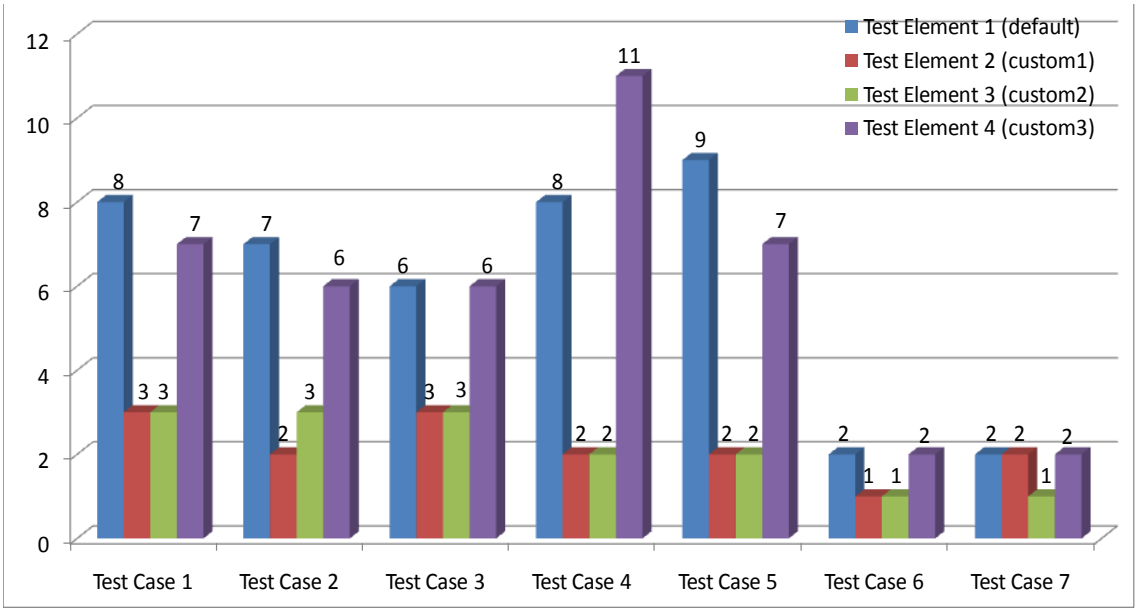


Figura 9.16: Tiempo empleado (ms)

9.4. Conclusiones

Aunque actualmente no existe un MVCS real para trabajar con una herramienta de CI, se ha realizado un prototipo que, pese a tener carencias que hacen que sea limitado, puede simular el funcionamiento de un MVCS. Así, se consiguen los siguientes beneficios:

- Se posibilita la definición de la granularidad de la unidad de versión utilizada por el sistema de control de versiones cuando se trabaja con un modelo software. De esta forma, no se guardan *falsas* nuevas versiones de un modelo, ganando fundamentalmente tiempo y espacio.

- Se evita que siempre que un usuario de la herramienta de modelado modifique un modelo, se inicie el ciclo de integración continua, siguiendo el funcionamiento de un VCS tradicional. De esa forma, se evita la pérdida de ciclos de CPU, la pérdida de tiempo, la ejecución de la herramienta de integración continua y la regeneración de artefactos no necesaria, que podría provocar la necesidad de volver a desplegar una aplicación.

Respecto a los algoritmos para comparar modelos y detectar las similitudes y diferencias entre ellos, hay un largo camino por delante que abre la puerta a nuevas investigaciones. Así, MCTest permite probar, comparar o estudiar dichos algoritmos de manera automática. Para ello, se utilizan parámetros como aciertos, fallos

o tiempo empleado por cada algoritmo. Cabe destacar que gracias al empleo de MCTest, ha resultado muy sencillo probar diferentes algoritmos con diferentes casos de prueba, mostrando los resultados en una interfaz de fácil comprensión.

Bloque V

Integración de MDE con herramientas de CI

Índice del bloque

CAPÍTULO 10

Problemática

Independientemente de la iniciativa MDE adoptada, hay un punto en el cual puede ser necesario generar artefactos (código fuente, documentación, ayudas, archivos de configuración, etc.) a partir de los modelos software. Ese es el motivo por el que la generación de artefactos es imprescindible en el contexto MDE, y por el que existen muchas herramientas para llevarla a cabo, como las citadas en [Palacios-González et al., 2008a] o [Palacios-González et al., 2008b].

En este capítulo se tratará la necesidad de generación de artefactos bajo el paradigma MDE. Se verán las técnicas utilizadas para realizar dicha generación, sus características y las herramientas más avanzadas que existen en la actualidad. Además, se dejará patente la falta de integración de este tipo de herramientas con los sistemas de integración continua actuales, ya que ninguna de las herramientas estudiadas puede utilizarse nativamente con ninguna de las herramientas de CI actuales.

⁂

10.1. Necesidad de la generación de artefactos

En [Völter, 2003a], se muestran varias razones que justifican la necesidad de generar artefactos, directamente relacionadas con las familias de productos software. Para los propósitos de este trabajo la razón más interesante es la siguiente:

> ❝*Se puede querer desarrollar una aplicación software utilizando un nivel de abstracción mayor que el de los lenguajes de programación tradicionales, por ejemplo porque se quiere que los expertos en un dominio sean capaces de "programar"la aplicación.*❞

Este motivo es muy interesante porque los desarrollos dirigidos por modelos que puedan ser integrados automáticamente permitirían a los expertos en un dominio no sólo programar, sino también hacerlo de manera *autónoma*, debido a que no necesitarían personal técnico para ayudarles a desplegar las aplicaciones. Al mismo tiempo, se guardarían copias de seguridad en los repositorios, o se podrían generar informes y documentación, todo de manera transparente al experto del dominio.

10.2. Técnicas de generación de artefactos

Dentro del proceso global de generación de artefactos, pueden existir diferentes tipos de transformaciones. La Fig. 10.1 muestra las más típicas. Para realizar estas transformaciones se utilizan varias técnicas o patrones [Völter, 2003a], que se resumen a continuación y que pueden ser utilizadas de manera individual o en combinación.

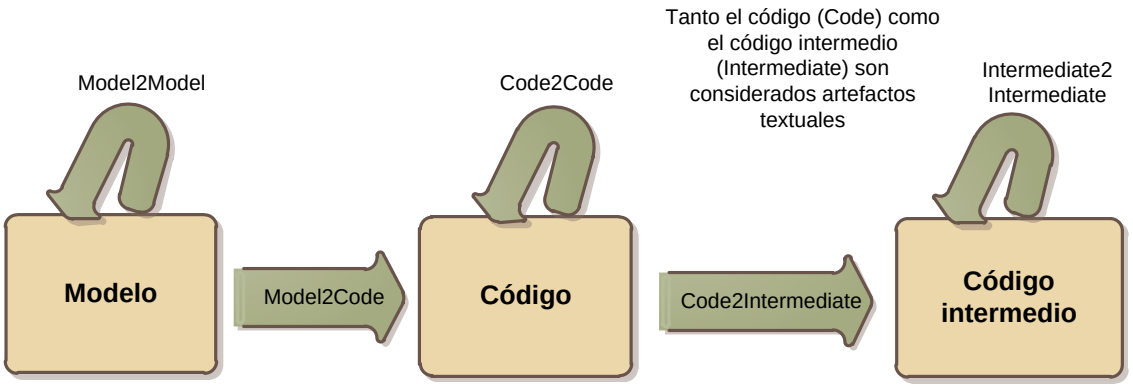


Figura 10.1: Transformaciones típicas entre artefactos

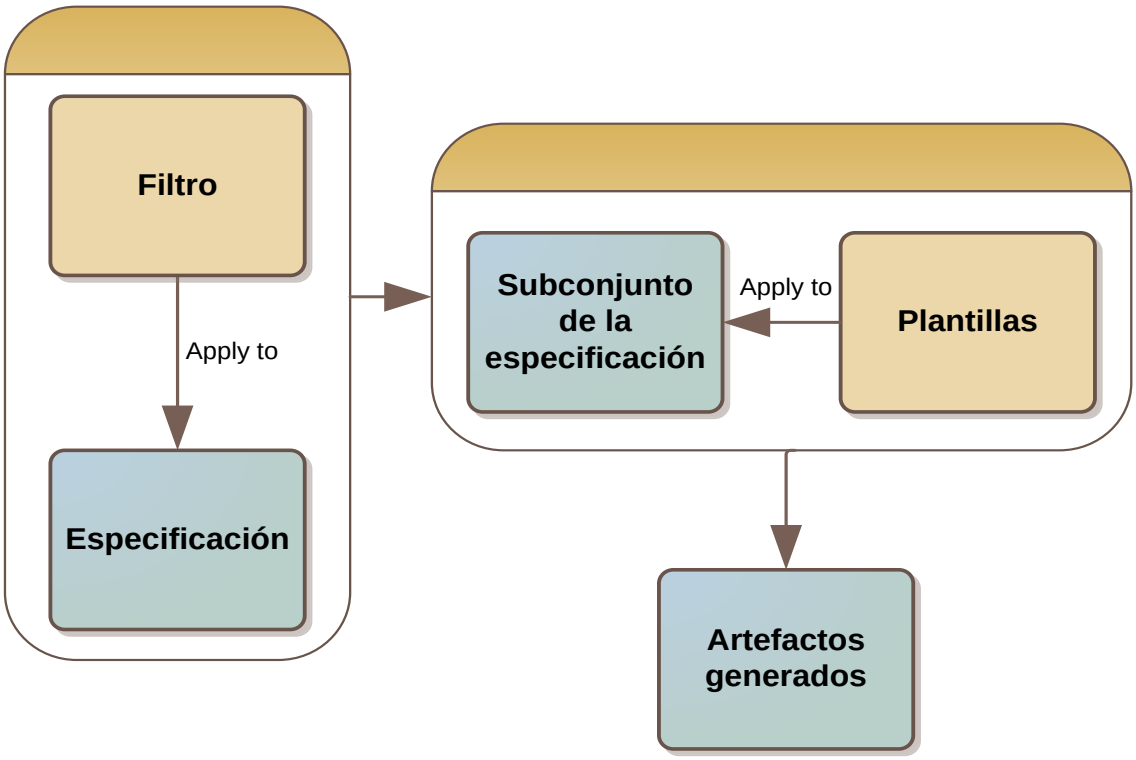


Figura 10.2: Generación mediante Plantillas

10.2.1. Generación basada en plantillas

Es la técnica más sencilla y también la más utilizada. Consiste en utilizar plantillas que contienen la parte de código fija que se generará a partir de una representación textual de un modelo. Adicionalmente, las plantillas podrán contar con partes variables que podrán ser ligadas a valores de la representación textual del modelo, teniendo en cuenta que posiblemente habrá mucha información en la representación textual del modelo que no interese y que será filtrada.

Un ejemplo muy típico es la aplicación de una hoja XSLT (Extensible Stylesheet Language Transformations) [Fitzgerald, 2003] a un modelo serializado en XML (en [Cleaveland and Cleaveland, 2001] se muestran ejemplos). Esta técnica se aconseja únicamente para pequeños modelos debido a que aplicarla a representaciones XMI de modelos MOF de gran tamaño puede ser una tarea muy complicada y costosa. Además, las plantillas dependerán de la sintaxis del modelo por lo que se complica aún más su uso.

En la Fig. 10.2 se muestra gráficamente esta técnica.

10.2.2. Generación basada en plantillas e instancias del metamodelo

Esta técnica añade cierta complejidad a la técnica anterior pero facilita el trabajo con modelos de gran tamaño. Primero se *parsea* el modelo representado textualmente (generalmente en XML) y se crea una instanciación del metamodelo. Por último se aplican las plantillas a la instancia del metamodelo. El uso de esta técnica acarrea importantes ventajas:

- Se gana independencia frente a la sintaxis concreta del modelo (p.e., diferentes versiones XMI de MOF).
- Se puede introducir lógica para comprobar restricciones en el modelo.
- Se puede utilizar un lenguaje de propósito general para trabajar con el metamodelo.

Un ejemplo de esta técnica puede verse en la herramienta oAW (openArchitectureWare) [Haase et al., 2007], en la que la sintaxis abstracta del metamodelo y las transformaciones con las plantillas son parámetros del compilador.

En la Fig. 10.3 se muestra gráficamente esta técnica.

10.2.3. Generación basada en el procesamiento de frames

Al igual que las clases pueden contener campos y pueden instanciarse, los *frames* [Bassett, 1996] pueden contener *slots* y se pueden instanciar. Se podría decir que son conceptos equivalentes. La principal diferencia es que los *frames* tienen una plantilla de código en su cuerpo, que cuando se evalúa en tiempo de ejecución, genera código. Son una alternativa a las plantillas pero de mayor complejidad debido a que, a diferencia de las plantillas, incorporan su propio lenguaje de programación.

Un ejemplo de esta técnica es el lenguaje de programación ANGIE[1]. Su procesador puede ser utilizado con éxito cuando se quieren desarrollar productos de una misma familia, siempre y cuando sean de pequeño tamaño.

En la Fig. 10.4 se muestra gráficamente esta técnica.

10.2.4. Generación basada en un API

Los generadores de código que utilizan un API se basan en tener una librería o biblioteca de clases que representa la sintaxis abstracta del lenguaje objetivo. Tienen

[1] http://www.d-s-t-g.com/neu/media/pdf/Facts_e/DLT21474.pdf

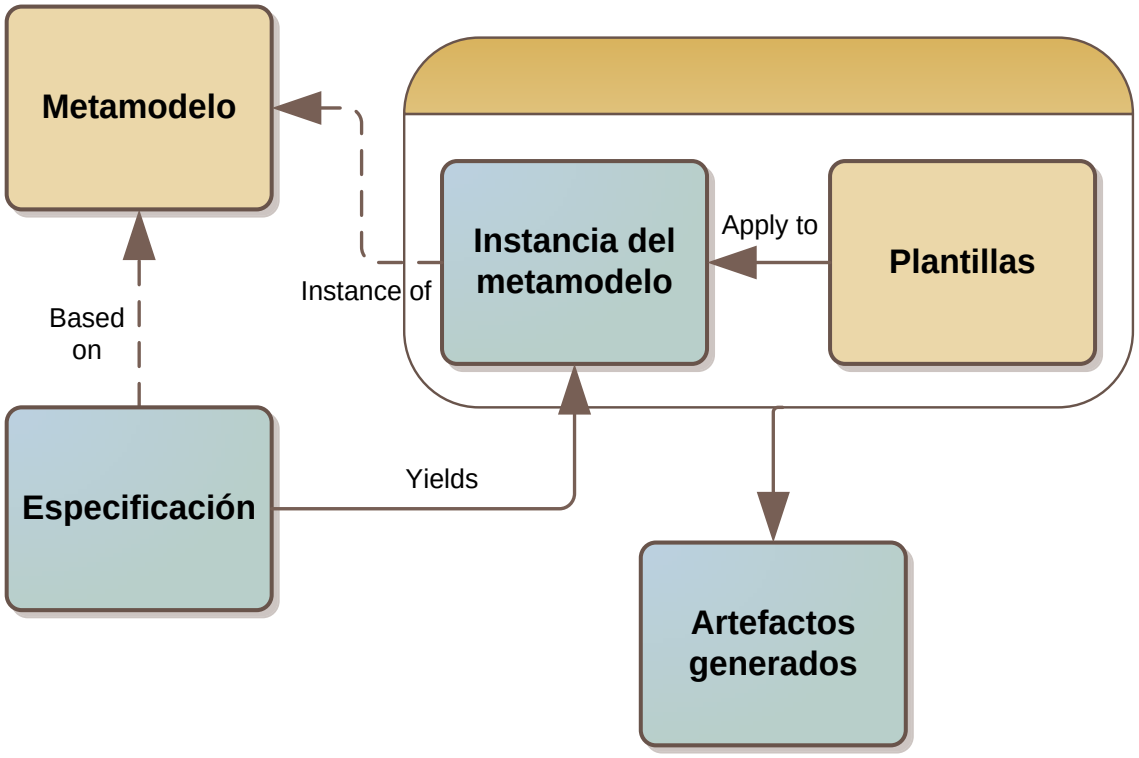


Figura 10.3: Generación mediante Plantillas y Metamodelo

la ventaja de que son sencillos de utilizar y que al utilizar un API, se puede garantizar que el lenguaje generado es correcto. Sin embargo, presentan inconvenientes como que son específicos del lenguaje reflejado en el API y que al no utilizar plantillas para especificar la parte fija de los artefactos a generar, generalmente es necesario utilizar una gran cantidad de código.

Un ejemplo es la herramienta CodeDOM [Dollard, 2004], que permite generar código para la sintaxis abstracta de los lenguajes de .NET basados en el CLR (Common-Language Runtime). Además, no está ligada a una sintaxis concreta ya que a partir del árbol de sintaxis abstracta que la herramienta genera internamente, puede generar código para diferentes lenguajes .NET mediante la implementación de una interfaz.

En la Fig. 10.5 se muestra gráficamente esta técnica.

10.2.5. Generación basada en atributos del código

Esta técnica consiste básicamente en un código no generado que es *adornado* con atributos o anotaciones, que sirven de especificación para poder generar código adicional basado en el código existente. Los atributos son necesarios porque sirven

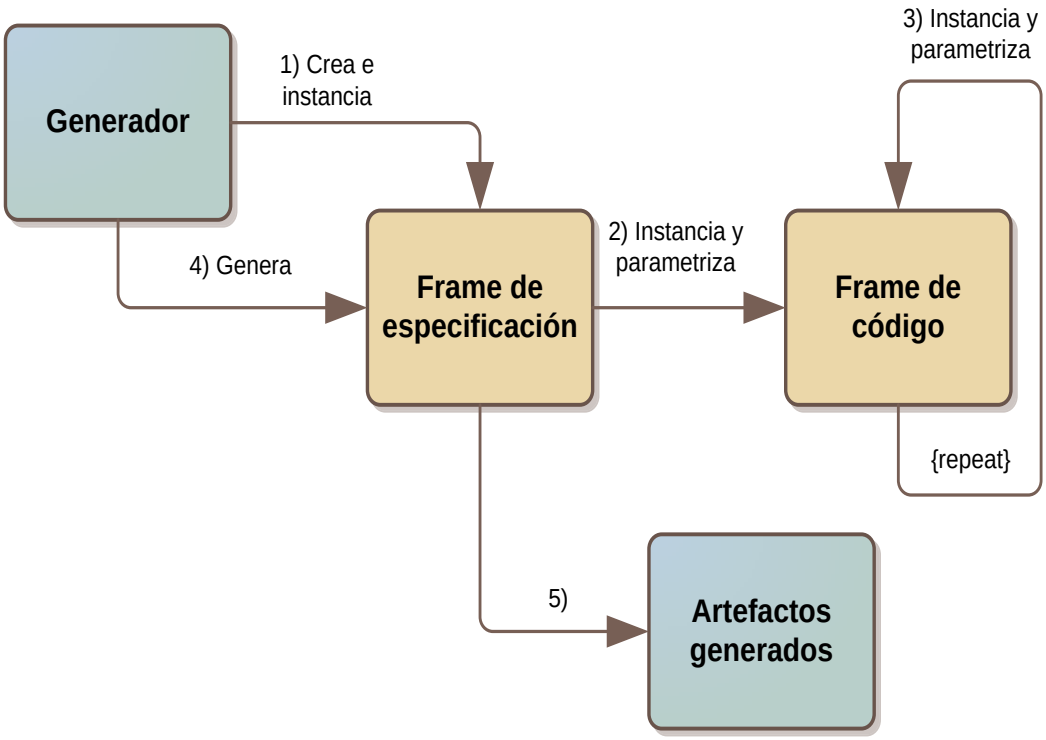


Figura 10.4: Generación mediante Procesamiento de Frames

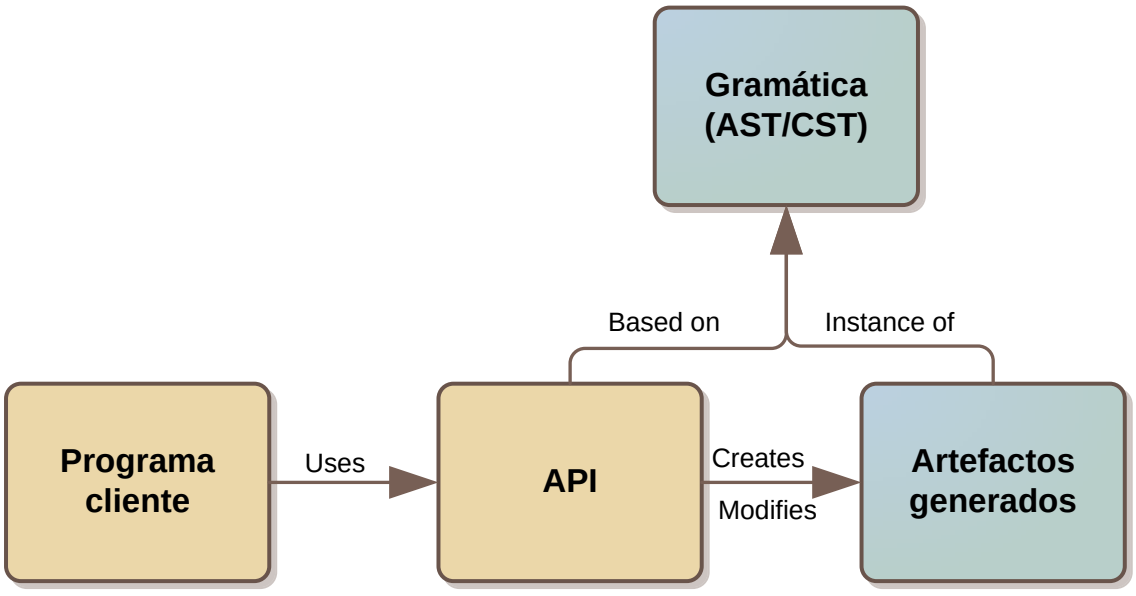


Figura 10.5: Generación mediante un API

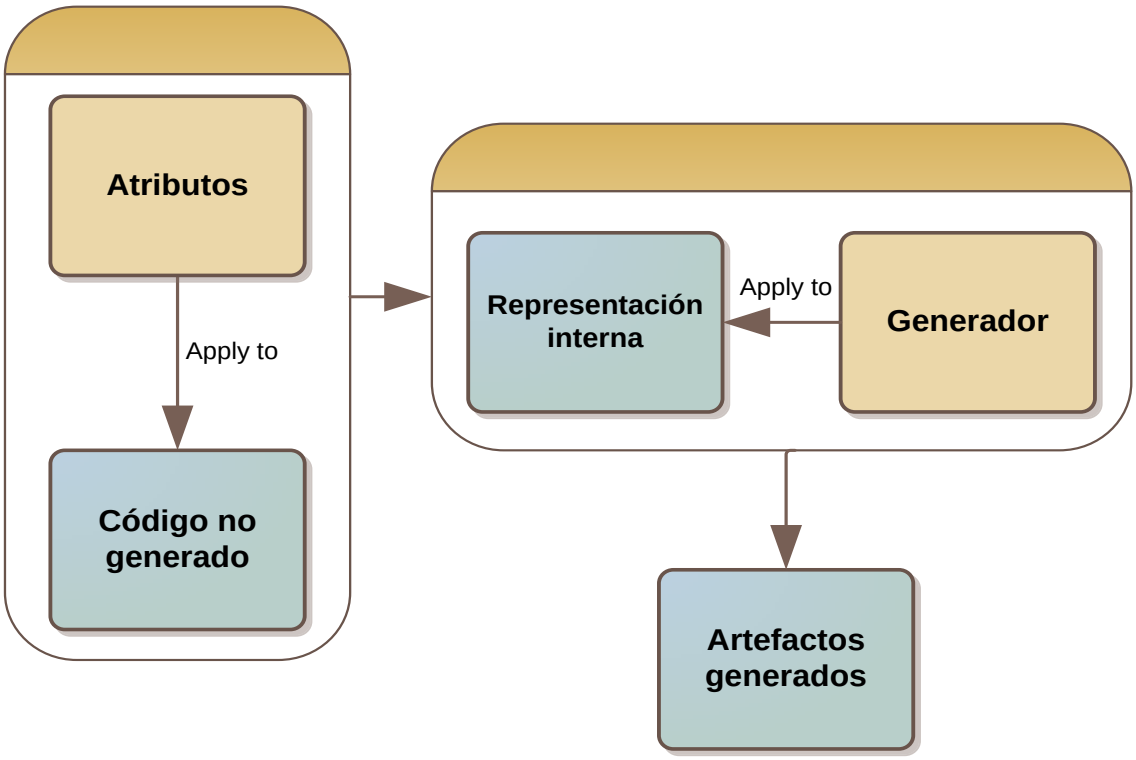


Figura 10.6: Generación mediante Atributos

para expresar acciones que no pueden ser expresadas por medio del lenguaje que se esté utilizando. La implementación de los atributos varía en función de las posibilidades del lenguaje que se emplee. Por ejemplo, en Java se utilizan comentarios especiales que empiezan con el símbolo @ y en .NET es posible definir atributos personalizados (son clases) que pueden ser compilados. Todos ellos son accesibles utilizando reflectividad, por ejemplo mediante CodeDOM o Reflection.Emit[2].

Uno de los ejemplos más conocidos es JavaDoc[3], que funciona en base a los comentarios que se hagan en el código fuente (aunque en lugar de generar código Java, genera código HTML).

En la Fig. 10.6 se muestra gráficamente esta técnica.

10.2.6. Generación en línea

En esta técnica la generación de código se hace implícitamente durante la interpretación o compilación de un código no generado o mediante un pre-compilador. Esta acción generalmente modifica el programa que posteriormente es compilado o

[2]http://msdn.microsoft.com/en-us/library/system.reflection.emit(vs.71).aspx
[3]http://java.sun.com/j2se/javadoc/

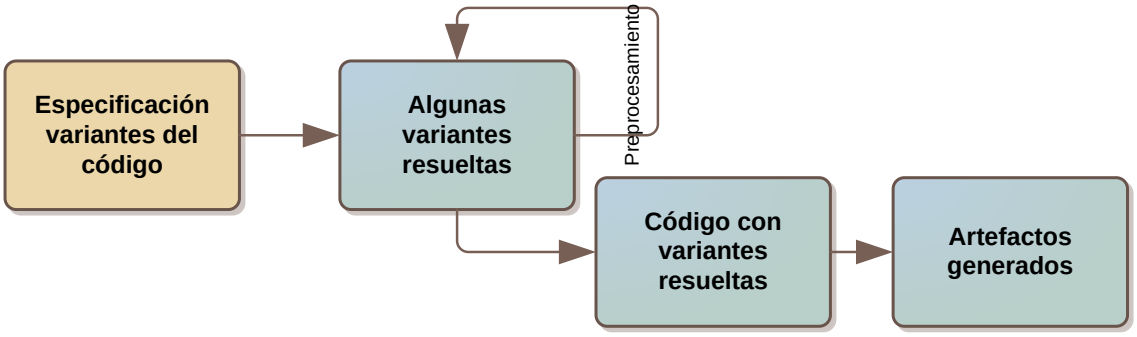


Figura 10.7: Generación en Línea

interpretado.

Un ejemplo es el preprocesador de C++, en el que se utilizan los elementos *#if* y *#endif* para indicar las condiciones bajo las cuales se compila el código que esté comprendido entre dichos elementos.

En la Fig. 10.7 se muestra gráficamente esta técnica.

10.2.7. Generación basada en tejido del código

El tejido de código consiste en combinar diferentes partes de código en un único elemento. Generalmente, dichas partes de código representan aspectos diferentes de un programa que podrán ser combinadas durante el tejido. El proceso se basa en la especificación de cómo las diferentes partes pueden unirse a través de los llamados *puntos de enlace*. Hay que resaltar que aunque tengan conceptos similares, el tejido de código es diferente a AOP (Aspect Oriented Programming), ya que hay otras formas de tejer código, diferentes a los aspectos. También es posible realizar AOP sin necesidad de generar código como por ejemplo mediante Smalltalk[4], que se hace de forma dinámica.

Un ejemplo muy típico de tejido es el lenguaje AspectJ [Kiczales et al., 2001], que es un lenguaje de programación orientado a aspectos construido como una extensión del lenguaje Java. Los aspectos se describen en Java extendido, generándose posteriormente código Java.

En la Fig. 10.8 se muestra gráficamente esta técnica.

[4]www.smalltalk.org/

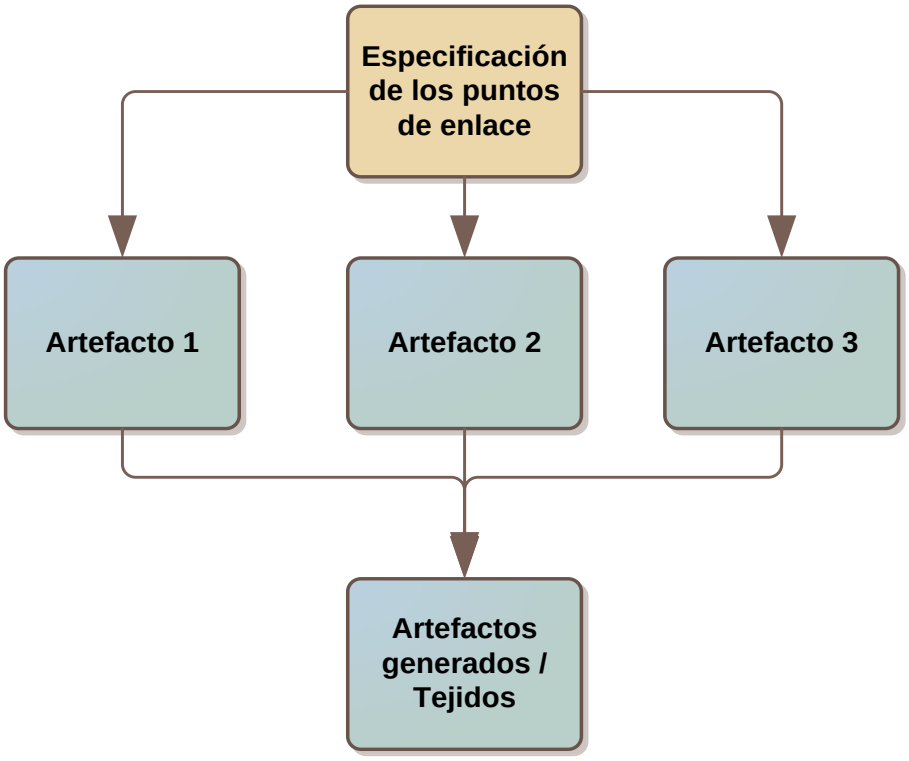


Figura 10.8: Generación mediante Tejido

10.2.8. Tabla de tecnologías de generación

Existen multitud de tecnologías y herramientas que, de una u otra forma, utilizan las técnicas mencionadas. En la Tabla 10.1 se muestran algunas de ellas.

10.3. Integración de artefactos generados y no generados

Mezclar artefactos generados y no generados puede acarrear problemas [Völter and Stahl, 2006]:

- En muchas ocasiones el código generado es muy difícil de entender y de modificar.
- Los VCSs sirven para guardar versiones de las fuentes de desarrollo. No tendría sentido guardar los artefactos generados (porque no son fuentes; las fuentes en ese caso son los modelos). Si se mezclan los artefactos generados y los no generados habría que guardar en el VCS también los artefactos generados, pudiendo llevar a problemas de inconsistencias entre modelos y artefactos.

Técnica/-Transform.	**Model2Mod.**	**Model2Code**	**Code2Code**	**Code2Inter.**
Plantillas	XML+XSLT	XML+XSLT	-	-
Plantillas + Metamodelo	oAW	oAW	OpenC++, OpenJava	-
Procesamiento de Macros	-	ANGIE	-	-
Basados en un API	APIs XML	Jenerator	CodeDOM	Reflection.Emit
En línea	-	-	Preprocesadores, Templates de C++	-
Atributos	-	XDoclet	-	.NET
Tejido	-	-	AspectJ, AspectC++	HyperJ

Tabla 10.1: Tecnologías de generación de artefactos

Por lo tanto, es conveniente tener separados los artefactos generados y no generados, e integrarlos en una etapa posterior. Existen varias técnicas o mecanismos para integrar el código [Völter, 2003a]. A continuación se indican algunas de las más utilizadas:

1. El código generado *llama* al código no generado por medio de librerías, de modo que se intenta generar el mínimo código posible, que confiará en los componentes implementados previamente.

2. El código generado hereda de clases del código no generado, pudiendo utilizar implementaciones de las clases del código no generado.

3. El código no generado *llama* al código generado por medio de librerías.

4. El código no generado *llama* al código generado, por medio de clases abstractas o interfaces que el código generado implementa.

Estas técnicas pueden ser combinadas según las necesidades de los proyectos, ya que lo importante es que queden bien delimitadas las responsabilidades y las interfaces teniendo una arquitectura claramente definida. En [Völter, 2003b] se puede ver un ejemplo de cómo combinar las diferentes técnicas de integración. Las *partial classes*[5] de C# es una técnica muy útil, ya que permiten definir clases separadas en diferentes archivos. C# también aporta otras técnicas muy novedosas e interesantes

[5]http://msdn.microsoft.com/en-us/library/wa80x488(VS.80).aspx

para separar el código en diferentes ubicaciones físicas como los *partial methods*[6] o los *extension methods*[7].

Como conclusión, se puede afirmar que las herramientas de generación de artefactos deberían separar los elementos generados de los no generados de forma clara, lo que hace que dicha característica sea de gran importancia en el momento de seleccionar una herramienta sobre las demás.

10.4. Ingeniería de ida y vuelta

RE (Round-trip engineering o Ingeniería de ida y vuelta) es un concepto íntimamente relacionado con la ingeniería inversa, la cual consiste en la reconstrucción del diseño de un producto a partir del mismo producto. Una definición dada por [Henriksson and Larsson, 2003] es:

> ❝*Siendo 'D' el diseño de un producto 'P', 'r' un proceso de ingeniería inversa de modo que r(P) = D y 'g' un proceso de generación de un producto de modo que g(D) = P. Si y sólo si g(r(P)) es equivalente a 'P' (y así r(g(D)) es equivalente a 'D') entonces (g, r) es un sistema de ingeniería de ida y vuelta.*❞

Además, RE es la base para nuevas técnicas como la llamada ARE (Automatic Round-trip Engineering), cuyos conceptos podrían ser útiles dentro del contexto de la ingeniería dirigida por modelos.

Sin embargo, autores como Markus Völter [Völter and Stahl, 2006] afirman que RE sólo es posible cuando el modelo y el código fuente representan la misma información desde dos o más puntos de vista diferentes. Así, RE no sería factible en MDE, donde los modelos tienen un mayor nivel de abstracción que los artefactos generados a partir de ellos. Al tener un nivel de abstracción mayor y emplear únicamente conceptos de un determinado dominio, nada asegura que los cambios realizados en el código puedan ser representados mediante la abstracción de los modelos. Por ello, se dice que MDE sólo tiene el sentido de ida [Völter and Stahl, 2006], no siendo importante el de vuelta. Así, los artefactos generados no deben ser modificados porque existen técnicas (véase sección 10.3) útiles para tener separados los artefactos generados y no generados.

Además, en el contexto de los DSLs, es totalmente recomendable que el código que se genera a partir de los modelos no tenga que ser modificado. De ser necesario, habría que refactorizar y cambiar el DSL y/o el modo de generar artefactos.

[6] http://msdn.microsoft.com/en-us/library/6b0scde8.aspx
[7] http://msdn.microsoft.com/en-us/library/bb383977.aspx

Si, pese a todo, se diera el caso en el que hubiera que modificar los artefactos generados manualmente, se deben emplear regiones protegidas (con comentarios en el código, atributos, etc.) que permitan diferenciarlos para que el generador no sobrescriba dichas regiones en posteriores iteraciones.

Así, se puede afirmar que la ingeniería inversa no es importante en cuanto a las herramientas de generación de artefactos MDE y, por tanto, no será una característica buscada en ninguna de ellas.

10.5. Herramientas de generación

Existen muchas herramientas cuyo objetivo principal es la generación de artefactos siguiendo los principios MDE. Para clasificarlas según sus características se han hecho varios trabajos. Stuart Kent [Kent, 2002] clasificó las diferentes herramientas según las funcionalidades que, según él, deberían ofrecer. Jean-Marc Jezequel [Jézéquel, 2005] hizo una clasificación según la técnica de transformación de modelos que aplica cada herramienta. Czarnecki y Helsen [Czarnecki and Helsen, 2003] establecieron una taxonomía para la clasificación de las herramientas MDA según el enfoque de transformación de modelos que utilizan. Naveed Ahsan Tariq y Naeem Akhter [Tariq and Akhter, 2005] elaboraron un estudio en el que citan las características que, también según ellos, han de tener las herramientas MDA, centrándose en las herramientas comerciales. A continuación se comentarán algunas de estas herramientas.

10.5.1. AndroMDA

AndroMDA[8] es una herramienta *Open Source* de generación de artefactos MDE. Soporta la utilización de UML y la importación/exportación mediante modelos serializados en XMI desde herramientas como por ejemplo MagicDraw o Poseidon. La transformación *model2model* se realiza mediante transformaciones definidas en Java pero se espera que se puedan utilizar otros lenguajes como por ejemplo QVT para definirlas. AndroMDA utiliza unos artefactos denominados *cartridges* para la generación de código. Existen muchos cartuchos disponibles para ser utilizados (p.e., Java, Spring, EJB2/3, .NET languages, Struts, XSD) y pueden ser extendidos mediante programación en lenguaje Java.

[8] http://www.andromda.org/

10.5.2. ArcStyler

ArcStyler es un IDE comercial utilizado a modo de ejemplo de aplicación en libros como [Hubert, 2001]. Soporta el ciclo de vida completo del desarrollo MDE, es decir, análisis, diseño, desarrollo y despliegue, basándose en modelos del negocio, reglas y lógica. Se utiliza XMI como mecanismo de intercambio de modelos, aunque también se puede utilizar J2SE o J2EE en un proceso denominado *harvesting*. No hay una clara distinción entre el PIM y el PSM y para realizar las transformaciones se utilizan los llamados *cartridges* o cartuchos escritos en JPython[9], permitiendo definir nuevos cartuchos o modificar los ya existentes para plataformas como Java o .NET. Al igual que AndroMDA, ArcStyler utiliza *cartridges*, pero en este caso se utilizan para las transformaciones entre modelos en lugar de para la generación de código. ArcStyler soporta herramientas como ANT, JUnit o Rational Rose.

10.5.3. Eclipse Modeling Project

Uno de los subproyectos más importantes del proyecto Eclipse[10] es el EMP (Eclipse Modeling Project) [Gronback, 2009], que engloba todos los temas relacionados con MDE dentro de la plataforma Eclipse. Tiene una serie de características como que todos los frameworks, herramientas o implementaciones de estándares dentro de EMP han de estar bajo la EPL (Eclipse Public Licence)[11] o que, pese a no estar ligado de ninguna manera al grupo OMG, implementa varios de sus estándares. El EMP se subdivide en varios proyectos de menor alcance, siendo algunos de los más importantes los siguientes:

Frameworks

- El núcleo central del EMP es el EMF (Eclipse Modeling Framework) [Budinsky et al., 2003,Steinberg et al., 2009], que proporciona el meta-metamodelo Ecore (muy próximo a MOF) y otras muchas herramientas relacionadas con él, como el soporte para QVT o XMI.

- El GMF (Graphical Modeling Framework)[12], sirve para definir editores gráficos personalizados basados en metamodelos definidos mediante el empleo de EMF.

[9] http://www.jython.org/
[10] http://www.eclipse.org/
[11] http://www.eclipse.org/legal/epl-v10.html - La EPL hace que todas las herramientas sean *Open Source* y de libre distribución
[12] http://www.eclipse.org/modeling/gmf/

- El TMF (Textual Modeling Framework), se utiliza para definir editores textuales personalizados.

Herramientas de desarrollo

- UML 2.x, implementación del metamodelo UML 2 basado en EMF.
- OCL, API para implementar restricciones en modelos basados en EMF.
- XSD Infoset, API para examinar, crear o modificar XML Schemas.

Herramientas de transformaciones

- M2M (Model-to-Model), herramientas para proporcionar la posibilidad de realizar transformaciones entre modelos.
- M2T (Model-to-Text), herramientas para transformar modelos a código u otros artefactos textuales.

Hay que resaltar que EMP no es realmente una herramienta de generación de artefactos, y sí un conjunto de herramientas y utilidades que otras herramientas pueden utilizar.

10.5.4. openArchitectureWare

oAW [Haase et al., 2007] es un conjunto de herramientas integradas para trabajar bajo el paradigma MDE en la plataforma Eclipse. Según sus autores es *a tool for building MDSD/MDA tools*. Por un lado utiliza tecnologías proporcionadas por el EMP como el EMF o el GMF, y por otro lado aporta nuevas tecnologías al EMP[13], como las siguientes:

- Xtend y Check al *M2M*.
- Xpand y Check al *M2T*.
- Xtext al *Textual Modeling Framework*.
- Workflow Engine, que da lugar al *MWE (Modeling Workflow Engine)*.

La Fig. 10.9 muestra la arquitectura general de openArchitectureWare.

[13]http://www.openarchitectureware.org/ - Dado su éxito, las principales herramientas de oAW han sido migradas recientemente al Eclipse Modeling Project

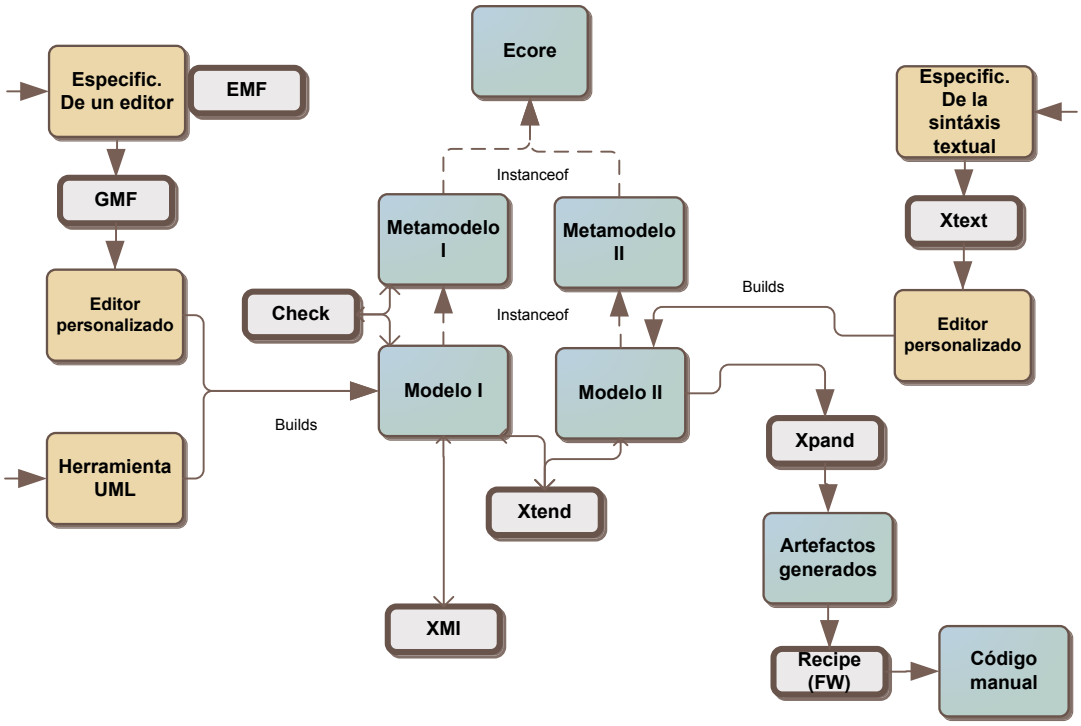


Figura 10.9: Arquitectura de openArchitectureWare

1. Si no se dispone de un metamodelo adecuado, hay que crearlo utilizando como tecnología subyacente el EMF.

2. Hay que crear modelos de los metamodelos. Para llevarlo a cabo hay basicamente cuatro alternativas:

   - Utilizar los editores basados en árbol que proporciona EMF. Son útiles para pequeños ejemplos pero no escalan bien.
   - Utilizar una herramienta UML tradicional que pueda ser adaptada a nuevos metamodelos mediante el uso de perfiles UML. Después, habrá que hacer una transformación entre el modelo UML y un modelo que conforme con el metamodelo creado.
   - Crear un editor gráfico mediante el GMF para trabajar con el metamodelo creado con EMF.
   - Crear un editor textual mediante Xtext para trabajar con el metamodelo creado con Xtext.

3. Los modelos creados con la herramienta que se haya utilizado se serializan a XMI.

4. Se puede utilizar el lenguaje *Check* (basado en OCL) para verificar restricciones de los modelos más allá de las estructuras definidas por los metamodelos. Las restricciones son reglas que los modelos deben cumplir para ser válidos (expresiones *booleanas*). Hay que tener en cuenta que el lenguaje Check se puede utilizar dentro de oAW o en editores de modelado externos.

5. Se utiliza el lenguaje *Xtend* para realizar transformaciones entre modelos (es una alternativa a otros como ATL, MTF o Tefkat).

6. Se utilizan las plantillas *Xpand* para generar artefactos textuales.

7. Se pueden emplear *Recipes* para controlar la generación manual de artefactos. Es una forma de ayudar a los desarrolladores a que desarrollen su código de la mejor forma posible, en relación con los artefactos generados.

A continuación se listarán ejemplos de las tecnologías de oAW, inspirados en su documentación:

Herramienta Modeling Worfklow Engine

Es un motor declarativo completamente configurable que sirve como punto central de las tareas que ejecutará el generador. Utiliza un lenguaje basado en XML para describir los diferentes componentes que se ejecutarán secuencialmente. Permiten ser invocados programáticamente, directamente desde Eclipse o desde la herramienta Ant.

```
//*****************************************************************************
<?xml version="1.0" encoding="windows-1252"?>
<workflow>
    <property file="workflow.properties"/>

        <component id="xmiParser" class="org.openarchitectureware.emf.XmiReader">
                <modelFile value="${modelFile}"/>
                <metaModelPackage value="data.DataPackage"/>
                <outputSlot value="model"/>
                <firstElementOnly value="true"/>
        </component>

        <component id="checker"
        class="org.openarchitectureware.check.CheckComponent">
                <metaModel id="mm"
                class="org.openarchitectureware.type.emf.EmfMetaModel">
                        <metaModelPackage value="data.DataPackage"/>
                </metaModel>
                <checkFile value="checks" />
                <emfAllChildrenSlot value="model" />
        </component>

        <component id="dirCleaner"
        class="org.openarchitectureware.workflow.common.DirectoryCleaner">
                <directory value="${srcGenPath}"/>
        </component>
```

```
        <component id="transformer" class="datamodel.generator.Transformer">
                <modelSlot value="model"/>
        </component>

        <component id="generator"
        class="org.openarchitectureware.xpand2.Generator" skipOnErrors="true">
                <metaModel
                id="mm" class="org.openarchitectureware.type.emf.EmfMetaModel">
                        <metaModelPackage value="data.DataPackage"/>
                </metaModel>
                <expand value="templates::Root::Root FOR model"/>
                <outlet path="${srcGenPath}/">
                        <postprocessor
                class="org.openarchitectureware.xpand2.output.JavaBeautifier"/>
                </outlet>
                <fileEncoding value="${fileEncoding}"/>
        </component>

        <component id="recipe" class="datamodel.generator.RecipeCreator"
        skipOnErrors="true">
                <appProject value="oaw4.demo.emf.datamodel.generator"/>
                <modelSlot value="model"/>
                <recipeFile value="recipes.recipes"/>
        </component>

</workflow>
//*****************************************************************************
```

Código fuente 10.1: Archivo `Workflow.oaw`

En el ejemplo (Cod. 10.1), se muestra un ejemplo del uso de los *workflows* de oAW. Como se puede observar, está formado por varios componentes precedidos por la carga de un archivo de propiedades que contendrá información que después podrá ser accedida mediante *${nombre-del-elemento}*. Los componentes, en orden, realizan las siguientes tareas:

1. Se carga un archivo XMI con información de un metamodelo creado con EMF.

2. Se realiza comprobaciones en los elementos de modelado que conforman con el metamodelo creado con EMF.

3. Se limpia el directorio en el que se generarán los archivos de salida.

4. Se realiza una transformación a partir del modelo mediante el uso del lenguaje Java.

5. Se genera código utilizando el metamodelo de la entrada y las plantillas creadas para realizar las transformaciones *model2text*.

6. Se crean *recipes* para asegurar que existe el código no generado adecuado que encaja con el código no generado. Así, podría ser necesario que existiera una clase que implemente una determinada interfaz generada por la herramienta. Mediante las *recipes*, implementadas en Java, se especifican necesidades o

restricciones que ha de tener el código generado manualmente por los desarrolladores.

Lenguaje Check

El lenguaje *Check* se utiliza para comprobar restricciones en los modelos. Es un lenguaje muy próximo a OCL.

```
//*************************************************************************
import data;

context Attribute ERROR "Names must be more than one char long" : name.length > 1;

context Entity ERROR "Names must be more than one char long" : name.length > 1;

context Entity ERROR "Names of Entity attributes must be unique":
        attribute.forAll(a1| attribute.notExists(a2| a1 != a2 && a1.name ==
        a2.name ) );
//*************************************************************************
```

Código fuente 10.2: Archivo `Checks.chk`

En el ejemplo (Cod. 10.2), se muestran tres restricciones realizadas sobre un metamodelo llamado *data*. Dichas restricciones representan que los nombres de las instancias de las metaclases *data::Attribute* y *data::Entity* tienen que tener más de un elemento. Además, los nombres de los atributos han de ser únicos. De lo contrario, se mostrará un mensaje de error.

Lenguaje Xpand

Xpand es un lenguaje de plantillas que sirve para generar artefactos en función de un metamodelo y que tiene características orientadas a objetos. Además, tiene un editor con ayudas como resalte de sintaxis o comprobación de errores.

```
//*************************************************************************
«EXTENSION templates::java»
«EXTENSION datamodel::generator::util::util»
«EXTENSION datamodel::helper»

«DEFINE Root FOR data::DataModel»
        «EXPAND Entity FOREACH entity»
«ENDDEFINE»

«DEFINE Entity FOR data::Entity»
        «FILE baseClassFileName() »
            // generatet at «timestamp()»
            public abstract class «baseClassName()» {
                «FOREACH attribute AS a»
                        private «a.type» «a.name»;

                        public void «a.setterName()» («a.type» value) {
                                this.«a.name» = value;
                        }

                        public «a.type» «a.getterName()»(){
                                return this.«a.name»;
                        }
```

```
                «ENDFOREACH»
            }
        «ENDFILE»
«ENDDEFINE»
//*****************************************************************************
```

Código fuente 10.3: Archivo `Root.xpt`

En el ejemplo (Cod. 10.3), se recorren todas las instancias de la metaclase *data::Entity* y se crea una clase Java para cada una de ellas.

Lenguaje Xtend

Xtend es un lenguaje que sirve para realizar transformaciones entre modelos y ofrecer puntos de extensión que permiten introducir lógica más allá de las plantillas. Por ejemplo, se pueden introducir características específicas de una plataforma que no van incluidas en el metamodelo.

```
//*****************************************************************************
import data;

create DataModel this duplicate(DataModel s):
entity.addAll( s.entity.duplicate() ) ->
setName(s.name);

create Entity this duplicate(Entity old):
attribute.addAll( old.attribute.duplicate() ) ->
reference.addAll( old.reference.duplicate() ) ->
setName( old.name );

create Attribute this duplicate(Attribute old):
setName( old.name ) ->
setType( old.type );

create EntityReference this duplicate(EntityReference old):
setName( old.name ) ->
setTarget( old.target.duplicate() );

//*****************************************************************************
```

Código fuente 10.4: Archivo `Trans.ext`

En el ejemplo (Cod. 10.4), se define una transformación que transforma un modelo en sí mismo.

Lenguaje Xtext

Xtext [Efftinge and Völter, 2006] es un framework que sirve para crear lenguajes de dominio específico textuales mediante la metamodelos basados en Ecore, editores textuales para ellos y *parsers*.

```
//*****************************************************************************
Model:
  (types+=Type)*;

Type:
  DataType | Entity;
```

```
DataType:
  "datatype" name=ID;

Entity:
  "entity" name=ID "{"
    (features+=Feature)*
  "}";

Feature:
  type=[Type|ID] name=ID;
//*****************************************************************************
```

Código fuente 10.5: Archivo `Mydsl.xtxt`

En el ejemplo (Cod. 10.5), se observa cómo se define una gramática que especifica el metamodelo y la sintaxis concreta que tendrá el lenguaje. A partir de dicha definición, se generará el *parser*, el editor y las clases que representan programáticamente al metamodelo.

10.5.5. ObjectiF

ObjectiF es una herramienta comercial realizada por microTOOL [Microtool, 2009], que da soporte a todo el ciclo de desarrollo: definir los requisitos, modelar el proceso de negocio, diseñar y refactorizar la arquitectura, modelar comportamiento dinámico y generar código probando los resultados. Hay versiones para Visual Studio y para Eclipse. La herramienta hace una clara distinción entre el PIM y el PSM, utilizándose diagramas UML de clases y de estados para definir el PIM. Las transformaciones entre PIM y PSM están predefinidas y soporta tecnologías como .NET, Structs, JBoss o Hibernate. Se puede generar código en Java, C++ o C#. La herramienta tiene funcionalidades avanzadas como soporte parcial a ingeniería inversa.

Además, una vez que se han generado artefactos a partir de los modelos, se pueden mantener sincronizados mediante la selección de los elementos de los modelos a partir de los cuales se quiere volver a regenerar código. Así, aunque de modo muy limitado, se realiza generación incremental de artefactos.

10.5.6. OptimalJ

OptimalJ [Compuware, 2005] es una herramienta comercial destinada a generar aplicaciones J2EE de manera automatizada. Existen dos versiones, una construida sobre Netbeans y la otra sobre Eclipse. Con OptimalJ se pueden generar aspectos estáticos y dinámicos de las aplicaciones mediante tres tipos de modelos:

- *Modelo del dominio.* Es el modelo equivalente al PIM del estándar MDA. Con OptimalJ, el PIM se representa únicamente con un diagrama de clases UML,

que viene integrado en la aplicación y que presenta ciertas limitaciones como la carencia de relaciones de *cero a muchos*. Hay dos modelos de dominio:

  - *Modelo de dominio de clases*. Especifica la parte estática, es decir, la estructura de la información con la que se trabajará (p.e., Coche, Usuario, Vendedor).
  - *Modelo de dominio de servicios*. Especifica la parte dinámica, es decir, las tareas típicas de negocio (p.e., Comprar, Vender, Alquilar).

- *Modelo de la aplicación*. Es el modelo equivalente al PSM del estándar MDA. Hay tres tipos:

  - *Modelo de presentación*. Muestra la interfaz de usuario. (tiene soporte para JSP, Structs, J2SE applet y Swing).
  - *Modelo de negocio*. Genera entidades de negocio (tiene soporte para EJB y DAOs -Data Access Objects-).
  - *Modelo de bases de datos*. Crea el esquema de la base de datos.

- *Modelo de código*. Es el modelo equivalente al ISM del estándar MDA, que contienen los artefactos finales generados (Java, XML, JSP, Entity Beans, Session Beans y SQL).

OptimalJ utiliza XMI para serializar los diferentes modelos y un lenguaje propietario denominado TPL (Template Pattern Language) para definir las transformaciones entre los diferentes modelos. En la Fig. 10.10 puede verse el aspecto general de OptimalJ.

El modelo de código tiene dos tipos de bloques:

- *Guarded Blocks*. Bloques de código generados con la herramienta que el usuario no puede modificar.

- *Free Blocks*. Bloques libres que el usuario puede modificar, de tal forma que cuando se vuelvan a generar los artefactos, dichas partes de código se respetarán y no se cambiarán.

El modelo del dominio y el modelo de código están sincronizados de tal forma que cuando se produce un cambio en el modelo de dominio, se cambia el código (mediante la sincronización activa). Además, si se hacen modificaciones en el modelo de código, siempre que sea posible, también se cambiará automáticamente el modelo

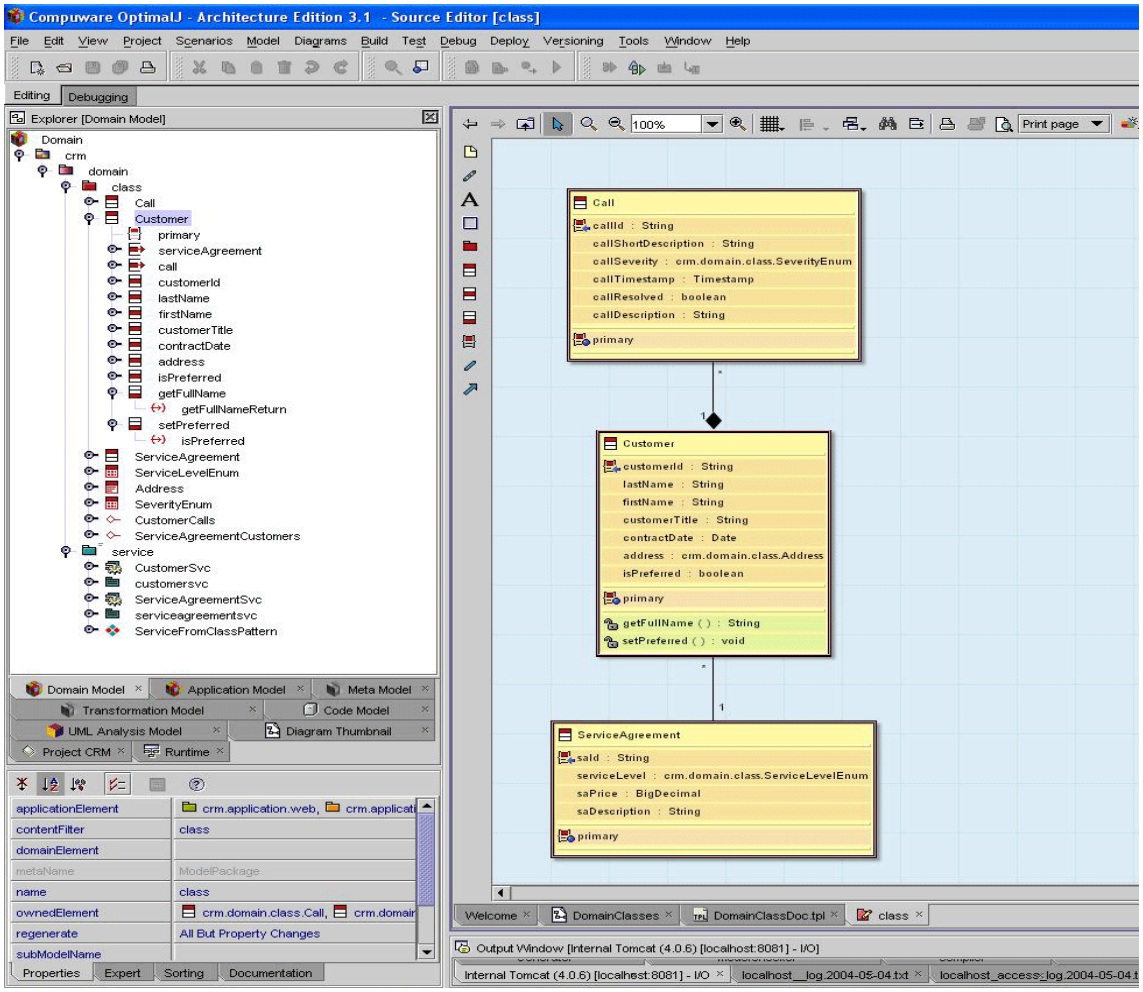


Figura 10.10: Aspecto de OptimalJ

del dominio. Esto es posible gracias a que la correspondencia entre ambos modelos es muy directa, simplificando mucho su realización.

La sincronización activa de OptimalJ sirve para conseguir que cuando alguien cambie el modelo del dominio, el modelo de la aplicación y el modelo de código también se modifiquen, permaneciendo los tres modelos en todo momento sincronizados. Puede lanzarse automática o manualmente y funciona de modo que se reconocen las partes de la aplicación que realmente son afectadas por los cambios en el modelo del dominio, regenerando únicamente los componentes afectados o creando nuevos componentes si fuera necesario. Con la sincronización activa se evita tener que regenerar todo la aplicación con cada cambio en el modelo de entrada.

Sin embargo, la sincronización activa de OptimalJ es muy limitada. La herramienta está orientada únicamente a generar aplicaciones J2EE con una estructura muy determinada y empleando un modelo de dominio con un nivel de abstracción

no muy elevado porque todos los elementos del modelo tienen la misma correspondencia con los elementos del modelo de la aplicación. Por esto, la realización de la sincronización activa se facilita con respecto a otras herramientas más generalistas y que trabajan con mayores niveles de abstracción mediante el uso de lenguajes de dominio específico.

10.5.7. Rational Rose XDE

Rational Rose XDE (eXtended Development Environment) [IBM, 2004] es un producto comercial de IBM totalmente integrado en Eclipse. Los modelos pueden ser creados utilizando la herramienta Rational Rose, que está integrada, pero también puede interactuar con otras herramientas gracias al estándar XMI. Posee una serie de *patrones de diseño* y *plantillas de código* que son intercambiables y que permiten generar código. Se pueden importar/exportar a través de un formato propietario de la herramienta. Además, la herramienta ofrece funcionalidades avanzadas como ingeniería inversa o generación incremental de código de forma parcial.

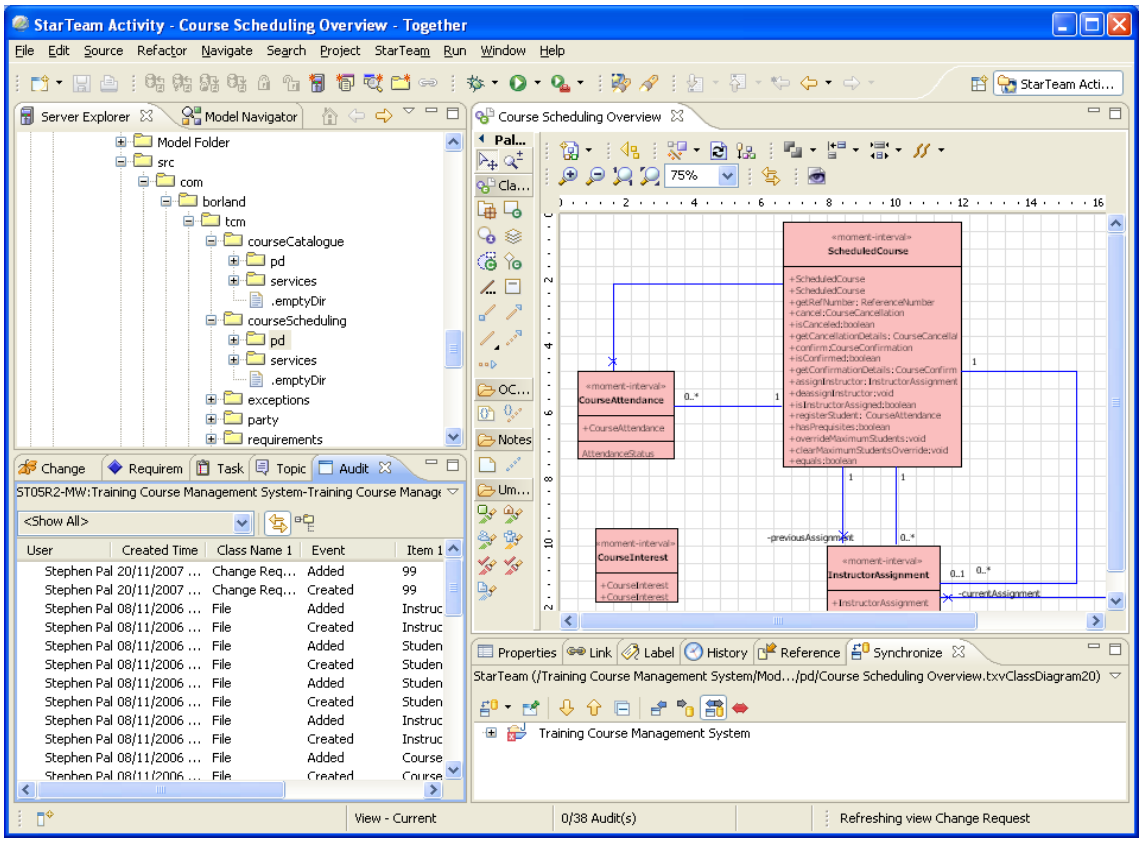


Figura 10.11: Aspecto de Borland Together

10.5.8. Together

Herramienta	Generación Incremental	Enlace
AndroMDA	-	http://www.andromda.org/
ArcStyler	-	http://www.arcstyler.com/
GMT	-	http://www.eclipse.org/gmt/
Kermeta	-	http://www.kermeta.org/
KMF	-	http://www.cs.kent.ac.uk/projects/kmf/
MOFScript	-	http://www.modelbased.net/mofscript/
Modfact	-	http://modfact.lip6.fr/ModFactWeb/
MTF	-	http://www.alphaworks.ibm.com/tech/mtf/
oAW	-	http://www.openarchitectureware.org/
objectiF	Limitada	http://www.microtool.de/objectiF/
OpenMDX	-	http://www.openmdx.org/
OptimalJ	Limitada	http://www.compuware.com/
Rational Rose XDE	Limitada	http://www-01.ibm.com/software/rational/announce/rose/
Together	Limitada	http://www.borland.com/us/products/together/

Tabla 10.2: Herramientas de generación de artefactos

Borland Together[14] es una herramienta comercial que sirve para realizar el modelado en UML y para generar código Java, C#, VB.NET, C o CORBA/IDL. Desde el 2006 está construido sobre Eclipse pero hay versiones que se integran con Visual Studio y BDS (Borland Developer Studio). Los modelos se pueden realizar con UML 1.4 y UML 2.0 o se pueden importar/exportar con XMI 2.0. También admite modelos importados desde Rational Rose o Rational XDE. Además, soporta BPMN (Business Process Modeling Notation), ER (Entity Relationships) e IDEF1x. La generación de artefactos se basa en la transformación *model2model* utilizando QVT y OCL mediante mapeos entre metamodelos que vienen con la herramienta y que no se pueden cambiar (los *scripts* QVT los tiene que hacer el usuario). También, se utilizan transformaciones *model2text* especificadas en Java desde un determinado metamodelo a texto.

En la Fig. 10.11 puede verse el aspecto general de Borland Together

10.5.9. Tabla de herramientas de generación

A modo de resumen, en la Tabla 10.2, se pueden ver algunas de las herramientas de generación de artefactos más importantes. En dicha tabla se especifican las he-

[14]http://www.borland.com/us/products/together/

rramientas que tienen características de generación incremental de artefactos (véase capítulo 12) y un enlace a dichas herramientas.

10.6. Conclusiones

Existen muchas técnicas que pueden ser utilizadas para generar artefactos a partir de modelos. También existen muchas herramientas que hacen uso de dichas técnicas para trabajar bajo la aproximación de desarrollo MDE. Sin embargo, ninguna de las herramientas vistas está pensada para trabajar con las herramientas de integración continua actuales sin la necesidad de una adaptación personalizada.

En el siguiente capítulo se presentará la solución para permitir utilizar una herramienta MDE con una herramienta de CI de forma eficiente. Para ello, se necesita obtener retroalimentación de las acciones realizadas por la herramienta MDE a través de la misma interfaz que proporciona la herramienta de CI para mostrar información de otras acciones realizadas por herramientas que sí tienen soporte nativo por parte de la herramienta de CI.

CAPÍTULO 11

Solución adoptada

Existe una gran cantidad de herramientas y utilidades, algunas tienen por objetivo permitir realizar la práctica CI (véase capítulo 4) y otras están dedicadas a generar artefactos a partir de modelos haciendo uso de la aproximación de desarrollo de software denominada MDE (véase capítulo 10).

Debido a que tanto CI como MDE son relativamente recientes y han empezado a tener gran importancia hace poco tiempo, aún no hay ninguna herramienta de CI que se integre de forma amigable con un generador de artefactos dirigido por modelos. Sin embargo, sí hay trabajos como [Völter and Stahl, 2006] o [Duvall et al., 2007] que han demostrado que ambas herramientas, por separado, mejoran la productividad y la calidad del desarrollo de software.

En este capítulo se mostrará la solución y la arquitectura utilizada para que la herramienta de integración continua y el generador de artefactos dirigido por modelos trabajen juntos de forma eficiente. Así, se obtendrá retroalimentación de las acciones realizadas, algo fundamental para ayudar a los desarrolladores y a los responsables de desarrollo a comprender qué acciones han ocurrido y por qué.

* * * *

11.1. Introducción

Para alcanzar el propósito planteado en este documento, se hace necesaria una integración entre la práctica CI y la aproximación de desarrollo MDE. Por tanto, son necesarias dos tipos de herramientas: por un lado una herramienta de CI y por otro lado una herramienta de generación de artefactos dirigida por modelos. Ambos tipos de herramientas existen y ofrecen mecanismos de extensión, por lo que se han adaptado haciendo uso de esos puntos de extensión, evitando la necesidad de crear nuevas herramientas.

Las interfaces de usuario son muy importantes en los desarrollos de software actuales y son el punto central de numerosos estudios (p.e., [Wade, 1984, Spinellis, 2002, Nguer and Spyratos, 2008]). El objetivo principal es encontrar un modo de mostrar información de manera eficiente y amigable para los usuarios de las aplicaciones [Shneiderman, 1987]. De ese modo, las herramientas de CI incorporan típicamente un módulo o componente que proporciona al usuario, de forma gráfica, la retroalimentación sobre las acciones ocurridas durante los procesos de integración continua que se van desarrollando.

Por defecto, las herramientas de CI no están diseñadas para trabajar con generadores de artefactos dirigidos por modelos. Por otra parte, la mayoría de las herramientas de CI permiten ejecutar aplicaciones a través de la línea de comandos, permitiendo utilizar cualquier generador que pueda ser ejecutado a través en modo consola. Sin embargo, de ese modo, no se podría interpretar directamente la salida generada mediante una interfaz amigable y homogénea, respecto a la ofrecida para el resto de herramientas para las que la herramienta de CI ofrezca soporte nativo y, por tanto, no podría detectarse, sin un esfuerzo adicional, si las operaciones han sido realizadas correctamente o no, o si el generador ofrece alguna información relevante. Sobre este punto, trabajos como [Néron et al., 2009] enfatizan en la necesidad de conseguir interfaces homogéneas para minimizar la complejidad en el uso de aplicaciones informáticas.

La idea principal es que la salida del generador sea ofrecida utilizando la misma interfaz gráfica que la salida de otras aplicaciones con soporte nativo por parte de la herramienta de CI. Como ejemplos podrían citarse las herramientas para realizar pruebas unitarias, análisis del código, generación de documentación o pruebas de cobertura.

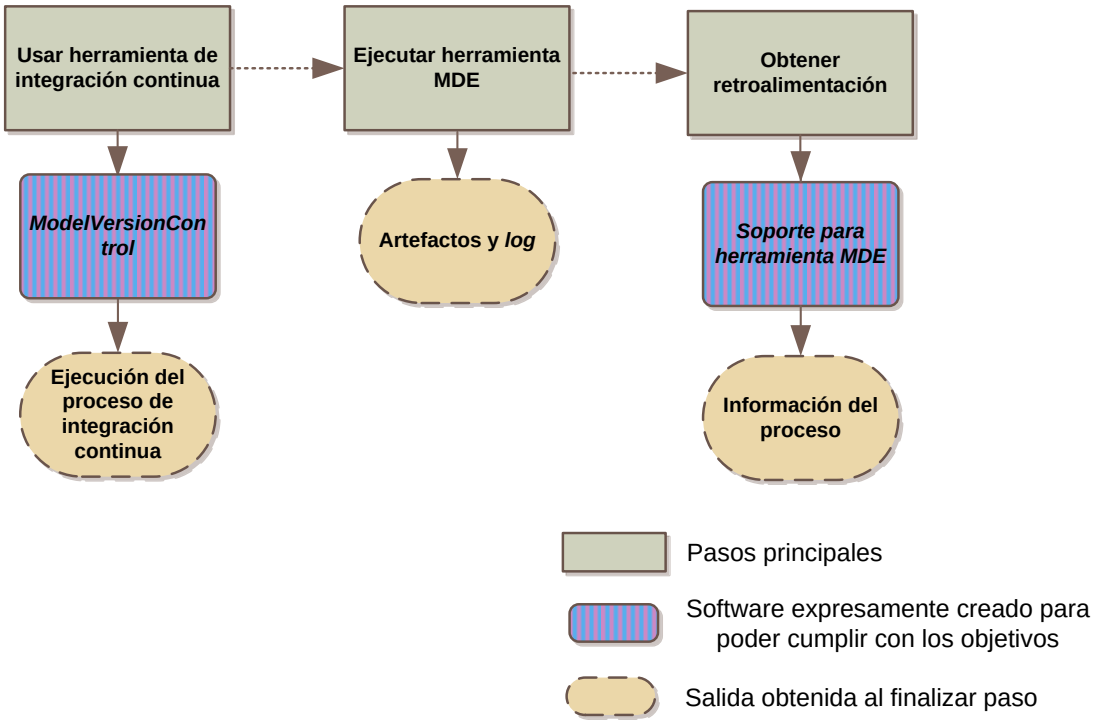


Figura 11.1: Pasos para relacionar CI con MDE

11.2. Arquitectura utilizada

Como se ha dicho (véase capítulo 9), se ha llegado a la conclusión de que Cruise Control (CC) es posiblemente la herramienta más utilizada y madura de cuantas existen para realizar la práctica CI. Además, es *Open Source* y ofrece mecanismos de extensión. Por todo ello, se ha optado por extender CC.

En el capítulo 10 se han mostrado herramientas que permiten generar artefactos en función a los diferentes modelos de entrada. Hay una gran variedad de herramientas pero en base a las necesidades de este trabajo, se ha optado por utilizar las provistas en el Eclipse Modeling Project[1], que anteriormente daban lugar a oAW[2]. Las razones principales son que se trata de las herramientas más próximas a los estándares existentes para modelos (MOF, UML, XMI, etc.) y que cumplen con la Eclipse Public Licence[3].

En la Fig. 11.1 se puede ver los tres pasos fundamentales y necesarios para la interacción entre CI y MDE.

[1]http://www.eclipse.org/modeling/
[2]http://www.openarchitectureware.org/
[3]http://www.eclipse.org/legal/epl-v10.html

11.2.1. Usar herramienta de integración continua

Para generar artefactos dirigidos por modelos, la herramienta de CI tiene que *vigilar* el repositorio *gobernado* por *ModelVersionControlSystem* (véase capítulo 9). Así, cuando detecta un cambio en dicho repositorio (mediante el empleo de *ModelVersionControl*), se comienza el resto del proceso de integración.

11.2.2. Ejecutar herramienta MDE

Cuando se detectan cambios en el repositorio, el siguiente paso será ejecutar la herramienta MDE. De ese modo, se obtendrán los artefactos generados y un archivo de registro de eventos, base de la retroalimentación que se mostrará al usuario de la herramienta. Para comenzar este proceso, se necesita modificar el archivo de configuración de CC (Fig. 11.2), para indicar el archivo del proyecto que se encargará de llevar a cabo las tareas de ejecución del generador MDE.

```
<tasks>
  <msbuild>
    <executable>C:/WINDOWS/Microsoft.NET/Framework/v3.5/MSBuild.exe</executable>
    <workingDirectory>D:/Ecommerce/v1.0/Config</workingDirectory>
    <projectFile>oAW.csproj</projectFile>
    <timeout>180000</timeout>
    <logger>C:/Program Files/CruiseControl.NET/server/ThoughtWorks.CruiseControl.MSBuild.dll</logger>
  </msbuild>
  ...
```

Figura 11.2: Configuración básica para utilizar la herramienta MDE

11.2.3. Obtener retroalimentación

En la Fig. 11.3 se puede observar la información obtenida por la herramienta de monitorización de CC cuando se generan artefactos dirigidos por modelos en el proceso de integración continua. Ha sido necesario adaptar ligeramente CC mediante el desarrollo de una extensión.

La información mostrada es la siguiente:

- El número total de acciones llevadas a cabo por la herramienta de CI.
- *Info*. En color verde se muestran los mensajes de información.
- *Warning*. En color amarillo se muestran los mensajes de advertencia.
- *Error*. En color rojo se muestran los mensajes de error. Lo cual indicará que no se ha podido llevar a cabo el proceso de generación de artefactos.

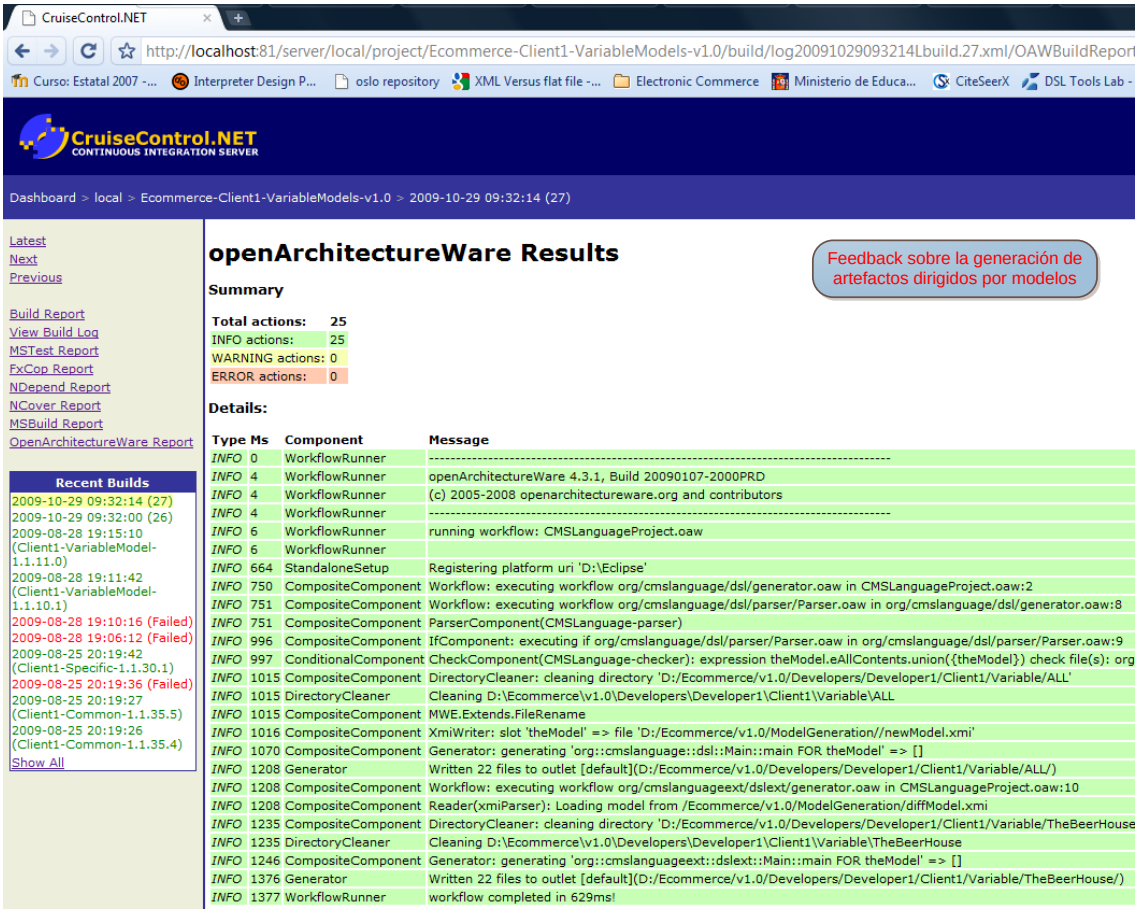


Figura 11.3: Retroalimentación obtenida en la herramienta de monitorización

Independientemente del tipo de mensaje, se utilizan cuatro columnas para mostrar la información:

- *Type*. El tipo de mensaje (*Info, Warning* o *Error*).
- *Ms*. El milisegundo en el que se produce dicho mensaje.
- *Component*. El componente (la clase) que ha producido el mensaje.
- *Message*. El contenido del mensaje.

Esta información muestra en detalle todo lo que el generador de artefactos ha realizado durante su ejecución.

11.3. Conclusiones

Se ha desarrollado un complemento o extensión que permite a una herramienta de CI obtener retroalimentación en un formato amigable y homogéneo de las acciones realizadas por la herramienta de generación de artefactos dirigida por modelos.

De este modo, los usuarios encargados de ello (jefes de proyecto, desarrolladores, etc.) podrán visualizar información sobre las acciones realizadas durante la integración continua de forma uniforme con respecto al resto de tareas realizadas durante dicho proceso, de igual modo que recomiendan trabajos como [Néron et al., 2009].

Bloque VI

Generación incremental de artefactos

Índice del bloque

CAPÍTULO 12

Problemática

Al trabajar con modelos, un aspecto clave es la generación automática de artefactos textuales de menor nivel de abstracción. Sin embargo, tradicionalmente la forma de llevar a cabo esta generación no tiene en cuenta la evolución de los sistemas a lo largo de su ciclo de vida, realizándose de forma poco flexible y muy repetitiva.

La generación incremental de artefactos en MDE hace referencia a la capacidad que pueden poseer los generadores para no tener que reconstruir todos los artefactos cada vez que se hace una modificación en un modelo, considerándose a los modelos como los elementos clave a partir de los cuales se construyen los sistemas software.

En este capítulo se mostrará el contexto de aplicación y el estado de la técnica en cuanto a la generación incremental de artefactos.

* * * *

12.1. Contexto de aplicación

Los modelos se transforman, generalmente de forma automática, a artefactos de menor nivel de abstracción (p.e., código fuente Java o documentación en formato HTML), facilitando así la generación de sistemas software [Völter and Stahl, 2006]. La idea básica es que un desarrollador crea un modelo y el generador se encarga de crear los artefactos que se correspondan con el modelo. Dichos artefactos podrían ser por sí mismos un sistema completo o un subsistema.

Los modelos, por tanto, están sometidos a los mismos desafíos que puede tener cualquier sistema software durante su evolución [Mens et al., 2005], y están sujetos a modificaciones a lo largo de todo su ciclo de vida debido, por ejemplo, a cambios en el negocio del cliente. Típicamente, si el desarrollador modifica el modelo a partir del cual se generan los artefactos, habría que volver a ejecutar el generador, volviendo a regenerar todos los artefactos, incluidos los que no han cambiado de una versión a otra del modelo. Esto es un problema en varios sentidos:

- No se optimizan los recursos computacionales de los que se dispone.
- Se dificulta el mantenimiento de aplicaciones en producción.
- Se imposibilita la realización de trazabilidad de los artefactos generados a lo largo de las diferentes iteraciones realizadas.

En la Fig. 12.1 se muestra una secuencia de 3 pasos en la que se ilustra cómo se comportaría un generador incremental de artefactos. Por simplicidad, se supondrá que cada elemento del diagrama conducirá a la generación de una clase con su mismo nombre en un lenguaje de programación orientado a objetos. En el caso 1 (caso inicial) habrá que generar las 4 clases A, B, C y D. Sin embargo, en el caso 2, habría que eliminar las clases C y D porque los elementos que las representan ya no están en el diagrama. En el caso 3, sin embargo, habría que añadir una clase E, puesto que se ha añadido en el diagrama un elemento para representarla, junto con una modificación producida en la clase B a través de B'. Pese a que, dependiendo del diseño del lenguaje, podrían existir dependencias entre los elementos que dificultarían la tarea de generación, se puede observar que siguiendo el esquema mostrado es muy probable que se ahorre tiempo en la generación de artefactos y que se facilite la modificación de aplicaciones ya en producción, reduciendo el impacto de los cambios en los modelos. Nótese que de otra forma, siempre habría que regenerar todos los artefactos con cada cambio. Manteniendo las diferentes versiones de los modelos, se podría incluso realizar trazabilidad de los cambios realizados, permitiendo saber

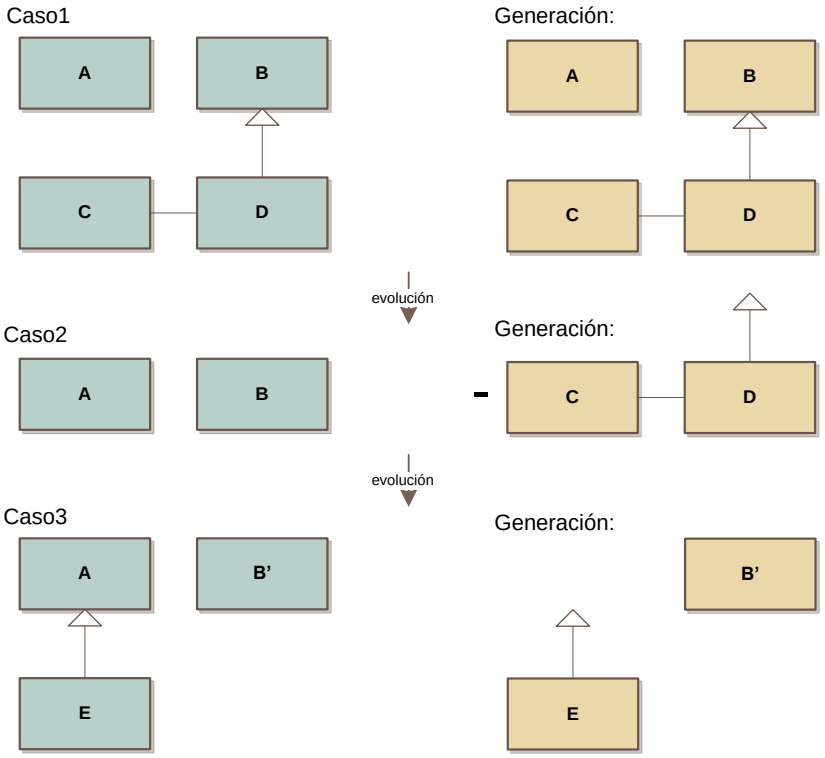


Figura 12.1: Ejemplo de secuencia de generación incremental de artefactos

qué se ha cambiado y cuándo, siendo una poderosa herramienta de mantenimiento y de depuración. Sin embargo, pese a las ventajas que puede suponer su empleo, la generación incremental de artefactos es un tema que aún está poco tratado por la literatura científica.

Es resumen, se puede afirmar que a mayor número de artefactos regenerados, aumenta la probabilidad de tener que recompilar código, rehacer pruebas, reanalizar la salida, etc. En otras palabras, a mayor número de artefactos regenerados, mayor será el impacto sobre las aplicaciones.

12.2. Herramientas actuales

Existen muchas herramientas dirigidas por modelos para generar artefactos[1]. Sin embargo, se han encontrado carencias en lo referente a la generación incremental en las más de 30 que se han evaluado, incluyendo las más utilizadas y citadas en trabajos como [Palacios-González et al., 2008a, Palacios-González et al., 2008b]. Por ello, aunque varias herramientas ofrecen características de generación incremental

[1]En la Tabla 10.2 se listan algunas de las herramientas más conocidas de generación de artefactos dirigidas por modelos, indicándose si tienen características de generación incremental

de artefactos, lo hacen de forma limitada, provocando que las herramientas actuales no estén preparadas para realizar cambios parciales en los artefactos generados.

Un ejemplo es objectiF[2], que sólo regenera las partes que se tocan en el modelo y que tiene carencias como que sólo admite modificaciones en tiempo de diseño o que no tiene en cuenta que un cambio en una parte del modelo podría afectar a otras partes.

Otro ejemplo es OptimalJ[3], que utiliza el mecanismo denominado *active synchronization* para asegurar que cuando un desarrollador cambia un modelo UML, el modelo y los artefactos J2EE previamente generados se sincronizan dentro del entorno de desarrollo proporcionado.

Por último, se podría citar una herramienta como EclipseUML[4], que tiene una tecnología propietaria integrada en Eclipse que permite diseñar sistemas software utilizando varios tipos de diagramas UML [OMG, 2007c]. A partir de alguno de los elementos de los diagramas se genera código fuente Java, que permanecerá sincronizado ante cambios en la vista de código o en la vista del diagrama.

En general, se ha observado que:

- Las herramientas con características de generación incremental de artefactos son herramientas muy específicas[5], en las que se combina la vista del modelo con la vista de los artefactos generados para un entorno de desarrollo determinado. Sin embargo, las herramientas de CI (véase capítulo 4) hacen una clara distinción entre las fuentes (es decir, los modelos) y la generación de artefactos, ya que dichas herramientas están situadas, lógica y físicamente, entre ellos. Por lo tanto, es necesaria una separación entre el modelado y la generación de artefactos:
  - Esta separación, será positiva porque los expertos en un dominio concreto únicamente tienen que preocuparse de modelar su negocio y, por tanto, no necesitan utilizar una herramienta que genere directamente artefactos de ningún tipo, debido a las dificultades que ello les podría suponer.
  - Cuando el resultado del modelado va directamente a un repositorio de modelos, se pueden controlar todas las versiones por las que pasan los modelos.
  - Los repositorios de modelos permiten el trabajo colaborativo y distribuido entre diferentes desarrolladores.

[2]http://www.microtool.de/objectiF/
[3]http://www.compuware.com/
[4]http://www.uml2.org/
[5]También son herramientas comerciales con tecnologías propietarias

- Para capturar la salida del generador y obtener retroalimentación de forma independiente al proceso, es necesario que sea la herramienta de CI la que se encargue de controlar la generación. Los modelos y otros metadatos serán parte de la entrada de la herramienta.

- No se han encontrado herramientas de generación de artefactos capaces de: procesar dos modelos en la entrada que conformen con un mismo metamodelo, calcular sus diferencias, representarlas de un modo adecuado y generar artefactos textuales de forma incremental en función de ellas.

- Las herramientas actuales tienden a generar artefactos de manera prefijada en función de una entrada, lo que significa que no tienen en cuenta la evolución de los modelos. De ese modo, una entidad *Coche* con una propiedad *Color = azul* podría generar el siguiente código SQL (Structured Query Language):

  INSERT INTO Items (tipo, color) VALUES ('coche', 'azul').

  Sin embargo, si se cambia la propiedad a *Color = rojo*, al regenerar el código, se generaría de nuevo:

  INSERT INTO Items (tipo, color) VALUES ('coche', 'rojo')

  Pero en este caso podría ser más adecuado generar otro tipo de sentencia porque lo que se quiere es hacer una modificación en función de los datos de entrada:

  UPDATE Items SET color = 'rojo' WHERE tipo = 'coche'

12.3. Estado de la técnica

Según [Hearnden et al., 2006], existen dos aproximaciones que permiten actualizar y sincronizar un modelo final[6] en base a los cambios producidos en un modelo inicial (Fig. 12.2).

12.3.1. Re-transformación

Con la re-transformación, cada actualización en el modelo origen (o inicial) requiere realizar una nueva transformación completa, produciendo un nuevo modelo objetivo (o final). Si se deseara actualizar un sistema software ya desplegado, habría que realizar una unión entre las dos últimas versiones de artefactos creados, lo cual

[6]Para los objetivos de este trabajo, este modelo se correspondería con los artefactos textuales generados

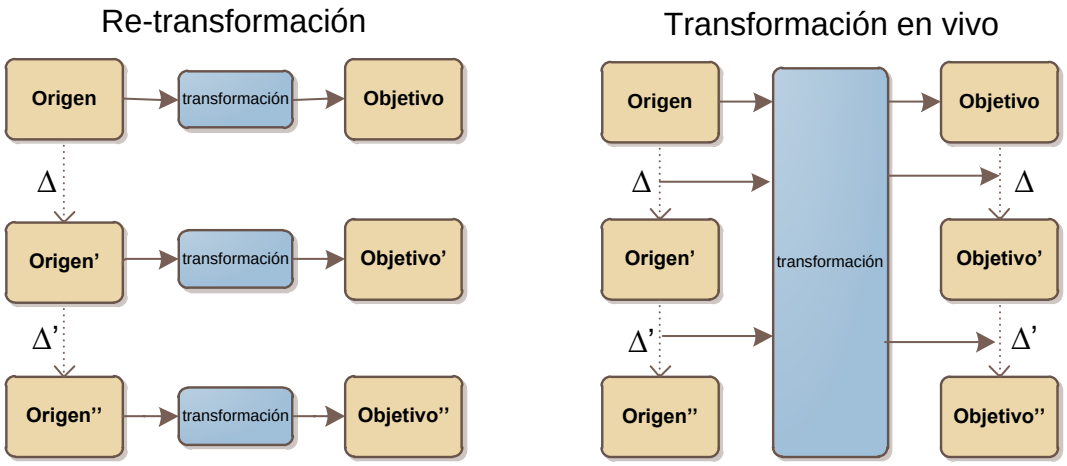


Figura 12.2: Estrategias para realizar actualizaciones incrementales en modelos

puede ser una tarea extremadamente compleja. Esta estrategia es muy básica, ya que se vuelven a generar todos los artefactos a partir de un modelo, y básicamente se limita a utilizar las tecnologías y herramientas existentes.

12.3.2. Transformación en vivo

Las transformaciones en vivo (*live transformations*) guardan el contexto de la primera transformación realizada sobre un modelo y lo siguen utilizando en futuras transformaciones, por lo que es un proceso continuado que no tiene un final definido. Así, no hace falta realizar la difícil unión entre los modelos objetivo porque se van realizando transformaciones parciales, únicamente en función de las variaciones producidas en el modelo origen.

Existen varias propuestas para realizar transformaciones en vivo. Por ejemplo, la propuesta presentada en [Hearnden et al., 2006] se basa en el empleo de un lenguaje de transformación declarativo basado en reglas y está fuertemente integrado con las estructura interna del motor de transformación llamado Tefkat [Lawley and Steel, 2006]. La idea es extender el motor de transformaciones basándose en el mecanismo estándar para la interpretación de lenguajes lógicos, es decir en la resolución SLD (Selection, Linear, Definitive), que es una restricción del principio general de resolución [Robinson, 1965]. En la Fig. 12.3 puede verse un ejemplo de árbol SLD extraído de [Hearnden et al., 2006]. La idea es que hay un objetivo G que consiste en un conjunto de literales, y hay un conjunto R que consiste en hechos y reglas. El

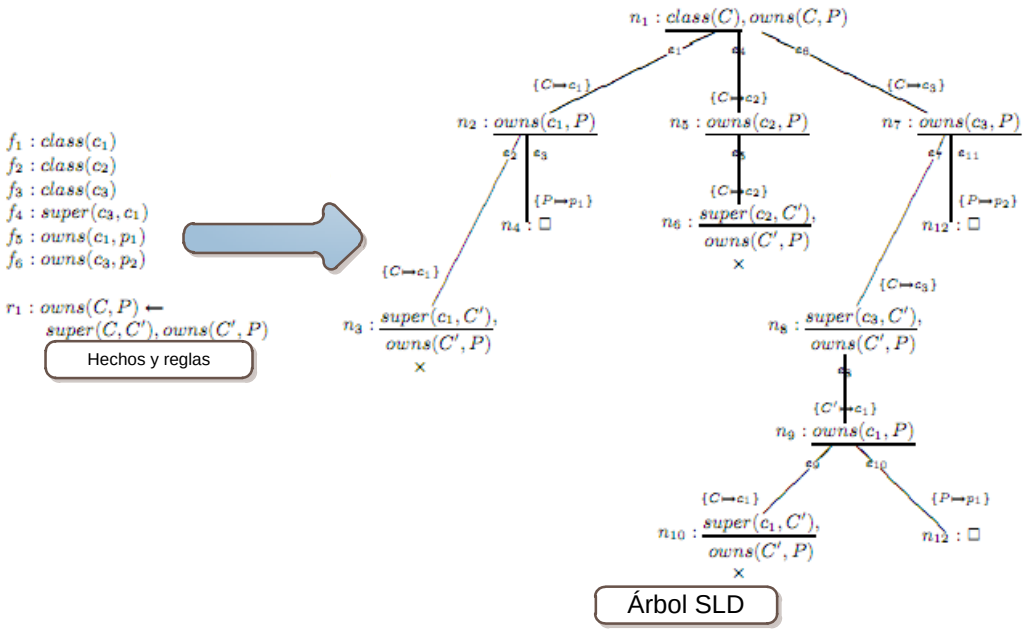


Figura 12.3: Árbol SLD (ejemplo)

algoritmo tiene que ir haciendo sustituciones hasta que se logra llegar a todos los objetivos vacíos (□), los cuales constituyen la solución final, o lo que es lo mismo, los artefactos que han de generarse a partir de una entrada. Cuando se producen cambios en el modelo inicial, dichos cambios se propagan al árbol, lo que significa que hay que guardar dicho contexto de ejecución para lograr la generación incremental de artefactos.

En [Ráth et al., 2008] se muestra otro trabajo perteneciente a la categoría de las transformaciones en vivo. La novedad de dicho trabajo consiste en que se presenta una aproximación basada en *incremental graph pattern matching* y en el manejo de transacciones complejas. Para ello, se emplea un framework de transformación de modelos denominado VIATRA2 [Varró and Balogh, 2007], que se basa en el algoritmo de las redes RETE [Forgy, 1990]. De cara al usuario, el lenguaje empleado está basado en la definición de restricciones, condiciones, reglas para definir manipulaciones elementales y reglas para la descripción de estructuras de control.

Ventajas de las transformaciones en vivo

Algunas de las ventajas de esta estrategia respecto a la primera son [Hearnden et al., 2006]:

- Mayor eficiencia, especialmente para cambios pequeños, realizando actualizaciones mucho más rápidas en los modelos objetivo.
- Menor necesidad de recursos computacionales para realizar la transformación.
- Es una solución más directa para encontrar los cambios en el modelo objetivo en función de los cambios en el modelo origen.

12.4. Conclusiones

Pese a que existen propuestas para facilitar la generación incremental de artefactos, éstas no se emplean habitualmente porque son trabajos que tienden a ser muy teóricos y que están orientados a la generación de modelos en lugar de artefactos textuales, lo que provoca un aumento de la complejidad en el empleo de las herramientas.

Sin embargo, aunque los generadores de artefactos dirigidos por modelos actuales no trabajan de forma incremental, existen varias herramientas que podrían facilitar dicha labor. Por un lado, aunque no suelen ser muy precisos, existen algoritmos para calcular la diferencia entre modelos. Por otro lado, existen técnicas que permiten representar las diferencias entre los modelos. Ambos aspectos, ampliamente estudiados en otros ámbitos de MDE, podrían utilizarse para la generación incremental de artefactos textuales.

En el siguiente capítulo se indicará cómo se ha afrontado la problemática detectada y cómo se le ha dado solución, permitiendo así, construir una parte fundamental de MDCIP (Model-Driven Continuous Integration Prototype).

CAPÍTULO 13

Solución adoptada

Las principales carencias que se han encontrado en las herramientas dirigidas por modelos para generar artefactos son:

- Las herramientas con características de generación incremental, son herramientas integradas y muy específicas. Un ejemplo es *objectiF*[1], que sólo regenera las partes que se tocan en el modelo y que tiene carencias como que sólo admite modificaciones en tiempo de diseño. Del mismo modo, no tiene en cuenta que un cambio en una parte del modelo podría afectar a otras partes del mismo.

- Las herramientas estudiadas generan artefactos de manera prefijada en función de una entrada, lo que implica que no se tiene en cuenta la evolución de los modelos. Así funciona, por ejemplo, oAW [Haase et al., 2007].

- No se han encontrado herramientas capaces de procesar dos modelos en la entrada que conforman a un mismo metamodelo, calcular las diferencias, representar de un modo adecuado y generar artefactos en función de ellas. Por ejemplo, la propuesta de [Cicchetti et al., 2007b] sólo está pensada para representar cambios, mientras que la herramienta EMF Compare [Toulmé, 2007] está más orientada a comparar modelos.

En este capítulo se mostrará la solución adoptada [García-Díaz et al., 2011c] tratando un aspecto considerado clave para la generación incremental de artefactos, la representación de las diferencias entre diferentes versiones de un mismo modelo[2]. También se verá la arquitectura utilizada y un caso de estudio para ofrecer una propuesta a la necesidad de herramientas de generación incremental de artefactos

[1] http://www.microtool.de/objectiF/

[2] Técnicamente no tendrían por qué ser siempre diferentes versiones, ya que podría seguir considerándose la misma versión en función de la unidad de versión que se haya decidido utilizar (véase sección 8.3)

dirigidas por modelos. Estas herramientas, al igual que ocurre con los MVCS, permitirán mejorar la forma en la que la integración continua se realiza cuando se trabaja con MDE.

⁂ ⁂ ⁂ ⁂

13.1. Introducción

La propuesta de este capítulo ofrece las siguientes características, que ninguna de las herramientas evaluadas, soporta en su totalidad:

- Generación de artefactos en función de la evolución de los modelos. Las herramientas actuales tienden a generar artefactos de manera prefijada en función de una entrada. Por ello, no tienen en cuenta si un elemento en el modelo se ha cambiado, se ha añadido, se ha borrado o si ya estaba en el modelo en la última versión de dicho modelo.

- Separación entre el modelado y la generación de artefactos. En muchas de las alternativas, se combina la vista del modelo con la vista de los artefactos generados para un entorno de desarrollo específico. Sin embargo, el usuario de la herramienta de modelado no tiene por qué querer o necesitar generar artefactos cuando modela su negocio. Del mismo modo, las herramientas de integración continua [Herbsleb and Grinter, 1999] hacen una clara distinción entre las fuentes (es decir, los modelos) y la generación de artefactos, ya que dichas herramientas están situadas lógica y físicamente entre ellos.

- Flexibilización de los metamodelos utilizados. No es conveniente restringirse al empleo de un único metamodelo (p.e., UML) sin tener la posibilidad de incorporar otros.

- Generación de artefactos independiente de plataforma. Algunas herramientas únicamente generan artefactos incrementalmente para una arquitectura y plataforma específica (p.e., para J2EE).

- Generación de artefactos flexible. Parte de las herramientas estudiadas generan artefactos de forma prefijada, imposibilitando cambiar los artefactos que se generarán a partir de un modelo de entrada, lo cual hace que la generación sea excesivamente rígida.

- Implementación abierta no comercial. Varias herramientas utilizan tecnologías propietarias comerciales que imposibilitan el estudio de su arquitectura interna.

El trabajo presentado en [García-Díaz et al., 2009b] puede ser visto como el antecedente al trabajo realizado en este capítulo. El objetivo es especificar la forma en que los cambios en una parte de un sistema conducirán a la generación de unos

u otros artefactos. Sin embargo, la propuesta presentada en este capítulo será más genérica y hará uso de las tecnologías actuales, que evitarán tener que crear un lenguaje y una infraestructura para el mismo, como se indica en [García-Díaz et al., 2009b].

13.2. Transformación directa

En el capítulo 12 se mostraron las dos aproximaciones principales para generar artefactos de manera incremental: la re-trasformación y la transformación en vivo.

La re-transformación es tan básica que consiste en volver a generar todos los artefactos y después tratar de hacer la operación *merge* a partir de los artefactos que ya estaban generados con otra versión del modelo original, lo cual difícilmente puede ser visto como una generación incremental, y puede llegar a ser extremadamente complicado.

Por otra parte, las transformaciones en vivo actuales están basadas en complicados lenguajes fundamentados en reglas, en la modificación de motores de transformaciones y en la necesidad de guardar el contexto de ejecución. Dichas tareas aumentan excesivamente la complejidad de la aproximación.

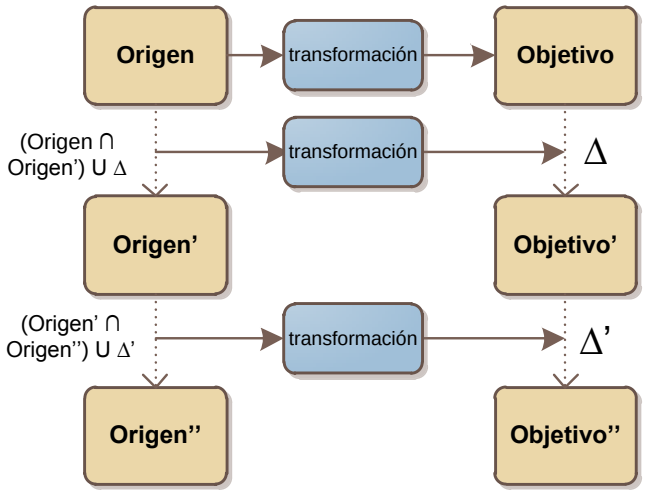


Figura 13.1: Transformación directa

La Fig. 13.1 muestra la base de la propuesta realizada en este trabajo [García-Díaz et al., 2011c] (véase sección 13.4). La idea básica es poder reutilizar las acciones que hacen los VCSs antes de almacenar los modelos en el repositorio. De esa forma,

el trabajo realizado se puede reutilizar en la fase de la generación de artefactos.

Característica	**Re-transformación**	**Transformación en vivo**	**Transformación directa**
Se reduce el impacto de las transformaciones	NO	SI	SI
Se reducen los recursos computacionales necesarios	NO	SI	SI
Se habilita la realización de trazabilidad de los sistemas	NO	SI	SI
Se evita guardar el contexto de ejecución	SI	NO	SI
Se orienta a trabajar con artefactos textuales	SI	NO	SI
Se reutiliza el trabajo realizado por los VCSs	NO	NO	SI
Se evita una adaptación de los motores de transformación	SI	NO	SI
Se lleva a cabo un desarrollo incremental de forma simple	NO	NO	SI

Tabla 13.1: Comparación entre las aproximaciones incrementales

La Tabla 13.1 muestra una comparativa entre las 3 aproximaciones:

- Con la transformación en vivo y la transformación directa se reduce el impacto de las transformaciones realizadas sobre los sistemas ya desplegados, ya que realmente sólo se generan los artefactos necesarios para cada caso. De ese modo, sólo una parte de las aplicaciones se ve afectada ante un cambio en el modelo de entrada.

- Con la transformación en vivo y la transformación directa se aumenta la eficiencia en los desarrollos debido a que no hay que generar todos los artefactos en todos los casos. De otro modo, habría que generar todos los artefactos incluso ante cambios mínimos en el modelo de entrada.

- Con la transformación en vivo y la transformación directa se puede llevar a cabo con cierta facilidad la trazabilidad de los sistemas. Es decir, se puede saber qué cambios en los modelos hicieron o harán que se modifiquen determinados artefactos en la salida. Esta acción puede servir, por ejemplo, para facilitar la depuración de los desarrollos de software.

- Una desventaja de la transformación en vivo es que hay que guardar el contexto

de ejecución. Es decir, hay que tener guardado en memoria el estado de las transformaciones, lo cual puede ser complejo y propenso a errores.

- Básicamente existen dos tipos de artefactos, los modelos y los textuales. Los modelos tienen una estructura interna en forma de grafo, lo cual provoca que la mayoría de los lenguajes existentes para transformar un modelo en otro estén basados en reglas. La generación de artefactos textuales es más directa y puede realizarse con éxito utilizando plantillas, mucho más sencillas de entender y utilizar que los lenguajes basados en reglas.

- Una gran ventaja de la transformación directa es que se basa en la comparación de los modelos. Cuando se trabaja con integración continua, los VCSs ya han realizado dicha comparación antes de llegar a la fase de generación de artefactos. De ese modo, el generador se aprovecha de las acciones ya realizadas por otras herramientas, haciendo que el proceso de generación incremental sea muy directo.

- Las propuestas para realizar transformación en vivo requieren una adaptación de los motores de transformaciones actuales, lo cual no es un proceso trivial y suele conducir a alternativas *ad-hoc*.

- Con la transformación directa se pueden generar artefactos de forma incremental de manera muy sencilla. El autor del lenguaje no tiene por qué preocuparse de aprender a programar con complicados lenguajes ni a profundizar en la semántica de los mismos.

13.3. Comparación de modelos

Desde un punto de vista práctico, generar artefactos incrementalmente requiere saber los cambios que se han producido en un modelo respecto a otro, es decir, es necesario conocer las diferencias entre los estados por los que pasa un modelo. Este aspecto es muy importante tanto para la generación incremental como para el buen funcionamiento de los sistemas de control de versiones para modelos (véase sección 8.11).

Como se ha comentado, conocer las diferencias entre modelos no es un proceso sencillo [Read and Cornell, 1977] y comprende tres fases [Brun and Pierantonio, 2008] de las que, para generar artefactos de manera incremental, resultan útiles las dos primeras, es decir, el cálculo de las diferencias entre modelos y la representación de dichas diferencias. La visualización de las diferencias en un formato amigable

de cara al usuario de la herramienta no es un aspecto importante, puesto que la generación se realiza sin intervención humana.

13.3.1. Cálculo de las diferencias entre modelos

Este es el paso fundamental ya que de él dependen los demás. Sin embargo, calcular las diferencias entre modelos es una tarea muy complicada que no tiene una única solución óptima porque depende de cada problema particular [Kolovos et al., 2009]. Ello provoca que haya numerosos estudios sobre el tema y se haya creado una herramienta como MCTest (véase sección 9.3), que permite ayudar a buscar el mejor algoritmo en función de cada lenguaje y del dominio de aplicación. El cálculo de las diferencias entre modelos ya ha sido tratado en este documento (véase sección 8.11.2).

13.3.2. Representación de las diferencias entre modelos

Un aspecto fundamental para generar artefactos de manera incremental es representar la diferencia entre la versión original de un modelo y otra nueva versión del modelo realizada posteriormente. De ese modo, se podrá saber qué partes del modelo han de ser tenidas en cuenta para generar artefactos, ya que en la mayoría de las ocasiones no será necesario regenerar todos los artefactos, puesto que habrá información en el nuevo modelo que regeneraría exactamente los mismos artefactos que el modelo inicial, con la consiguiente pérdida de tiempo y recursos.

Para representar las diferencias entre modelos, han surgido diferentes iniciativas durante los últimos años. Además, en [Cicchetti et al., 2007c] se enumeran una serie de características que, según los autores, ha de tener cualquier técnica de representación de diferencias entre modelos. Entre ellas destacan *la independencia del metamodelo y el hecho de estar basada en modelos.*

Así, la representación no debería estar basada en un metamodelo específico, permitiendo el empleo metamodelos adaptados a las necesidades de cada problema. De hecho, una solución comúnmente aceptada en MDE es el empleo de una arquitectura de cuatro capas en la que la capa con mayor nivel de abstracción es un único meta-metamodelo (p.e., el espacio de modelado MOF [Djurić, 2006] utilizado en la especificación de MDA [Miller et al., 2003]), a partir del cual se pueden definir diferentes metamodelos. Además, autores como David Frankel dicen que para el éxito de MDE es crucial el uso de un metamodelo raíz y común a todos los demás [Frankel, 2003].

En cuanto a basarse en modelos, un principio también comúnmente aceptado

es el de *Everything is a model* de Bézivin [Bézivin, 2005], que junto con el empleo de metamodelos y meta-metamodelos, permiten trabajar utilizando un mayor nivel de abstracción, con las numerosísimas herramientas que existen bajo el paradigma MDE.

Por otra parte, otras características como la compactibilidad, la transformatividad y la componibilidad pueden ser interesantes y ofrecen posibilidades, pero no son clave para la generación incremental de artefactos.

Scripts de editado

Una de las técnicas más utilizadas para representar las diferencias entre modelos son los *scripts* de editado, utilizados en trabajos como [Alanen and Porres, 2003], que pertenece a la categoría de *deltas dirigidas*[3].

Consiste en crear secuencias de operaciones primitivas que describen las modificaciones que ha sufrido un modelo respecto a otro y se basa en una secuencia de transformaciones que permiten representar las diferencias como una delta (Δ).

Para describir cómo es la Δ, utilizan 7 transformaciones elementales que han de cumplir con una serie de restricciones que limitan el empleo de esta técnica. Estas transformaciones, que describen los cambios de un modelo respecto a otro, pueden ser: *new, del, set, insert, remove, insertAt* o *removeAt*. Así:

M_{new} - $M_{old} = \Delta$

$M_{old} + \Delta = M_{new}$

Basadas en colores

Otra técnica muy utilizada es la basada en colores, que pertenece a la categoría de *deltas simétricas*. Esta técnica, tratada en trabajos como [Ohst et al., 2003], consiste en mostrar gráficamente las diferencias mediante el empleo de un diagrama. Este diagrama contiene las partes comunes del modelo inicial y del modelo final. Con marcas como colores o símbolos, se representan las partes que han cambiado. Otro ejemplo es el EMF Compare [Toulmé, 2007], que representa y muestra las diferencias entre elementos de forma gráfica.

Modelo de la diferencia

Como las técnicas propuestas hasta la fecha no cumplían con los requisitos esperados, en Cicchetti et al. [Cicchetti et al., 2007c, Cicchetti et al., 2007b] se propone

[3]La representación de diferencias se puede dividir en dos grandes categorías: deltas dirigidas y deltas simétricas. Las primeras se basan en una secuencia de operaciones y las segundas en mostrar las diferencias como un conjunto

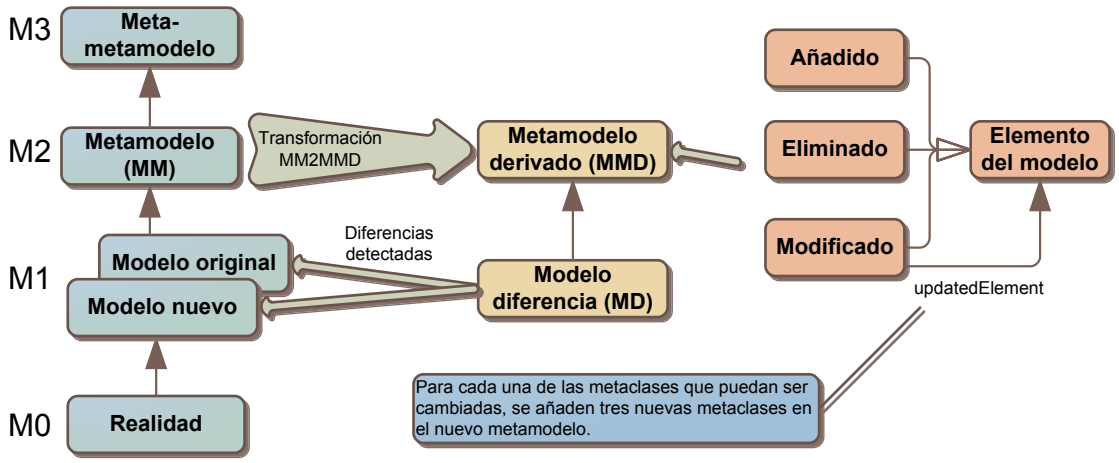


Figura 13.2: Representación de diferencias entre modelos

una nueva técnica. La idea es que dados dos modelos que conforman a un mismo metamodelo, la diferencia entre ambos generará otro modelo que conforma a otro metamodelo derivado del primero de ellos. Así, los modelos surgidos de las diferencias serán artefactos de primera clase en el proceso de generación. Entonces, se captarán las diferencias que hay en el segundo modelo respecto al primero, ya que se pudieron *borrar*, *añadir* o *modificar* elementos. La Tabla 13.2 muestra una comparativa de las técnicas expuestas. Gracias al empleo del modelo de la diferencia se consigue que la propia diferencia entre los modelos sea un modelo que conforma a un metamodelo, lo cual permite trabajar con cualquier herramienta o tecnología MDE.

Propiedad	***Scripts* de editado**	**Basadas en colores**	**Modelo de la diferencia**
Basada en modelos	NO	SI	SI
Independiente del metamodelo	NO	SI	SI
Compacta	NO	NO	SI
Transformatica	NO	NO	SI
Componible	SI	NO	SI

Tabla 13.2: *Scripts* de editado, basadas en colores y modelo de la diferencia

En la Fig. 13.2 se puede observar cómo a partir de la diferencia entre dos modelos se crea un nuevo modelo que conforma con otro metamodelo. El nuevo metamodelo estará derivado del anterior metamodelo porque a todas las metaclases que pueden ser cambiadas se les añade otras tres nuevas metaclases, una para cada posible tipo de cambio (borrar, añadir o modificar). En la Fig. 13.3 se muestra un ejemplo obtenido

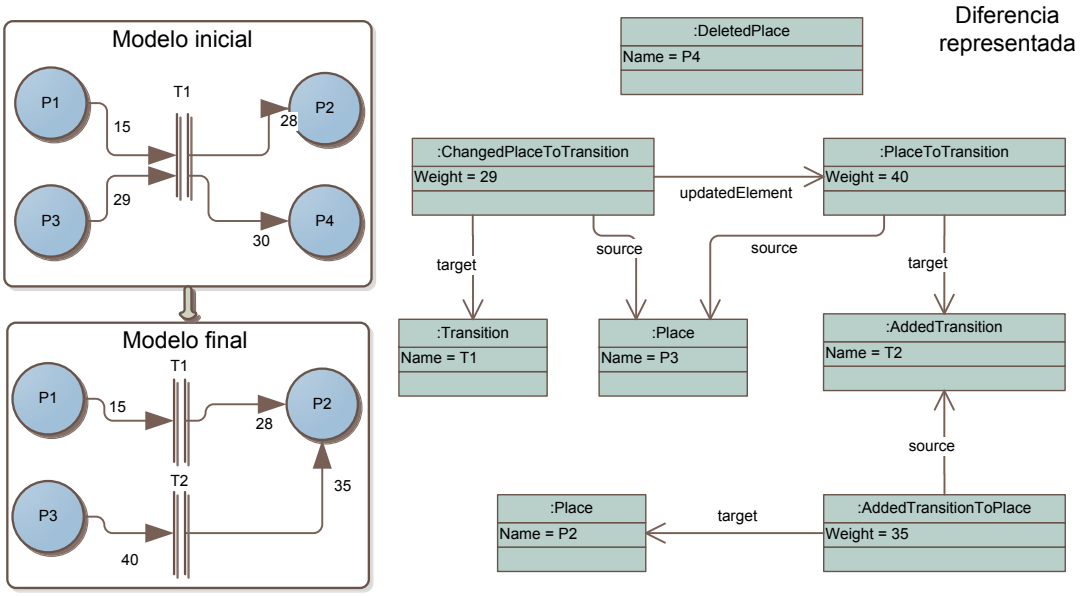


Figura 13.3: Representación de diferencias entre modelos (ejemplo)

a partir del trabajo expuesto en [Cicchetti et al., 2007a], en el que se muestra la diferencia entre dos *redes de Petri* [Peterson, 1981].

13.4. Propuesta basada en el cálculo y representación de la diferencia

Los trabajos actuales relacionados con la obtención de la diferencia entre versiones de un mismo modelo, se centran en proporcionar avances en ámbitos de la ingeniería dirigida por modelos diferentes a la generación de artefactos. Sin embargo, estudiando las analogías, se observa que el procesamiento de las diferencias entre modelos también es interesante de cara a realizar generaciones incrementales. Así, la generación incremental de artefactos se puede tratar con técnicas análogas a las utilizadas en los trabajos citados previamente. De hecho, al representar la diferencia entre un modelo original y una nueva versión realizada posteriormente, se podría llegar a averiguar qué partes del modelo han de ser tenidas en cuenta para generar artefactos, ya que en la mayoría de las ocasiones no será necesario regenerar todos los artefactos.

El algoritmo de cálculo de las diferencias entre modelos tiene una gran importancia en este trabajo puesto que es un paso necesario y básico para las demás operaciones. En ese sentido se ha utilizado la herramienta EMF Compare [Toulmé, 2007], la más destacada para realizar comparaciones sin restringirse a un meta-

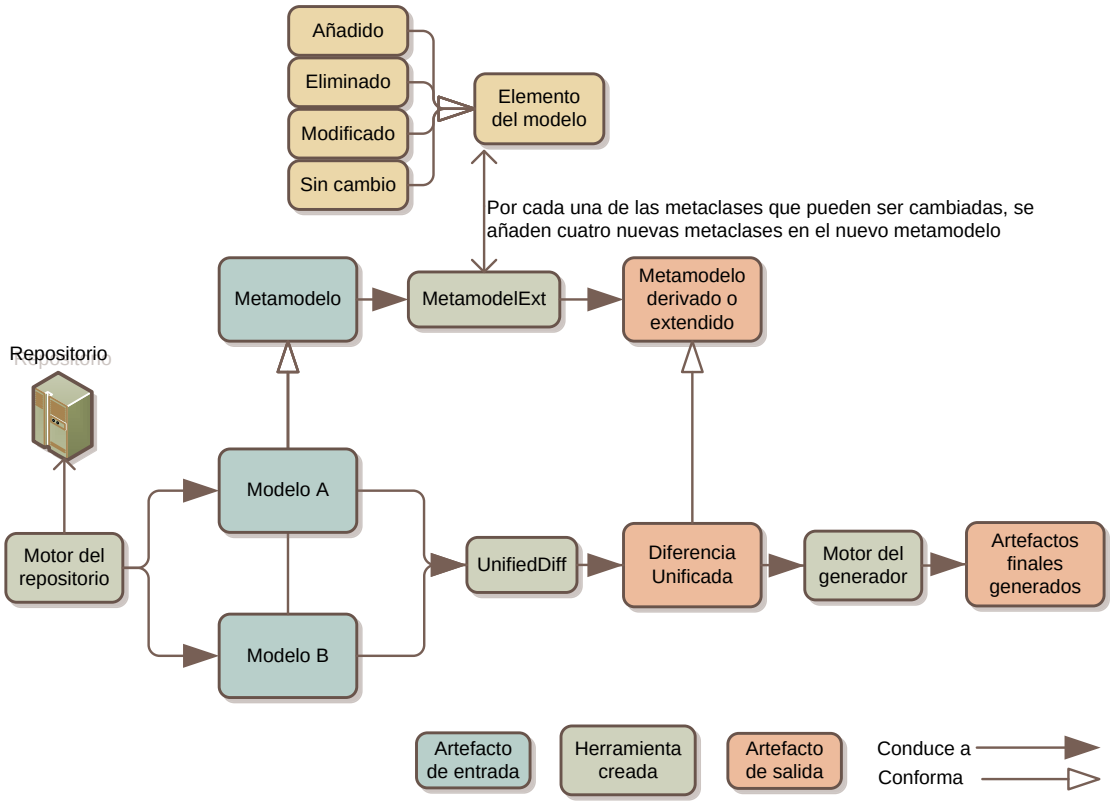


Figura 13.4: Arquitectura propuesta

modelo específico (véase sección 8.11.2), lo que permitirá ser aplicada a diferentes tipos de aplicaciones y dominios de conocimiento. Además, como la elección de cada algoritmo concreto (EMF Compare incorpora un algoritmo genérico) depende fundamentalmente de la semántica subyacente de los elementos de cada lenguaje de dominio específico [Kolovos et al., 2009], se ha creado la herramienta MCTest con el objetivo de mejorar la calidad de dichos algoritmos (véase sección 9.3).

Por otra parte, para no perder la esencia y las ventajas del uso de de una propuesta basada en modelos, se ha propuesto una arquitectura basada parcialmente en la representación de las diferencias entre modelos [Cicchetti et al., 2007c] (Fig. 13.4).

La idea, cuando se pretende generar artefactos, es utilizar siempre las dos últimas versiones del modelo del sistema, que se hayan introducido previamente en algún tipo de repositorio como consecuencia de las acciones de los desarrolladores. Dicho repositorio, por su parte, podría estar gobernado por un MVCS y hacer uso de las técnicas para conocer las diferencias entre modelos expuestas previamente. Un paso fundamental es extender el metamodelo original, creando uno con las nuevas

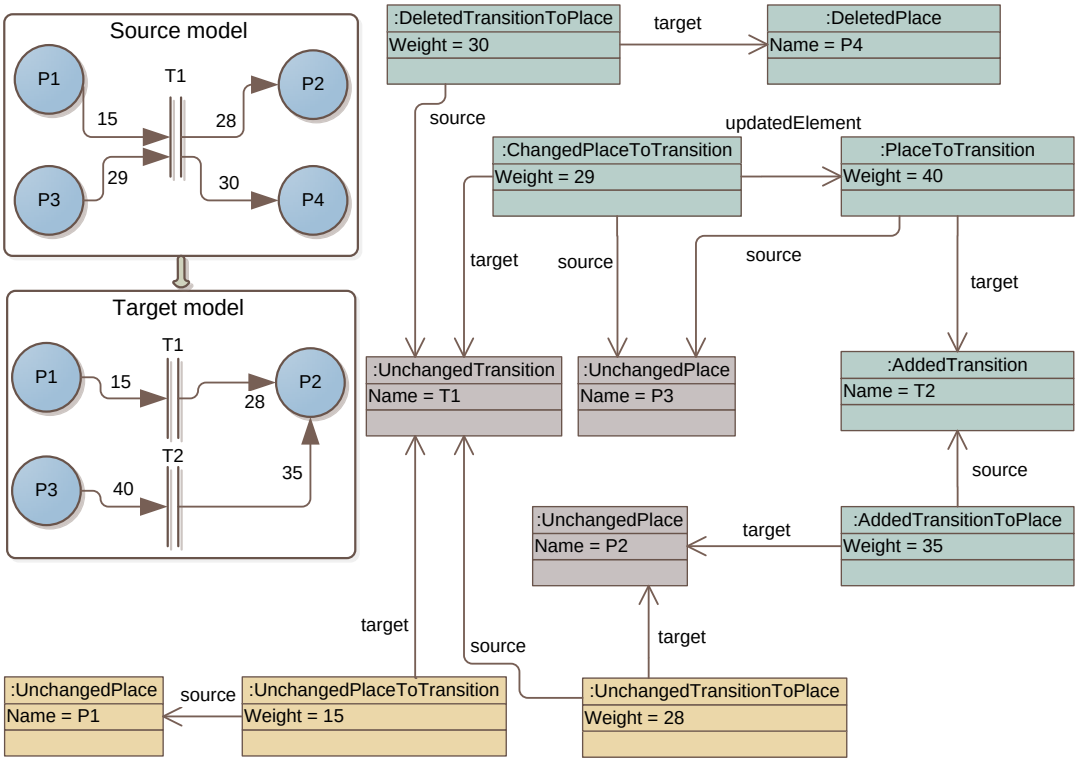


Figura 13.5: Representación de la diferencia unificada (ejemplo)

metaclases. Sin embargo, el modelo de la diferencia sufre una variación con respecto a la propuesta original debido a que si una modificación en un elemento provoca que se tengan que regenerar artefactos derivados de otro elemento que no ha sido alterado, dicho elemento no estará en el modelo de la diferencia y provocará inconsistencias.

La solución planteada es, por tanto, unificar el modelo de la diferencia con las partes de la última versión del modelo que no han sido modificadas, dando lugar a la *diferencia unificada*. Para facilitar el empleo de las tecnologías actuales de generación de artefactos, se puede afirmar que se trabaja en todo momento con una diferencia independiente del modelo [Konemann, 2009], la cual es auto-contenida, es decir, la información se representan sin necesidad de hacer uso del modelo original.

Pese a que con el modelo de la diferencia se podrían generar artefactos en muchos casos, ya que se basa en metamodelos y modelos, con la diferencia unificada se podrán tratar todos las posibilidades debido a que se trabaja con todas las alternativas posibles.

En la Fig. 13.5 se muestra un ejemplo basado en la Fig. 13.3, en el que se puede observar la diferencia entre dos modelos de redes de Petri [Peterson, 1981]

que conforman a un mismo metamodelo. En la parte superior se muestra la diferencia del modelo planteada en el trabajo original. En la parte inferior están los elementos añadidos para que dicha diferencia pueda ser considerada una diferencia unificada. Así, se logra un nuevo modelo, instancia de un metamodelo derivado del inicial, en el que se contiene toda la información necesaria para modificar, añadir, eliminar o mantener cualquiera de los artefactos implicados en el sistema. En este ejemplo, las diferencias ya afectan a casi todo el modelo por lo que habría que añadir pocos elementos. Sin embargo, en casos reales con modelos mucho mayores es muy probable que las diferencias afecten a un fragmento muy pequeño del modelo, haciendo esta técnica aún más útil.

13.5. Arquitectura utilizada

En lugar de desarrollar desde cero una herramienta dirigida por modelos de generación incremental de artefactos, se ha decidido utilizar y adaptar las herramientas existentes, en concreto las provistas en el Eclipse Modeling Project[4], que anteriormente daban lugar a oAW. Las razones principales ya se han dado en el capítulo 11 y son, sobre todo, el cumplimiento de estándares y con la *Eclipse Public Licence*[5].

En la Fig. 13.6 se representan los cuatro pasos fundamentales para generar artefactos de manera incremental.

13.5.1. Crear el metamodelo Ecore

Dado que algunas iniciativas MDE como MDA o TMDE [García-Díaz et al., 2010a] (véase capítulo 7) abogan por la generación de artefactos a partir de un meta-metamodelo basado en estándares (es decir, MOF), y la implementación de MOF más comúnmente aceptada es Ecore, será necesario diseñar un metamodelo Ecore como primer paso para la generación incremental.

Este metamodelo dependerá del DSL que se utilice para cada caso concreto de aplicación y puede ser construido directamente mediante UML o mediante una transformación de un DSL de mayor nivel de abstracción (p.e., DSL Tools [Cook et al., 2007]). Para lograr dicha transformación, se podrían utilizar algunas propuestas como las de Bézivin et al. [Bézivin et al., 2005b, Bézivin et al., 2005a].

Una vez que el metamodelo esté creado, ya se puede utilizar cualquier herramienta que soporte Ecore para crear y manipular modelos que conformen con el metamodelo definido.

[4]http://www.eclipse.org/modeling/
[5]http://www.eclipse.org/legal/epl-v10.html

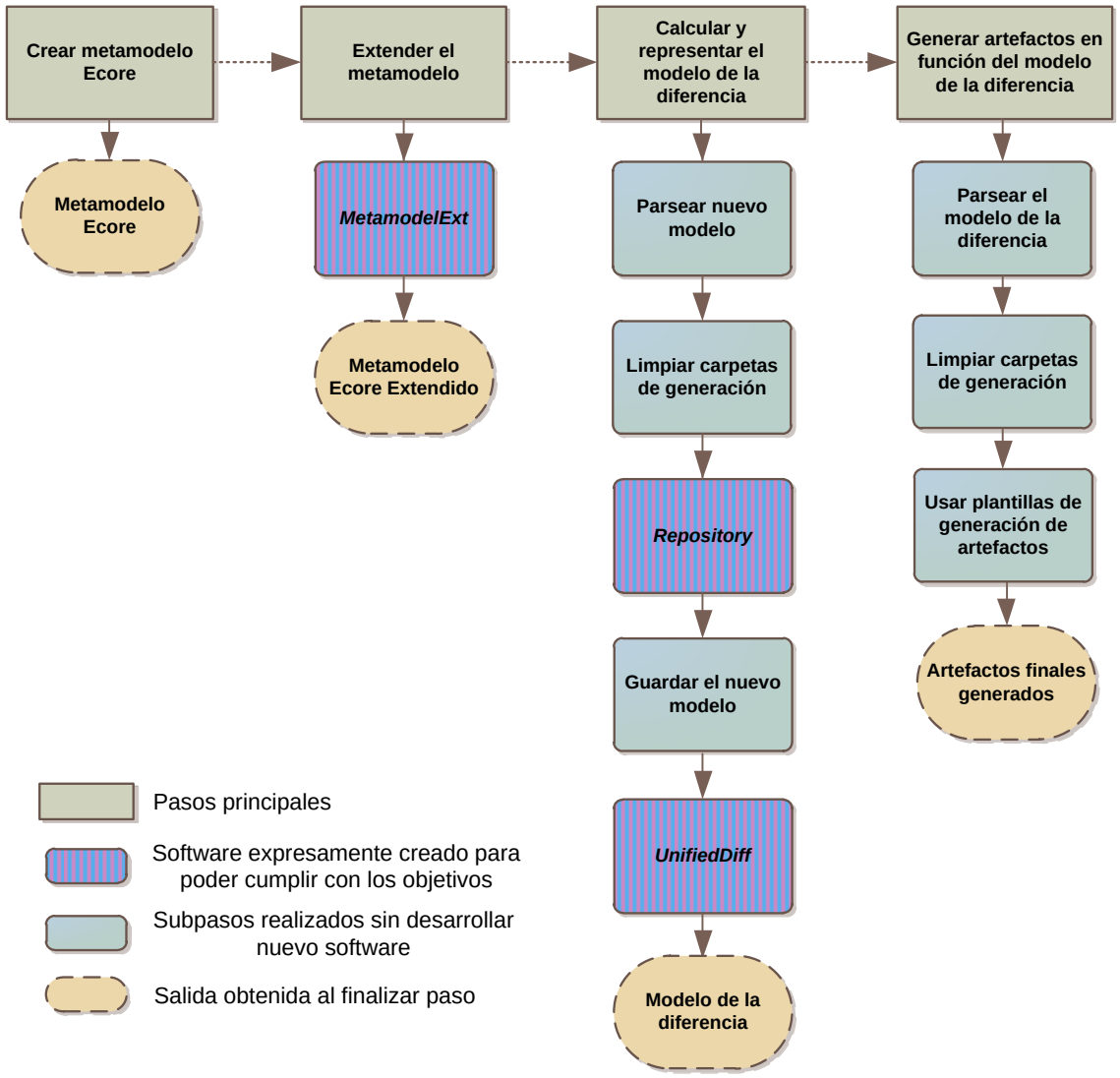


Figura 13.6: Pasos para desarrollar la generación de artefactos incremental

13.5.2. Extender el metamodelo

La extensión del metamodelo original se realiza en base al metamodelo Ecore. Esta extensión, se hace cuando se desarrolla el metamodelo original, no siendo parte del proceso de generación de artefactos y siendo necesaria su realización una única vez para cada familia de productos [Clements and Northrop, 2002]. Para ello, a partir de cada metamodelo Ecore, se crea su metamodelo extendido, en el que para cada metaclase original se añaden 4 nuevas metaclases (añadido, borrado, modificado y no modificado), que heredan de la original. Así, se permite representar la diferencia de dos modelos en base al metamodelo extendido.

El cálculo del metamodelo extendido se realiza utilizando una herramienta que

ENTRADA PARA MODEL WORKFLOW ENGINE

```
<workflow>
  <property name='modelFile' />
  <property name='targetDir' value='src-gen/'/>
  <bean class="org.eclipse.mwe.emf.StandaloneSetup">
     <platformUri value=".."/>
  </bean>

  <component class="org.eclipse.mwe.emf.Reader">
    <modelSlot value="ecoreModel" />
    <uri value="${metamodelDir}" />
  </component>

  <component class="MWE.Extends.MetamodelExt">
    <packageSlot value="ecoreModel" />
    <outputSlot value="diff" />
  </component>

  <component class="org.openarchitectureware.emf.XmiWriter">
     <inputSlot value="diff" />
     <modelFile value="${targetNewMetamodelFile}" />
  </component>
</workflow>
```

Se lee el metamodelo original

Se calcula el metamodelo extendido

Se guarda el metamodelo extendido

Figura 13.7: Extensión el metamodelo original (ejemplo)

se ha denominado *MetamodelExt*. Forma parte de la extensión para MWE llamada *MWE.Extends*, que también se ha desarrollado explícitamente para este trabajo, la cual ofrece el soporte necesario para generar artefactos incrementalmente.

En la Fig. 13.7 puede verse el archivo MWE (Model Workflow Engine)[6], que se utiliza para guiar el proceso de extensión del metamodelo original.

13.5.3. Calcular y representar el modelo de la diferencia

Este paso, a diferencia de los anteriores, se lleva a cabo siempre que se realiza la generación de artefactos. Esto es debido a que el objetivo es utilizar las diferencias entre una versión de un modelo respecto a la última a partir del cual se había realizado una generación. De esa forma, se podrá ir gestionando la evolución de los modelos y, al mismo tiempo, la evolución de los artefactos que se generan a partir de los mismos.

Para calcular la diferencia unificada, MWE.Extends ofrece el siguiente soporte:

- *Repository*. Se encarga de simular el comportamiento de un repositorio. Permite realizar pruebas de forma fluida y podría ser sustituido en cualquier

[6]MWE es una herramienta del Eclipse Modeling Project, que originalmente era parte de oAW. Tiene un objetivo similar a otras herramientas como Ant o MSBuild pero está especialmente orientada a trabajar con modelos

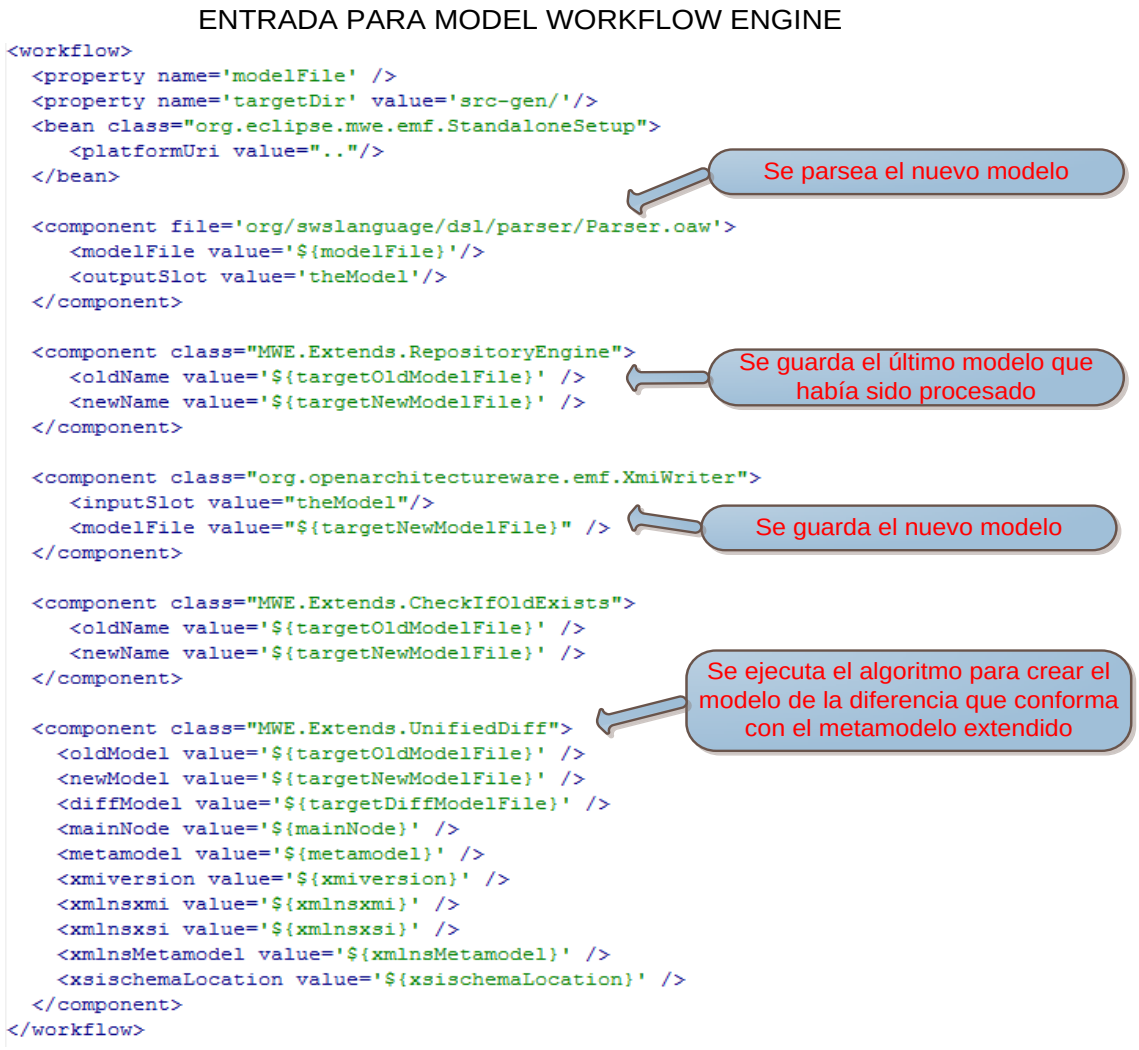


Figura 13.8: Cálculo y representación de la diferencia entre modelos (ejemplo)

momento por un sistema de control de versiones para modelos que ofrezca mayores características.

- *UnifiedDiff*. Permite calcular la diferencia entre dos versiones de un modelo, que conforman a un mismo metamodelo, generando así otro modelo que conforma a otro metamodelo derivado del primero de ellos (el metamodelo extendido se crea con la herramienta *MetamodelExt*). Además, incluye todos los elementos que permanecen intactos en la última versión del modelo para permitir generar artefactos en función de ellos si así fuera necesario. Los modelos surgidos de las diferencias serán artefactos de primera clase en el proceso de generación, permitiendo captar en un único archivo a todos los elementos que se

han modificado, borrado, añadido o que permanecen intactos. Así, se consigue tener toda la información necesaria unificada en un único modelo, favoreciendo el empleo de las tecnologías actuales para trabajar con herramientas tales como repositorios, algoritmos de comparación, plantillas de generación, etc.

En la Fig. 13.8 puede verse el archivo MWE que se utiliza para guiar la obtención de la diferencia entre dos modelos.

13.5.4. Generar artefactos en función del modelo de la diferencia

ENTRADA PARA MODEL WORKFLOW ENGINE

```
<workflow>
  <bean class="org.eclipse.mwe.emf.StandaloneSetup">
     <platformUri value=".."/>
  </bean>

  <component id="xmiParser" class="org.eclipse.mwe.emf.Reader">
     <uri value="${modelFile}"/>
     <modelSlot value="theModel"/>
     <firstElementOnly value="true"/>
  </component>

  <component class='oaw.workflow.common.DirectoryCleaner' directory='${targetDir}'/>

  <component class='oaw.xpand2.Generator'>
     <metaModel id='mm' class='org.eclipse.m2t.type.emf.EmfRegistryMetaModel'/>
     <expand value='org::swslanguage::dsl::MainExt::main FOR theModel'/>
     <genPath value='${targetDir}'/>
  </component>
</workflow>
```

Se parsea el modelo de la diferencia

Se prepara la carpeta destino

Se generan los artefactos incrementalmente

Figura 13.9: Generación incremental de artefactos (ejemplo)

Una vez se tiene la diferencia unificada, el último paso es generar los artefactos en base al metamodelo extendido. Para ello, y aunque existen otras alternativas, se han utilizado plantillas Xpand [Efftinge and Kadura, 2006], que forman parte del EMP y tienen soporte para el empleo de interesantes características como polimorfismo. Así, por ejemplo, se permite que una metaclase y la metaclase *añadida* que hereda de ella generen los mismos artefactos sin escribir ni una sólo línea de código extra en la plantilla.

En la Fig. 13.9 puede verse el archivo MWE que se utiliza para guiar la generación incremental de artefactos.

13.6. Caso de estudio

Se ha creado un caso de uso con el objetivo de probar y mostrar los resultados que proporciona la generación incremental de artefactos. Se basa en una aplicación real denominada Tweeterati[7], que es un cliente de Twitter[8] (véase sección 16.2.3).

Durante la evaluación se han hecho, en orden secuencial, 15 cambios en el modelo de entrada utilizado por el generador de artefactos. La Fig. 13.10 muestra un fragmento de código que representa una transición entre dos versiones de un modelo y el aspecto del lenguaje de plantillas utilizado para generar artefactos según los cambios detectados.

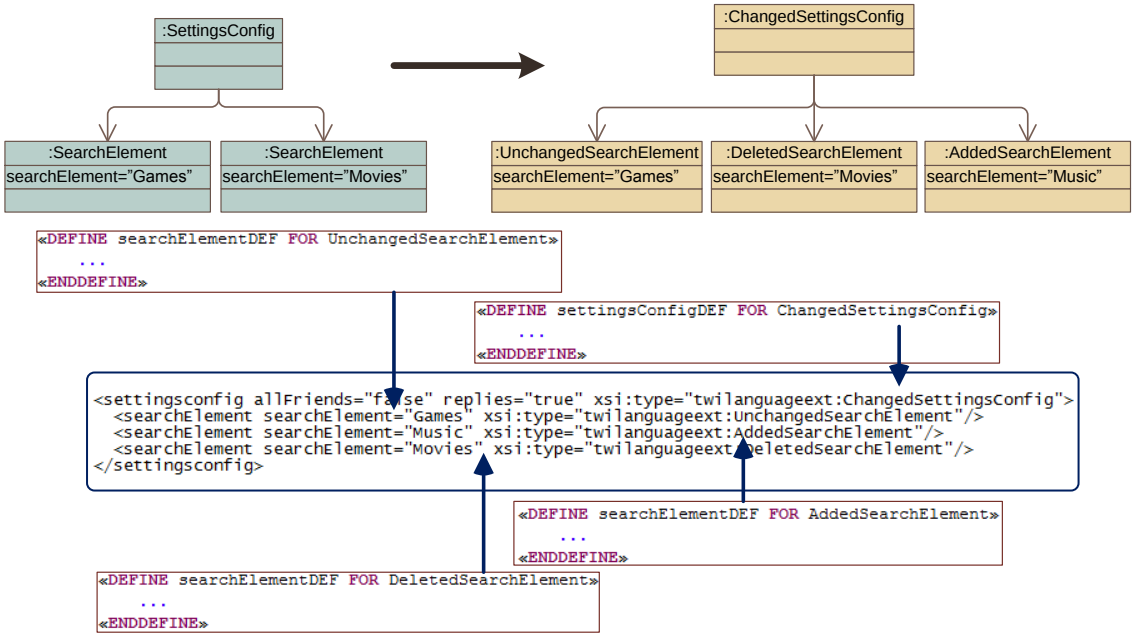


Figura 13.10: Evolución de un modelo y generación con plantillas (fragmento)

Las generaciones que han ido sucediéndose a lo largo de los cambios del modelo de entrada han dado una serie de resultados, sugiriendo las ventajas, que extrapolado a otros dominios, ofrece la generación incremental de artefactos (Tabla 13.3)[9].

[7] http://tweeterati.codeplex.com/
[8] https://twitter.com/
[9] El detalle de este trabajo se encuentra en la sección 16.2.3

Parámetro	Generación tradicional	Generación incremental
Requiere detener la aplicación?	Si	Si afecta a los ensamblados
Es posible realizar trazabilidad?	No	Si
Número artefactos	1620	56
Tamaño artefactos (KB)	19,956	0,091
Tiempo de generación (s)	11,940	8,821
Tiempo de despliegue (min)	2,905	1,313

Tabla 13.3: Resultados obtenidos con generación incremental

13.7. Conclusiones

Si no se emplea la generación incremental de artefactos, habrá que regenerar todos los artefactos cada vez que exista un cambio en los modelos. De esa forma, si se quiere cambiar algún aspecto de una aplicación que ya está en funcionamiento, habrá que detenerla, generar los artefactos y volver a desplegarla.

Utilizar la generación incremental aumenta la probabilidad de que los artefactos que se generan no afecten a otros, haciendo posible que se modifiquen aspectos de una aplicación sin pararla.

Un ejemplo podría ser un modelo que genere sentencias INSERT en una base de datos de una aplicación. Con esta propuesta se podrían generar sentencias UPDATE cuando se detecta un cambio en el modelo, actualizando la base de datos y permitiendo continuar la ejecución de la aplicación. Si no se pudiera controlar el momento en el que el modelo ha cambiado y generar sentencias en consecuencia, habría que parar la aplicación, borrar las sentencias anteriores y realizar los nuevos INSERT (a diferencia de los UPDATE).

Un aspecto interesante es la trazabilidad. Suponiendo que M, M' y M" son las diferentes versiones del modelo de la Fig. 12.1. $\triangle$, $\triangle$' y $\triangle$" son las diferencias unificadas de una versión respecto a la versión anterior, siendo $\varnothing$ el conjunto vacío y g($\triangle$), g($\triangle$') y g($\triangle$") la generación de artefactos (A, A' y A") producidas a partir de una diferencia unificada respecto a la versión anterior. Entonces:

M - $\varnothing$ = $\triangle$ = M $\rightarrow$ g($\triangle$) = A

M' - M = $\triangle$' $\rightarrow$ g($\triangle$') = A'

M" - M' = $\triangle$" $\rightarrow$ g($\triangle$") = A"

Así, es posible saber las diferentes versiones por las que ha pasado el sistema y exactamente qué artefactos han sido creados, eliminados o modificados con cada una de las mismas. Sin generación incremental de artefactos, siempre se generarían

todos los artefactos y, por lo tanto, la trazabilidad no podría ser realizada.

Respecto al número y el tamaño de los artefactos que han de volver a regenerarse directa o indirectamente, se puede afirmar que con la generación incremental de artefactos se reduce potencialmente el número de artefactos generados, y al reducirse el número de los artefactos generados, el tamaño de los mismos también lo hace.

El tiempo de generación es el menos importante de los parámetros estudiados porque no afecta directamente a las aplicaciones en desarrollo o ya desplegadas. Además, está abierto a la incorporación de diferentes algoritmos y técnicas que mejoren la obtención y la generación de los artefactos. Por otra parte, como tanto el número de artefactos como su tamaño potencialmente se reduce, es de esperar que también se reduzca el tiempo de despliegue de las aplicaciones.

En los resultados mostrados en esta sección, los mayores beneficios se obtienen respecto al número de artefactos que de alguna manera hay que generar y al tamaño de éstos. El tiempo de generación también se ha visto reducido aunque no en un factor tan importante, debido a la necesidad extra de cálculo cuando se trabaja con el metamodelo extendido y la diferencia unificada. En este caso concreto de aplicación, la ganancia en cuanto al tiempo de despliegue, aunque considerable, no ha sido muy alta debido a la gran cantidad de dependencias que había entre los archivos de la solución software en cuanto a las unidades de compilación y de subida al servidor en el que se ha subido la aplicación durante la etapa de despliegue. En aplicaciones con menores dependencias la ganancia sería mucho mayor.

En resumen, se ha adaptado una herramienta dirigida por modelos que permite realizar generación incremental de artefactos a partir de modelos basados en estándares. Algunos de los beneficios obtenidos son:

- Se evita que cada cambio en un modelo provoque la regeneración completa de todos los artefactos derivados de dicho modelo. Así, se generarán únicamente los artefactos estrictamente necesarios.

- Cuantos menos artefactos se regeneren, menor será el impacto provocado sobre el sistema software final, que incluso podría estar ya desplegado en una máquina de un cliente.

- Al contemplar la modificación, eliminación y la inserción de elementos, se consigue que se pueda gestionar la evolución de los sistemas software ya desplegados.

Bloque VII

Integración Continua Dirigida por Modelos

Índice del bloque

CAPÍTULO 14

Construcción de un prototipo MDCI

A lo largo de los anteriores capítulos se han presentado diversos problemas y las soluciones propuestas. De ese modo, ya se está en disposición de construir un prototipo, e implícitamente un proceso, para llevar a cabo MDCI (Model-Driven Continuous Integration). Así, se ha seleccionado una iniciativa MDE, se han adaptado los sistemas de control de versiones a los modelos software, se han adaptado los generadores de artefactos dirigidos por modelos a las herramientas de CI y se ha logrado una generación incremental de artefactos en dicho ámbito.

En este capítulo se resumirán los problemas encontrados durante todo el trabajo y las soluciones propuestas. Además, se mostrará la estructura general y los principales componentes de la herramienta MDCIP (Model-Driven Continuous Integration Prototype).

⁎ ⁎ ⁎ ⁎

14.1. Soluciones parciales propuestas

Para realizar las actividades que conducen al objetivo de esta Tesis (véanse secciones 1.3 y 1.4), ha habido que solventar diversas dificultades a lo largo de los anteriores capítulos, sobre todo debido a la falta de herramientas adecuadas, tanto nivel comercial como académico. Así, en el anexo B se muestran las herramientas utilizadas, incluyendo otras que se han tenido que implementar específicamente para llevar a cabo el desarrollo de Model-Driven Continuous Integration Prototype.

A continuación, se resume el trabajo realizado [García-Díaz et al., 2011a]. Este trabajo está dividido en objetivos parciales, que han permitido tener una mejor visión de las actividades y de los beneficios obtenidos.

14.1.1. Elección de una iniciativa MDE

Para solucionar este problema, se ha realizado el trabajo mostrado en [García-Díaz et al., 2010a]. En dicho trabajo, se presenta la propuesta TALISMAN MDE (TMDE), cuya principal novedad es el empleo de una mezcla de principios para conseguir la máxima productividad con la máxima interoperabilidad, portabilidad y reusabilidad.

Es posible que la aproximación más razonable sea una mezcla de las principales propuestas que existen hasta la fecha, es decir, MDA y las Software Factories. Así, se puede generar una parte del sistema (la parte común a toda la familia de productos) utilizando las *mejores prácticas* para una plataforma específica y, al mismo tiempo, generar otras partes del sistema utilizando los estándares y las capas propuestas por MDA (véase capítulo 7).

14.1.2. Detección de nuevas versiones en los VCSs

Para solucionar este problema, ha sido necesario disponer de un MVCS. Por ese motivo, se ha desarrollado un prototipo que simula el comportamiento, en cuanto a la unidad de versión, de las propuestas realizadas por los autores más relevantes en dicho ámbito.

La idea es, basándose en los problemas citados en [Oliveira et al., 2005], permitir decidir, cada vez que se suben las fuentes al repositorio, la unidad de versión utilizada para lograr la máxima flexibilidad en la realización de pruebas. Los modelos se guardarán en un repositorio, en el que la herramienta de integración continua realizará comprobaciones cada cierto tiempo. Así, en el momento en el que se detecte una nueva versión del modelo, se lanzará el proceso de integración continua.

Por otra parte, se ha presentado la herramienta MCTest [García-Díaz et al., 2011b], que sirve para estudiar, probar y analizar algoritmos de comparación de modelos, muy útil para mejorar la precisión de los sistemas de control de versiones para los modelos de cada lenguaje de dominio específico concreto (véase capítulo 9).

14.1.3. Integración de MDE con herramientas de CI

Para solucionar este problema, se ha integrado la práctica de la integración continua y la aproximación de desarrollo de la ingeniería dirigida por modelos. Para realizar este trabajo han sido necesarias dos tipos de herramientas: una herramienta de CI y una herramienta de generación de artefactos MDE. Ambos tipos de herramientas existen y ofrecen mecanismos de extensión, por lo que no se han creado nuevas herramientas sustitutivas. Así, se han adaptado las herramientas existentes haciendo uso de sus puntos de extensión (véase capítulo 11).

14.1.4. Generación incremental de artefactos

Para solucionar este problema, se ha tenido que calcular, representar y procesar la diferencia entre dos versiones de un mismo modelo [García-Díaz et al., 2011c]. Se hace con el objetivo de poder generar artefactos incrementalmente en función de la evolución a la que se va sometiendo el modelo.

Para no perder la esencia de los modelos, se han seguido las características que ha de tener cualquier representación de diferencias entre modelos [Cicchetti et al., 2007c] y la técnica propuesta en [Cicchetti et al., 2007b]. La idea es que, dados dos modelos que conforman a un mismo metamodelo, la diferencia entre ambos generará otro modelo que conforma a otro metamodelo derivado del primero de ellos. En [Cicchetti et al., 2007b] se pretenden reflejar tres posibles tipos de diferencias entre un modelo respecto a una versión anterior de dicho modelo. Estas características son: *borrar, añadir* o *cambiar* un elemento. Sin embargo, como nada impide que haya dependencias entre los artefactos que podrían generarse a partir de diferentes elementos de un modelo, también habrá que conservar los elementos que *no han sido cambiados*. Esto es debido a que aunque no se hayan modificado, podrían ser necesarios a la hora de regenerar cualquier artefacto (véase capítulo 13).

14.2. Aspectos generales de MDCIP

La Fig. 14.1 muestra el diseño general del prototipo creado.

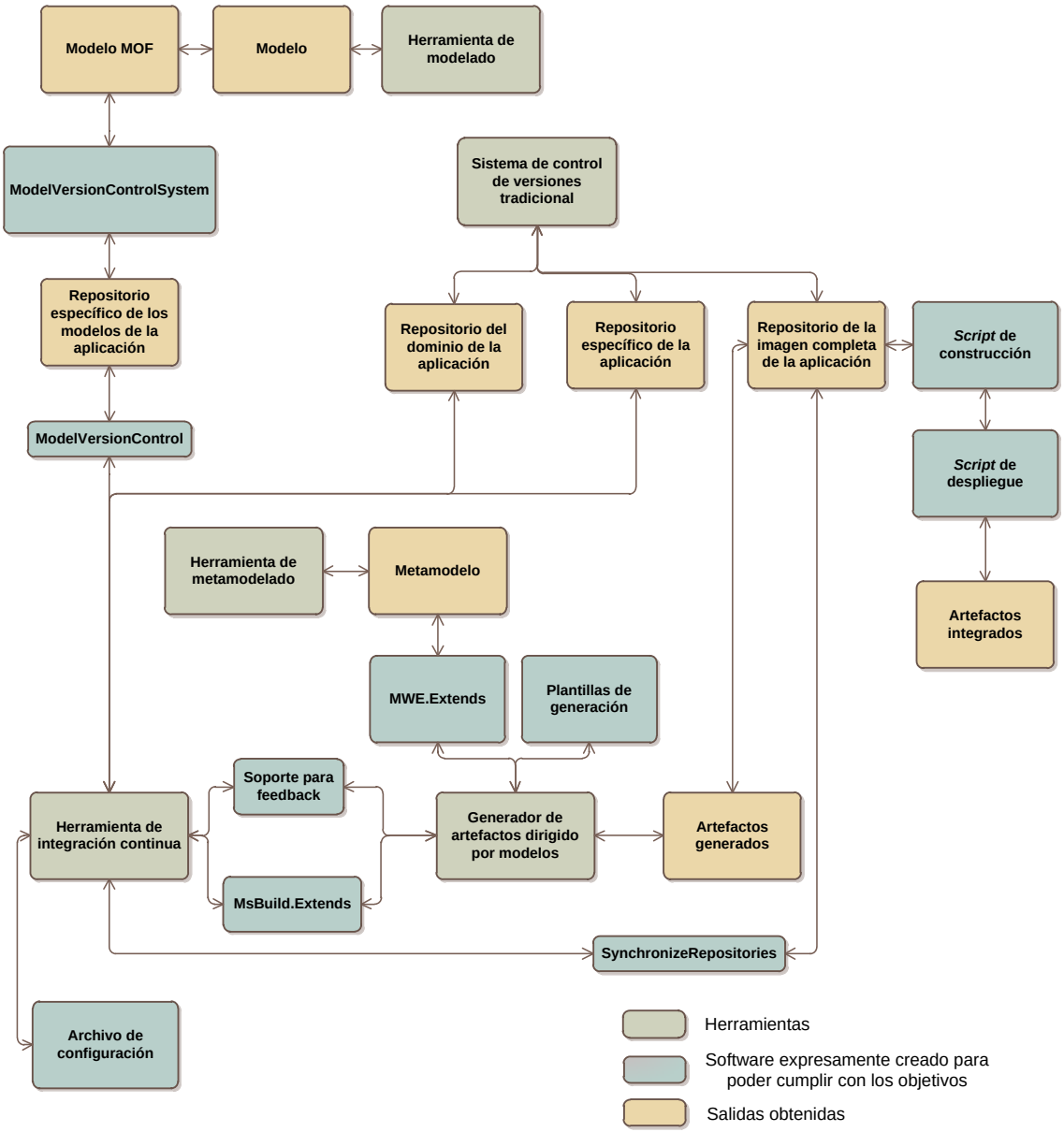


Figura 14.1: Prototipo MDCI (MDCIP)

1. Los usuarios de la herramienta de modelado, que en realidad están utilizando un DSL de alto nivel de abstracción, crean modelos. Dichos modelos, si no son ya inicialmente modelos MOF, son transformados para que lo sean, cumpliendo de esa forma con los estándares de la industria.

2. Los modelos de la parte variable de la línea de productos son almacenados en el reposito denominado *ModelVersionControlSystem*. De ese modo, la herramienta de integración continua, gracias a la extensión *ModelVersionControl*, puede detectar si se han producido cambios en el repositorio de los modelos.

3. La herramienta de integración continua ofrece soporte nativo para trabajar con los otros tres repositorios y detectar los cambios que en ellos se producen mediante el empleo de un sistema de control de versiones tradicional.

4. La extensión *SynchronizeRepositories* permite a la herramienta de integración continua sincronizar todos los cambios producidos en los repositorios de la aplicación, en el repositorio único de la imagen completa de la aplicación.

5. Para que la herramienta de integración continua funcione con todas las herramientas adyacentes, es necesario configurar un archivo con todas las especificaciones (p.e., el fragmento mostrado en la Fig. 14.4).

6. La herramienta de metamodelado permite crear el metamodelo, base del DSL utilizado por la herramienta de modelado.

7. Mediante el empleo de las herramientas incorporadas en *MWE.Extends* y el uso de plantillas de generación de artefactos adaptadas, se generan artefactos de forma automática. Para lograr una adecuada generación, la herramienta hace uso de las herramientas incorporadas en *MsBuild.Extends*.

8. La herramienta de integración continua ofrece retroalimentación del proceso de generación de artefactos mediante una extensión creada para tal fin.

9. Utilizando los *scripts* de construcción y de despliegue se logra que se generen los ensamblados, la documentación, los diferentes tipos de prueba y que se despliegue el software en la máquina correspondiente.

En la Fig. 14.2 se muestra el prototipo creado, con un punto de vista en el que se pueden observar las cuatro principales problemáticas tratadas durante todo el trabajo.

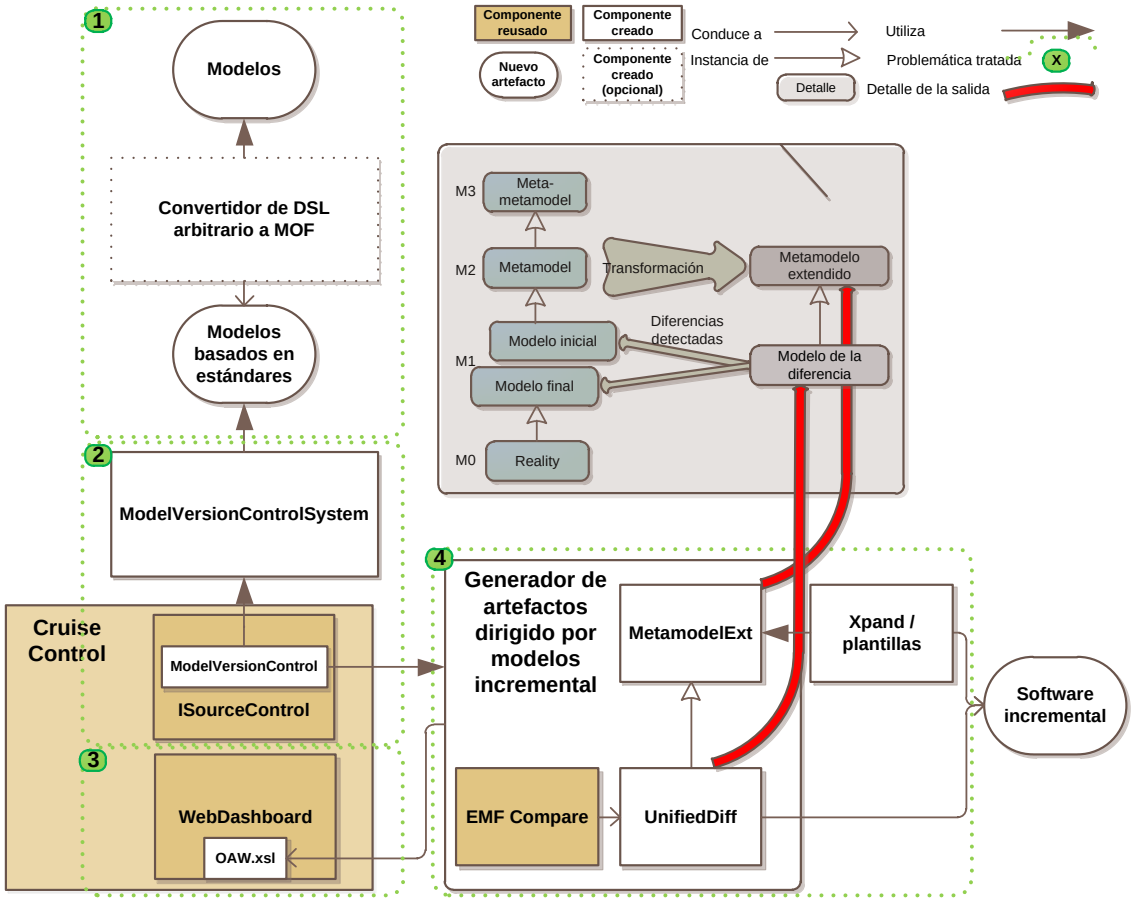


Figura 14.2: Prototipo MDCI (MDCIP) - otro punto de vista

14.3. Arquitectura básica de MDCIP

El prototipo desarrollado durante la realización de este trabajo es fiel reflejo de la estructura global del proceso MDCI (Model-Driven Continuous Integration) que se presentará en la Fig. 17.6. Además, en la Fig. 14.3 se puede observar un aspecto general del prototipo realizado y las tecnologías básicas utilizadas para su desarrollo. Para explicar el funcionamiento del prototipo[1], se hará a partir de su núcleo, es decir, de la herramienta de integración continua. En la Fig. 14.4, pueden verse las cuatro secciones utilizadas en el archivo de configuración de CC[2] para controlar el estado

[1]Ya se ha explicado gran parte del mismo en los capítulos que fueron resolviendo las diferentes problemáticas surgidas durante la realización de este trabajo

[2]Para explicar el funcionamiento básico de la herramienta se ha utilizado un ejemplo en el que se realiza la integración continua dirigida por modelos de una familia de productos de comercio electrónico. Para otros dominios de conocimiento la estructura utilizada sería idéntica

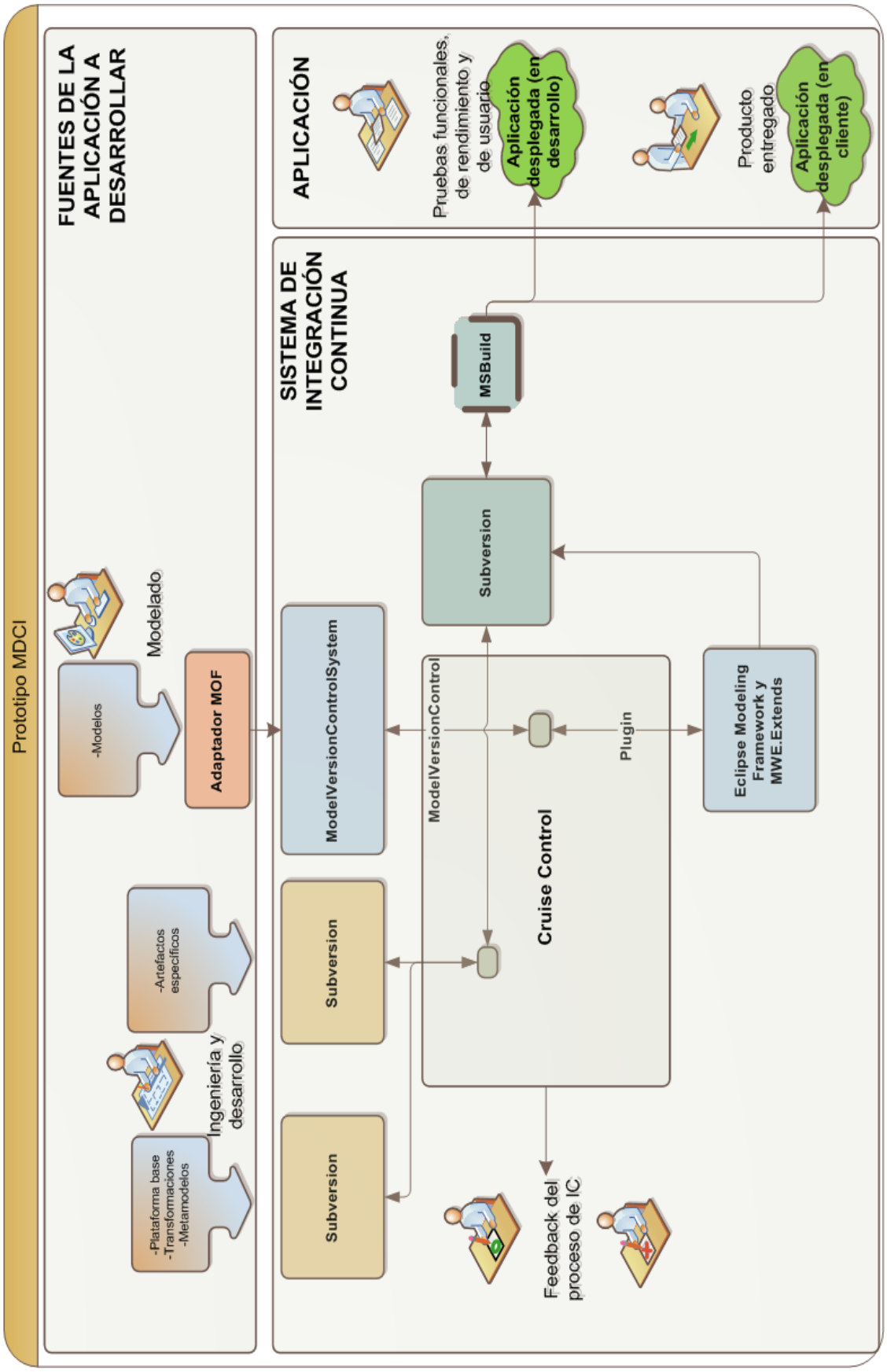


Figura 14.3: Arquitectura básica de MDCIP

ccnet.config

```
<cruisecontrol xmlns:cb="urn:ccnet.config.builder">
  <project name="Ecommerce-Common-v1.0">
    ...
  </project>
  <project name="Ecommerce-Client1-Specific-v1.0">
    ...
  </project>
  <project name="Ecommerce-Client1-VariableModels-v1.0">
    ...
  </project>
  <project name="Ecommerce-Client1-Integration-v1.0">
    ...
</cruisecontrol>
```

Figura 14.4: Archivo de configuración de Cruise Control (ccnet.config)

de cuatro[3] repositorios que separan a los diferentes tipos de artefactos con los que se trabaja en el ejemplo práctico citado:

- *Ecommerce-Common-v1.0*. Repositorio para la plataforma base, para las transformaciones, para las plantillas de generación de artefactos y para el metamodelo; es el repositorio que comparten todos los productos de una misma familia.
- *Ecommerce-Client1-Specific-v1.0*. En él se guardan los artefactos específicos para un determinado producto.
- *Ecommerce-Client1-VariableModels-v1.0*. Repositorio gobernado por un sistema de control de versiones para modelos. En él se guardan los modelos, instancias de los metamodelos utilizados para representar los artefactos esquemáticos repetitivos, que cambian de un producto a otro de una familia de productos y que siguen un mismo patrón.
- *Ecommerce-Client1-Integration-v1.0*. Repositorio de integración, utilizado para realizar una combinación de todos los artefactos justo antes de realizar el proceso de construcción del software (generación de documentación, compilación, despliegue, etc.).

Dado que el archivo de configuración de la herramienta de CI se organiza en función de los diferentes repositorios utilizados para cada caso, en los siguientes apartados se mostrará más detalle de las diferentes secciones que forman parte de archivo *ccnet.config*, al mismo tiempo que se profundizará en el detalle del prototipo.

[3]Se podrían haber utilizado menos repositorios a nivel físico pero para el prototipo se ha optado por separarlos y ofrecer así una mayor claridad

14.3.1. Repositorio del dominio de las aplicaciones

```
<project name="Ecommerce-Common-v1.0">
  <webURL>http://127.0.0.1</webURL>
  <workingDirectory>D:/CCNetProjects/Ecommerce-Common-v1.0</workingDirectory>
  <artifactDirectory>D:/Ecommerce/v1.0/Feedback</artifactDirectory>
  <state type="state" />
  <sourcecontrol type="svn">
    <trunkUrl>file:///D:/Ecommerce/v1.0/RepositoryCommon</trunkUrl>
    <workingDirectory>D:/CCNetProjects/Ecommerce-Common-v1.0</workingDirectory>
  </sourcecontrol>
  <triggers>
    <intervalTrigger name="Subversion" seconds="6" />
  </triggers>
  <labeller type="svnRevisionLabeller">
    <prefix>Common-</prefix>
    <major>1</major>
    <minor>1</minor>
    <url>file:///D:/Ecommerce/v1.0/RepositoryCommon</url>
  </labeller>
  <tasks>
    <synchronizeRepositories>
      <path>D:/CCNetProjects/Ecommerce-Client1-Integration-v1.0</path>
      <utilPath>D:/Ecommerce/v1.0/Clients/Client1/ConfigBuild</utilPath>
    </synchronizeRepositories>
  </tasks>
  <publishers>
    <xmllogger logDir="D:/Ecommerce/v1.0/Feedback/buildlogs" />
    <statistics>
      <statisticList>
        <firstMatch name="Svn Revision" xpath="//modifications/modification/changeNumber" />
      </statisticList>
    </statistics>
    <modificationHistory onlyLogWhenChangesFound="true" />
    <artifactcleanup cleanUpMethod="KeepLastXBuilds" cleanUpValue="20" />
  </publishers>
</project>
```

Cada 6 segundos se sincronizan los cambios comunes de toda la línea de productos en el repositorio de integración de cada cliente

Figura 14.5: ccnet.config (repositorio 1)

Como se puede observar en la Fig. 14.5, un proyecto está formado por varias secciones que sirven para configurar el comportamiento de la herramienta de CI. Básicamente hay que especificar la ubicación en la que la herramienta de CI guardará los artefactos de salida tras realizar las operaciones pertinentes con ellos y la forma de especificar la retroalimentación (sección *publishers*) obtenida a partir de la realización de dichas operaciones. También se especificará la ubicación del repositorio del dominio de las aplicaciones, que está gobernado por un sistema de control de versiones.

La herramienta de CI *preguntará* al sistema de control de versiones periódicamente[4] para saber si se han realizado modificaciones por parte del equipo de desa-

[4]En el ejemplo se hace cada 6 segundos pero en producción se hará cada más tiempo para evitar sobrecargas de los sistemas. Durante la realización de este documento no se han encontrado estudios que justifiquen científicamente el tiempo óptimo entre integraciones, aunque en [García-Díaz et al.,

rrolladores. Siempre que se detecten cambios, se creará un registro que almacenará el resultado obtenido de todas las operaciones impulsadas por la herramienta de CI. En este caso, únicamente se ha especificado explícitamente que se realice la tarea *synchronizeRepositories*, que consistirá en trasladar todos los cambios derivados de las acciones producidas en los tres repositorios principales (en este caso, en el repositorio del dominio de las aplicaciones) al repositorio de la imagen completa de la aplicación (repositorio de integración). Esto es así porque el proceso de construcción (compilación, generación de documentación, despliegue, etc.) se realizará siempre desde el repositorio de la imagen, cuando los artefactos provenientes de los demás repositorios se sincronicen.

14.3.2. Repositorio específico de la aplicación

```
<project name="Ecommerce-Client1-Specific-v1.0">
  <webURL>http://127.0.0.1</webURL>
  <workingDirectory>D:/CCNetProjects/Ecommerce-Client1-Specific-v1.0</workingDirectory>
  <artifactDirectory>D:/Ecommerce/v1.0/Feedback</artifactDirectory>
  <state type="state" />
  <sourcecontrol type="svn">
    <trunkUrl>file:///D:/Ecommerce/v1.0/Clients/Client1/RepositorySpecific</trunkUrl>
    <workingDirectory>D:/CCNetProjects/Ecommerce-Client1-Specific-v1.0</workingDirectory>
  </sourcecontrol>
  <triggers>
    <intervalTrigger name="Subversion" seconds="6"  />
  </triggers>
  <labeller type="svnRevisionLabeller">
    <prefix>Client1-Specific-</prefix>
    <major>1</major>
    <minor>1</minor>
    <url>file:///D:/Ecommerce/v1.0/Clients/Client1/RepositorySpecific</url>
  </labeller>
  <tasks>
    <synchronizeRepositories>
      <path>D:/CCNetProjects/Ecommerce-Client1-Integration-v1.0</path>
      <utilPath>D:/Ecommerce/v1.0/Clients/Client1/ConfigBuild</utilPath>
    </synchronizeRepositories>
  </tasks>
  <publishers>
    <xmllogger logDir="D:/Ecommerce/v1.0/Feedback/buildlogs" />
    <statistics>
      <statisticList>
        <firstMatch name="Svn Revision" xpath="//modifications/modification/changeNumber" />
      </statisticList>
    </statistics>
    <modificationHistory onlyLogWhenChangesFound="true" />
    <artifactcleanup cleanUpMethod="KeepLastXBuilds" cleanUpValue="20" />
  </publishers>
</project>
```

Cada 6 segundos se sincronizan los cambios específicos de un cliente en su repositorio de integración

Figura 14.6: ccnet.config (repositorio 2)

2010b] se trata dicha problemática

El repositorio específico de la aplicación (Fig. 14.6) tiene una configuración y un comportamiento muy parecidos al del dominio de las aplicaciones. La diferencia básica es que en este caso se utiliza un repositorio para cada cliente debido a que se trabaja con artefactos específicos para cada uno de ellos. Sin embargo, el repositorio del dominio de las aplicaciones es único y está compartido entre todos los clientes de la familia de productos, evitando así la duplicación innecesaria de artefactos e inconsistencias entre ellos. La tarea *synchronizeRepositories* funciona de forma idéntica en el sentido de que su misión es trasladar todos los cambios producidos en el repositorio específico al repositorio de la imagen completa de la aplicación, es decir, al repositorio de integración.

14.3.3. Repositorio específico de los modelos de la aplicación

La Fig. 14.7 muestra la configuración realizada para trabajar con el repositorio específico de los modelos de la aplicación. Como se puede observar, difiere levemente respecto a las configuraciones explicadas en los apartados previos. Esto es así porque en este caso, lo que se necesita es que la herramienta de CI invoque al generador de artefactos dirigido por modelos y obtenga su retroalimentación (el archivo *oAW.xml* especificado en la sección *publishers*), para tratarla de manera uniforme al resto de información mostrada por la herramienta. Otra peculiaridad es que se puede apreciar como, en lugar del sistema de control de versiones *Subversion*, se utiliza *ModelVersionControlSystem*. El último paso será invocar una tarea denominada *AutoSvnCommit*[5], que servirá para hacer un *commit* automático en el repositorio de integración, provocando de ese modo que se comience a construir el software completo para el cliente. En el caso real de utilización del prototipo mostrado, se ha optado por iniciar la construcción final del software sólo a partir de cambios en el repositorio específico de los modelos de la aplicación[6], pero podría haberse realizado a partir de cambios en cualquier otro repositorio.

14.3.4. Repositorio de la imagen completa de la aplicación

El último repositorio mostrado (Fig. 14.8) es el que se encarga realmente de construir el software a partir de todos los artefactos que se encuentren en el momento inicial dentro del repositorio de la imagen completa de la aplicación (repositorio de integración). Las tareas que se realizan son:

[5] *AutoSvnCommit*, *ModelVersionControlSystem*, *synchronizeRepositories* al igual que otras herramientas, han sido desarrolladas expresamente para la realización del prototipo. Puede consultarse todo el software utilizado y realizado en el anexo B

[6] Esa decisión puede depender de las necesidades del proyecto

```
<project name="Ecommerce-Client1-VariableModels-v1.0">
  <webURL>http://127.0.0.1</webURL>
  <workingDirectory>D:/CCNetProjects/Ecommerce-Client1-VariableModels-v1.0</workingDirectory>
  <artifactDirectory>D:/Ecommerce/v1.0/Feedback</artifactDirectory>
  <state type="state" />
  <sourcecontrol type="modelVersionControl">
    <repositoryPath>D:/Ecommerce/v1.0/Clients/Client1/RepositoryVariableModels</repositoryPath>
  </sourcecontrol>
  <triggers>
    <intervalTrigger name="ModelVersionControl" seconds="6" />
  </triggers>
  <tasks>
    <msbuild>
      <executable>C:/WINDOWS/Microsoft.NET/Framework/v3.5/MSBuild.exe</executable>
      <workingDirectory>D:/Ecommerce/v1.0/Clients/Client1/Config</workingDirectory>
      <projectFile>oAW.csproj</projectFile>
      <timeout>180000</timeout>
      <logger>C:/Program Files/CruiseControl.NET/server/ThoughtWorks.CruiseControl.MSBuild.dll</logger>
    </msbuild>
    <msbuild>
      <executable>C:/WINDOWS/Microsoft.NET/Framework/v3.5/MSBuild.exe</executable>
      <workingDirectory>D:/Ecommerce/v1.0/Clients/Client1/Config</workingDirectory>
      <projectFile>svnCommit.csproj</projectFile>
      <timeout>180000</timeout>
      <logger>C:/Program Files/CruiseControl.NET/server/ThoughtWorks.CruiseControl.MSBuild.dll</logger>
    </msbuild>
  </tasks>
  <publishers>
    <merge>
      <files>
        <file>D:/Ecommerce/v1.0/Feedback/oAW/oAW.xml</file>
      </files>
    </merge>
    <xmllogger logDir="D:/Ecommerce/v1.0/Feedback/buildlogs" />
    <statistics>
      <statisticList>
        <firstMatch name="Svn Revision" xpath="//modifications/modification/changeNumber" />
      </statisticList>
    </statistics>
    <modificationHistory onlyLogWhenChangesFound="true" />
    <artifactcleanup cleanUpMethod="KeepLastXBuilds" cleanUpValue="20" />
  </publishers>
</project>
```

Cada 6 segundos se generan los artefactos correspondientes a los cambios de versión de los modelos del repositorio de un cliente y se da orden de comenzar la construcción de su producto a partir de los artefactos contenidos en su repositorio de integración

Figura 14.7: ccnet.config (repositorio 3)

Sincronización final del repositorio de integración

El primer paso será introducir en el repositorio de integración aquellos artefactos que no estuvieran en él antes de empezar con la creación y despliegue de artefactos.

Compilación de artefactos

Para compilar los artefactos del caso de estudio, se utiliza un proyecto realizado con *MSBuild*. Podría realizarse con cualquier otra herramienta pero en este caso concreto se ha optado por reutilizar las soluciones creadas automáticamente por Visual Studio, el entorno de desarrollo utilizado para la realización de la familia de productos.

```
<project name="Ecommerce-Client1-Integration-v1.0">
  <webURL>http://127.0.0.1</webURL>
  <workingDirectory>D:/CCNetProjects/Ecommerce-Client1-VariableModels-v1.0</workingDirectory>
  <artifactDirectory>D:/Ecommerce/v1.0/Feedback</artifactDirectory>
  <state type="state" />
  <sourcecontrol type="svn">
    <trunkUrl>file:///D:/Ecommerce/v1.0/Clients/Client1/RepositoryIntegration</trunkUrl>
    <workingDirectory>D:/CCNetProjects/Ecommerce-Client1-VariableModels-v1.0</workingDirectory>
  </sourcecontrol>
  <triggers>
    <intervalTrigger name="Subversion" seconds="6" />
  </triggers>
  <labeller type="svnRevisionLabeller">
    <prefix>Client1-Integration-</prefix>
    <major>1</major>
    <minor>1</minor>
    <url>file:///D:/Ecommerce/v1.0/Clients/Client1/Reposi
  </labeller>
  <tasks>
    <synchronizeRepositories>
      <path>D:/CCNetProjects/Ecommerce-Client1-Integration-v1.0</path>
      <utilPath>D:/Ecommerce/v1.0/Clients/Client1/ConfigBuild</utilPath>
    </synchronizeRepositories>
    <msbuild>
      <executable>C:/WINDOWS/Microsoft.NET/Framework/v3.5/MSBuild.exe</executabl
      <workingDirectory>D:/CCNetProjects/Ecommerce-Client1-Integration-v1.0</wor ingDirectory>
      <projectFile>TheBeerHouse.sln</projectFile>
      <timeout>180000</timeout>
      <logger>C:/Program Files/CruiseControl.NET/server/ThoughtWorks.CruiseContr 1.MSBuild.dll</logger>
    </msbuild>
    <msbuild>
      <executable>C:/WINDOWS/Microsoft.NET/Framework/v3.5/MSBuild.exe</executable>
      <workingDirectory>D:/CCNetProjects/Ecommerce-Client1-Integration-v1.0</workingDirectory>
      <projectFile>build.csproj</projectFile>
      <timeout>180000</timeout>
      <logger>C:/Program Files/CruiseControl.NET/server/ThoughtWorks.CruiseControl.MSBuild.dll</logger>
    </msbuild>
  </tasks>
  <publishers>
    <merge>
      <files>
        <file>D:/Ecommerce/v1.0/Feedback/FxCop/FxCop.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/MsTest/MsTest.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NCover/NCover.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/ApplicationMetrics.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/AssembliesBuildOrder.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/AssembliesDependencies.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/AssembliesMetrics.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/CQLResult.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/InfoWarnings.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/NDependMain.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/TypesDependencies.xml</file>
        <file>D:/Ecommerce/v1.0/Feedback/NDepend/TypesMetrics.xml</file>
      </files>
    </merge>
    <xmllogger logDir="D:/Ecommerce/v1.0/Feedback/buildlogs" />
    <statistics>
      <statisticList>
        <firstMatch name="Svn Revision" xpath="//modifications/modification/changeNumber" />
      </statisticList>
    </statistics>
    <modificationHistory onlyLogWhenChangesFound="true" />
    <artifactcleanup cleanUpMethod="KeepLastXBuilds" cleanUpValue="20" />
  </publishers>
</project>
</cruisecontrol>
```

Cada 6 segundos, y siempre que se haya dado la orden de construcción del producto de un cliente: 1- se sincronizan los artefactos generados por la herramienta MDE en el repositorio de integración, 2- se realiza la compilación del software y 3- se hace, junto con el despliegue, la generación de pruebas, documentación u otros artefactos

Figura 14.8: ccnet.config (repositorio 4)

Generación y despliegue de artefactos

El paso final será realizar una serie de tareas, consideradas muy útiles por toda la comunidad de desarrollo de software.

1. Pruebas unitarias del producto software (utilizando la herramienta *MsTest*[7]).
2. Generación automática de documentación a partir de los comentarios del código fuente (utilizando la herramienta *NDoc*[8]).
3. Análisis de los ensamblados para comprobar si se han utilizado las guías de mejores prácticas para su construcción (utilizando la herramienta *FxCop*[9]).
4. Análisis de la estructura del código para realizar validaciones y *refactorings* (utilizando la herramienta *NDepend*[10]).
5. Análisis de la cobertura del código (utilizando la herramienta *NCover*[11]).
6. Despliegue del producto en la maquina requerida.

Con todo ello, se consigue que el desarrollo dirigido por modelos esté perfectamente integrado y desplegado en la máquina del cliente o en una máquina de pruebas, dependiendo del caso.

14.4. Estructura del espacio de trabajo

La Fig. 14.9, muestra la estructura del espacio de trabajo en el que se gestiona la evolución del proyecto comentado en el ejemplo anterior[12].

En la izquierda se observa la estructura de 4 carpetas utilizadas por la herramienta de integración continua con el objetivo de almacenar las últimas versiones de las fuentes sacadas de los repositorios y a partir de las cuales se realizará la integración. La única excepción es la carpeta *Ecommerce-Client1-Integration-v1.0*, en la cual no se introducen fuentes obtenidas de un repositorio, sino que se realiza una unión entre las fuentes de las otras 3 carpetas.

Por su parte, en la derecha se puede ver la estructura que representa una primera versión del proyecto con las siguientes características:

[7] http://msdn.microsoft.com/es-es/library/ms182489(VS.80).aspx
[8] http://ndoc.sourceforge.net/
[9] http://msdn.microsoft.com/en-us/library/bb429476(VS.80).aspx
[10] http://www.ndepend.com/
[11] http://www.ncover.com/
[12] En un entorno real dicha estructura podría ser mucho más compleja y estar repartida por diferentes máquinas

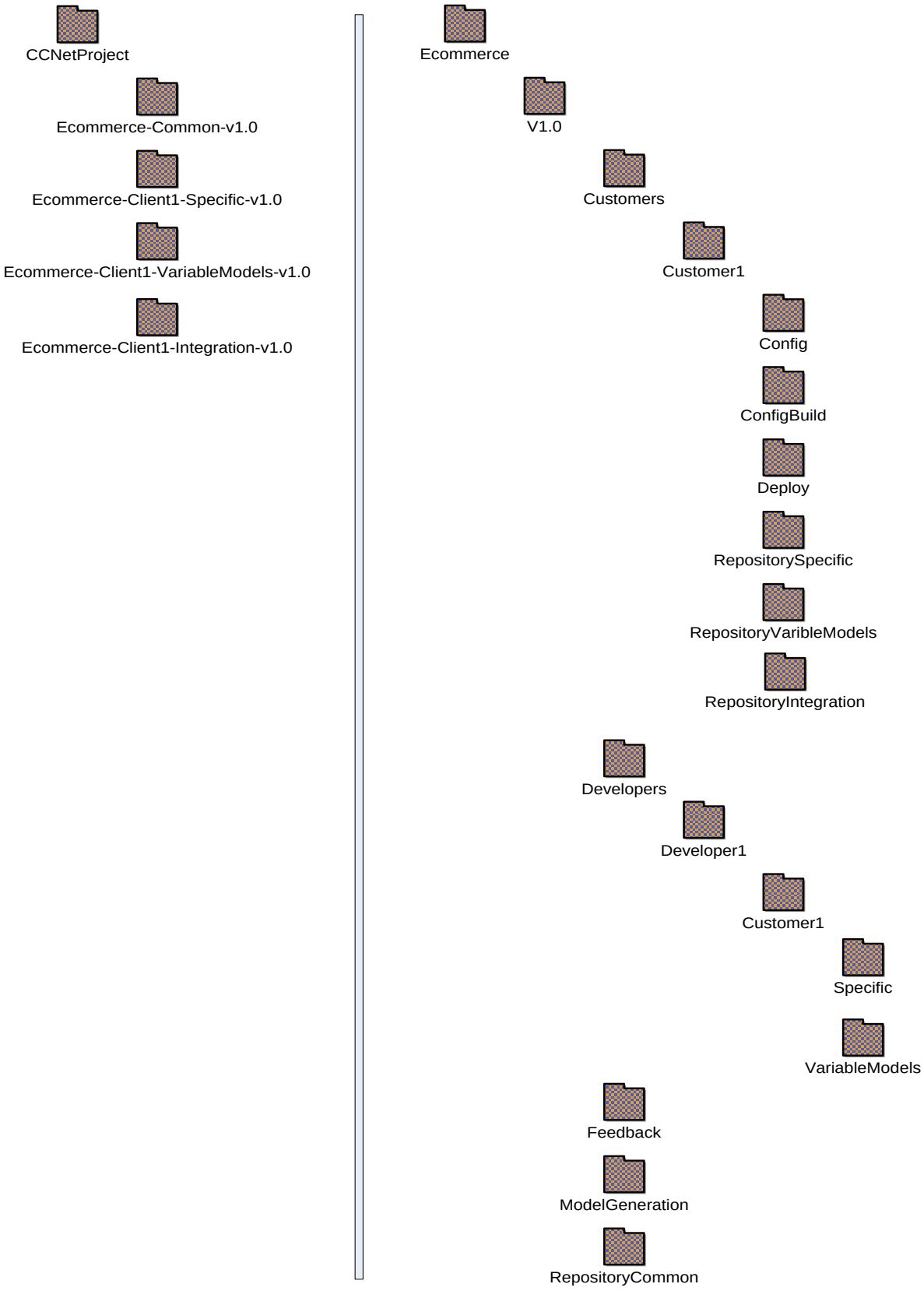


Figura 14.9: Estructura del espacio de trabajo (ejemplo)

1. Carpeta *Customers*. Espacio de trabajo para cada cliente de cada producto. Dentro hay las siguientes carpetas:

   - *Config*. Archivos de configuración necesarios para el proyecto de integración continua. Utilizado básicamente para el software contenido en la extensión *MsBuild.Extends*.
   - *ConfigBuild*. Archivos de configuración necesarios para el proceso de construcción, es decir, el despligue, la generación de pruebas, documentación y otros artefactos dentro de la integración continua.
   - *Deploy*. Carpeta en la que se despliega el sistema final.
   - *RepositorySpecific*. Repositorio para la parte específica de un producto determinado.
   - *RepositoryVariableModels*. Repositorio para la parte variable de un producto determinado (mediante el uso de un DSL de alto nivel de abstracción).
   - *RepositoryIntegration*. Repositorio final de cada producto (se une la parte común, específica y variable).

2. Carpeta *Developers*. Espacio de trabajo para el equipo de desarrollo de cada producto determinado para cada cliente específico. Dentro hay dos subcarpetas principales de trabajo (la específica y variable de cada producto).

3. Carpeta *Feedback*. Almacén para toda la retroalimentación ofrecida por la herramienta de CI. En esta carpeta hay varios archivos (p.e., msbuild-results.xml, HistoryData.xml) y varias subcarpetas con información específica de herramientas utilizadas durante el proceso de integración continua (p.e., FxCop, NDoc, oAW, MsTest, NDepend). Dicha información es utilizada por la herramienta de monitorización para mostrar resultados en un formato amigable para el usuario final.

4. Carpeta *ModelGeneration*. Sitio en el que la herramienta de generación incremental de artefactos realiza su trabajo a partir de los modelos creados o modificados por parte del equipo de desarrollo.

5. Carpeta *RepositoryCommon*. Repositorio común para toda la familia de productos.

14.5. Principales artefactos de entrada

Los listados de código mostrados a continuación son algunos de los principales *scripts* utilizados para guiar la construcción y el despliegue de los artefactos que se generan (en base al ejemplo mostrado para simplificar el entendimiento del prototipo). Por motivos de espacio no se listan en el documento otros archivos utilizados como *TheBeerHouse.NDepend.xml TheBeerHouse.FxCop* o *TheBeerHouse.NDoc*.

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="OpenArchitectureWare"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="3.5">
  <!-- imports -->
  <Import Project="$(MSBuildExtensionsPath)\MSBuildCommunityTasks\MSBuild.Community
.Tasks.Targets" />

  <!-- property groups -->
  <PropertyGroup>
    <ArtifactPath>D:\Ecommerce\v1.0\Feedback</ArtifactPath>
    <DSLProjectMetamodel>CMSLanguage</DSLProjectMetamodel>
    <DSLProjectPath>D:\Ecommerce\v1.0\Clients\Client1\Config</DSLProjectPath>
    <EclipsePath>D:\Eclipse</EclipsePath>
    <EclipseWorkspacePath>D:\Eclipse\Workspace</EclipseWorkspacePath>
    <JavaLibPath>C:\Program Files\Java\jre1.6.0\lib</JavaLibPath>
    <MweExtendsJar>D:\Eclipse\workspace\MWE.Extends\MWE.Extends.jar</MweExtendsJar>
    <OpenArchitectureWareAssembly>D:\Recursos\Investigacion\Doctorado\Software
\MsBuild.Extends\OpenArchitectureWare\bin\Release\OpenArchitectureWare.dll
</OpenArchitectureWareAssembly>
    <OpenArchitectureWareOutput>$(ArtifactPath)\oAW\oAW.xml
</OpenArchitectureWareOutput>
    <Script>oaw.cmd</Script>
    <WorkingDirectory>$(DSLProjectPath)</WorkingDirectory>
    <Workflow>CMSLanguageProject.oaw</Workflow>
  </PropertyGroup>

  <!-- targets -->
  <ItemGroup>
    <CorrectWebConfigs Include="$(OutputPath)\$(Environment)\**
\web.config.$(Environment)" />
  </ItemGroup>

  <UsingTask TaskName="OpenArchitectureWareTask"
AssemblyFile="$(OpenArchitectureWareAssembly)" />
  <Target Name="OpenArchitectureWare">
    <OpenArchitectureWareTask DSLProjectMetamodel="$(DSLProjectMetamodel)"
DSLProjectPath="$(DSLProjectPath)" EclipsePath="$(EclipsePath)"
EclipseWorkspacePath="$(EclipseWorkspacePath)" JavaLibPath="$(JavaLibPath)"
OpenArchitectureWareOutput="$(OpenArchitectureWareOutput)" Script="$(Script)"
WorkingDirectory="$(WorkingDirectory)" Workflow="$(Workflow)"
MweExtendsJar="$(MweExtendsJar)" >
    </OpenArchitectureWareTask>
  </Target>

</Project>
```

Código fuente 14.1: Archivo `oAW.csproj`

```
ï»¿<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="AutoSvnCommit" xmlns="http://schemas.microsoft.com
/developer/msbuild/2003" ToolsVersion="3.5">

  <!-- property groups -->
  <PropertyGroup>
    <AutoSvnCommitAssembly>D:\Recursos\Investigacion\Doctorado\Software
```

```
\MsBuild.Extends\AutoSvnCommit\bin\Release\AutoSvnCommit.dll
</AutoSvnCommitAssembly>
    <LocalPath>D:\Ecommerce\v1.0\Developers\Developer1\Client1
\VariableModels</LocalPath>
    <Message>AutoSvnCommit!</Message>
    <WorkingDirectory>C:\Program Files\Subversion\bin</WorkingDirectory>
  </PropertyGroup>

  <!-- tasks -->
  <UsingTask TaskName="AutoSvnCommitTask"
AssemblyFile="$(AutoSvnCommitAssembly)" />
  <Target Name="AutoSvnCommit">
    <AutoSvnCommitTask LocalPath="$(LocalPath)" Message="$(Message)"
WorkingDirectory="$(WorkingDirectory)">
    </AutoSvnCommitTask>
  </Target>

</Project>
```

Código fuente 14.2: Archivo `svnCommit.csproj`

```
Microsoft Visual Studio Solution File, Format Version 11.00
# Visual Studio 2010
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "TheBeerHouse",
"TheBeerHouse\TheBeerHouse.csproj", "{7F12F089-7C02-46EF-81C1-87910051A903}"
EndProject
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "TestProject",
"TestProject\TestProject.csproj", "{E88F0FB6-C653-492C-B761-504B1040BE5D}"
EndProject
Global
        GlobalSection(TeamFoundationVersionControl) = preSolution
                SccNumberOfProjects = 5
                SccEnterpriseProvider = {4CA58AB2-18FA-4F8D-95D4-32DDF27D184C}
                SccTeamFoundationServer = https://team.managedfusion.com/
                SccLocalPath0 = .
                SccProjectUniqueName1 = Database\\Database.dbp
                SccProjectName1 = Database
                SccLocalPath1 = Database
                SccProjectUniqueName2 = TheBeerHouse\\TheBeerHouse.csproj
                SccProjectName2 = TheBeerHouse
                SccLocalPath2 = TheBeerHouse
                SccProjectUniqueName3 = ..\\..\\Managed\u0020Fusion\\ManagedFusion
\\Source\\ManagedFusion.csproj
                SccProjectTopLevelParentUniqueName3 = TheBeerHouse.sln
                SccProjectName3 = ../../Managed\u0020Fusion/ManagedFusion/Source
                SccLocalPath3 = ..\\..\\Managed\u0020Fusion\\ManagedFusion\\Source
                SccProjectUniqueName4 = ..\\..\\Managed\u0020Fusion
\\ManagedFusion.Web\\Source\\ManagedFusion.Web.csproj
                SccProjectTopLevelParentUniqueName4 = TheBeerHouse.sln
                SccProjectName4 = ../../Managed\u0020Fusion/ManagedFusion.Web/Source
                SccLocalPath4 = ..\\..\\Managed\u0020Fusion\\ManagedFusion.Web
\\Source
        EndGlobalSection
        GlobalSection(TestCaseManagementSettings) = postSolution
                CategoryFile = TheBeerHouse.vsmdi
        EndGlobalSection
        GlobalSection(SolutionConfigurationPlatforms) = preSolution
                Debug|Any CPU = Debug|Any CPU
                Release|Any CPU = Release|Any CPU
        EndGlobalSection
        GlobalSection(ProjectConfigurationPlatforms) = postSolution
                {7F12F089-7C02-46EF-81C1-87910051A903}.Debug|Any CPU.ActiveCfg =
Release|Any CPU
                {7F12F089-7C02-46EF-81C1-87910051A903}.Debug|Any CPU.Build.0 =
Release|Any CPU
                {7F12F089-7C02-46EF-81C1-87910051A903}.Release|Any CPU.ActiveCfg =
Release|Any CPU
                {7F12F089-7C02-46EF-81C1-87910051A903}.Release|Any CPU.Build.0 =
```

```
Release|Any CPU
                {E88F0FB6-C653-492C-B761-504B1040BE5D}.Debug|Any CPU.ActiveCfg =
Release|Any CPU
                {E88F0FB6-C653-492C-B761-504B1040BE5D}.Debug|Any CPU.Build.0 =
Release|Any CPU
                {E88F0FB6-C653-492C-B761-504B1040BE5D}.Release|Any CPU.ActiveCfg =
Release|Any CPU
                {E88F0FB6-C653-492C-B761-504B1040BE5D}.Release|Any CPU.Build.0 =
Release|Any CPU
        EndGlobalSection
        GlobalSection(SolutionProperties) = preSolution
                HideSolutionNode = FALSE
        EndGlobalSection
EndGlobal
```

Código fuente 14.3: Archivo `TheBeerHouse.sln`

```
<Project ToolsVersion="3.5" DefaultTargets="Build" xmlns=
"http://schemas.microsoft.com
/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>9.0.30729</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{7F12F089-7C02-46EF-81C1-87910051A903}</ProjectGuid>
    <ProjectTypeGuids>{603c0e0b-db56-11dc-be95-000d561079b0};
{349c5851-65df-11da-9384-00065b846f21};{fae04ec0-301f-11d3-bf4b-00c04f79efbc}
</ProjectTypeGuids>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>TheBeerHouse</RootNamespace>
    <AssemblyName>TheBeerHouse</AssemblyName>
    <TargetFrameworkVersion>v3.5</TargetFrameworkVersion>
    <MvcBuildViews>true</MvcBuildViews>
    <SccProjectName>%24/ASP.NET MVC Website Programming Book/Code/TheBeerHouse
</SccProjectName>
    <SccLocalPath>.</SccLocalPath>
    <SccAuxPath>https://team.managedfusion.com</SccAuxPath>
    <SccProvider>{4CA58AB2-18FA-4F8D-95D4-32DDF27D184C}</SccProvider>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
    <DocumentationFile>bin\TheBeerHouse.XML</DocumentationFile>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="ManagedFusion, Version=1.0.3368.24340, Culture=neutral,
processorArchitecture=MSIL">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>lib\ManagedFusion.dll</HintPath>
    </Reference>
    <Reference Include="ManagedFusion.Web, Version=0.0.0.0, Culture=neutral,
processorArchitecture=MSIL">
      <SpecificVersion>False</SpecificVersion>
```

```
      <HintPath>lib\ManagedFusion.Web.dll</HintPath>
    </Reference>
    <Reference Include="System" />
    <Reference Include="System.Data" />
    <Reference Include="System.Core">
      <RequiredTargetFramework>3.5</RequiredTargetFramework>
    </Reference>
    <Reference Include="System.Data.Linq">
      <RequiredTargetFramework>3.5</RequiredTargetFramework>
    </Reference>
    <Reference Include="System.Runtime.Serialization">
      <RequiredTargetFramework>3.0</RequiredTargetFramework>
    </Reference>
    <Reference Include="System.ServiceModel">
      <RequiredTargetFramework>3.0</RequiredTargetFramework>
    </Reference>
    <Reference Include="System.ServiceModel.Web">
      <RequiredTargetFramework>3.5</RequiredTargetFramework>
    </Reference>
    <Reference Include="System.Web.Abstractions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\..\..\..\..\..\Program Files\Microsoft ASP.NET\ASP.NET MVC RC
\Assemblies\System.Web.Abstractions.dll</HintPath>
      <RequiredTargetFramework>3.5</RequiredTargetFramework>
    </Reference>
    <Reference Include="System.Web.Mvc, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\..\..\..\..\..\Program Files\Microsoft ASP.NET\ASP.NET MVC RC
\Assemblies\System.Web.Mvc.dll</HintPath>
    </Reference>
    <Reference Include="System.Web.Routing, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\..\..\..\..\..\Program Files\Microsoft ASP.NET\ASP.NET MVC RC
\Assemblies\System.Web.Routing.dll</HintPath>
      <RequiredTargetFramework>3.5</RequiredTargetFramework>
    </Reference>
    <Reference Include="System.Drawing" />
    <Reference Include="System.Web" />
    <Reference Include="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL" />
    <Reference Include="System.Configuration" />
    <Reference Include="System.Web.Services" />
    <Reference Include="System.XML" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="App_GlobalResources\Messages.designer.cs">
      <AutoGen>True</AutoGen>
      <DesignTime>True</DesignTime>
      <DependentUpon>Messages.resx</DependentUpon>
    </Compile>
    <Compile Include="Configuration\ArticlesElement.cs" />
    <Compile Include="Configuration\CommerceElement.cs" />
    <Compile Include="Configuration\ContactFormElement.cs" />
    <Compile Include="Configuration\ForumsElement.cs" />
    <Compile Include="Configuration\NewslettersElement.cs" />
    <Compile Include="Configuration\PollsElement.cs" />
    <Compile Include="Configuration\TheBeerHouseArticlesSection.cs" />
    <Compile Include="Configuration\TheBeerHouseCommerceSection.cs" />
    <Compile Include="Configuration\TheBeerHouseForumsSection.cs" />
    <Compile Include="Configuration\TheBeerHouseNewslettersSection.cs" />
    <Compile Include="Configuration\TheBeerHousePollsSection.cs" />
    <Compile Include="Configuration\TheBeerHouseSection.cs" />
    <Compile Include="Controllers\ActionFilters\AtomResult.cs" />
    <Compile Include="Controllers\ActionFilters\ResponseType.cs" />
    <Compile Include="Controllers\ActionFilters\ServiceAttribute.cs" />
```

```
    <Compile Include="Controllers\ArticleController.cs" />
    <Compile Include="Controllers\BaseController.cs" />
    <Compile Include="Controllers\CommerceController.cs" />
    <Compile Include="Controllers\ForumController.cs" />
    <Compile Include="Controllers\LocalizationController.cs" />
    <Compile Include="Controllers\NewsletterController.cs" />
    <Compile Include="Controllers\PollController.cs" />
    <Compile Include="Controllers\UserController.cs" />
    <Compile Include="Default.aspx.cs">
      <DependentUpon>Default.aspx</DependentUpon>
      <SubType>ASPXCodeBehind</SubType>
    </Compile>
    <Compile Include="Global.asax.cs">
      <DependentUpon>Global.asax</DependentUpon>
    </Compile>
    <Compile Include="Models\Article.cs" />
    <Compile Include="Models\ArticleCollectionWrapper.cs" />
    <Compile Include="Models\ArticleQueries.cs" />
    <Compile Include="Models\Comment.cs" />
    <Compile Include="Models\CommerceQueries.cs" />
    <Compile Include="Models\Department.cs" />
    <Compile Include="Models\Extensions.cs" />
    <Compile Include="Models\ForumQueries.cs" />
    <Compile Include="Models\IPagination.cs" />
    <Compile Include="Models\Iso3166CountryCodes.cs" />
    <Compile Include="Models\ManageUserInformation.cs" />
    <Compile Include="Models\Pagination.cs" />
    <Compile Include="Models\PollQueries.cs" />
    <Compile Include="Models\Post.cs" />
    <Compile Include="Models\Product.cs" />
    <Compile Include="Models\ProfileInformation.cs" />
    <Compile Include="Models\ShippingMethod.cs" />
    <Compile Include="Models\ShoppingCart.cs" />
    <Compile Include="Models\ShoppingCartItem.cs" />
    <Compile Include="Models\TheBeerHouse.cs">
      <DependentUpon>TheBeerHouse.dbml</DependentUpon>
    </Compile>
    <Compile Include="Models\TheBeerHouse.designer.cs">
      <AutoGen>True</AutoGen>
      <DesignTime>True</DesignTime>
      <DependentUpon>TheBeerHouse.dbml</DependentUpon>
    </Compile>
    <Compile Include="Models\UserInformation.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
    <Compile Include="Properties\Resources.Designer.cs">
      <AutoGen>True</AutoGen>
      <DesignTime>True</DesignTime>
      <DependentUpon>Resources.resx</DependentUpon>
    </Compile>
    <Compile Include="Views\Article\ViewArticle.aspx.cs">
      <DependentUpon>ViewArticle.aspx</DependentUpon>
      <SubType>ASPXCodeBehind</SubType>
    </Compile>
    <Compile Include="Views\Article\ViewArticle.aspx.designer.cs">
      <DependentUpon>ViewArticle.aspx</DependentUpon>
    </Compile>
    <Compile Include="Views\Shared\Article\ArticleItem.ascx.cs">
      <DependentUpon>ArticleItem.ascx</DependentUpon>
      <SubType>ASPXCodeBehind</SubType>
    </Compile>
    <Compile Include="Views\Shared\Article\ArticleItem.ascx.designer.cs">
      <DependentUpon>ArticleItem.ascx</DependentUpon>
    </Compile>
    <Compile Include="Views\Shared\Pager.ascx.cs">
      <DependentUpon>Pager.ascx</DependentUpon>
      <SubType>ASPXCodeBehind</SubType>
    </Compile>
    <Compile Include="Views\Shared\Pager.ascx.designer.cs">
```

```
      <DependentUpon>Pager.ascx</DependentUpon>
    </Compile>
    <Compile Include="Views\Shared\Site.Master.cs">
      <DependentUpon>Site.Master</DependentUpon>
      <SubType>ASPXCodeBehind</SubType>
    </Compile>
    <Compile Include="Views\Shared\Site.Master.designer.cs">
      <DependentUpon>Site.Master</DependentUpon>
    </Compile>
  </ItemGroup>
  <ItemGroup>
    <Content Include="Default.aspx" />
    <Content Include="Global.asax" />
    <Content Include="Views\Article\CategoryIndex.aspx" />
    <Content Include="Views\Article\CreateArticle.aspx" />
    <Content Include="Views\Article\CreateCategory.aspx" />
    <Content Include="Views\Article\Index.aspx" />
    <Content Include="Views\Article\ManageArticles.aspx" />
    <Content Include="Views\Article\ManageCategories.aspx" />
    <Content Include="Views\Article\ManageComments.aspx" />
    <Content Include="Views\Article\RemoveArticle.aspx" />
    <Content Include="Views\Article\RemoveCategory.aspx" />
    <Content Include="Views\Commerce\CompleteOrder.aspx" />
    <Content Include="Views\Commerce\CreateDepartment.aspx" />
    <Content Include="Views\Commerce\CreateProduct.aspx" />
    <Content Include="Views\Commerce\ManageDepartments.aspx" />
    <Content Include="Views\Commerce\ManageOrders.aspx" />
    <Content Include="Views\Commerce\ManageProducts.aspx" />
    <Content Include="Views\Commerce\ManageShipping.aspx" />
    <Content Include="Views\Commerce\ManageStore.aspx" />
    <Content Include="Views\Commerce\OrderDetail.aspx" />
    <Content Include="Views\Commerce\ViewProduct.aspx" />
    <Content Include="Views\Commerce\TransactionError.aspx" />
    <Content Include="Views\Commerce\Index.aspx" />
    <Content Include="Views\Commerce\ViewDepartment.aspx" />
    <Content Include="Views\Commerce\ViewShoppingCart.aspx" />
    <Content Include="Views\Forum\CreateForum.aspx" />
    <Content Include="Views\Forum\CreatePost.aspx" />
    <Content Include="Views\Forum\Index.aspx" />
    <Content Include="Views\Forum\ManageForums.aspx" />
    <Content Include="Views\Forum\ManagePosts.aspx" />
    <Content Include="Views\Forum\RemoveForum.aspx" />
    <Content Include="Views\Forum\ViewForum.aspx" />
    <Content Include="Views\Forum\ViewPost.aspx" />
    <Content Include="Views\Localization\TestLocalization.aspx" />
    <Content Include="Views\Newsletter\CreateNewsletter.aspx" />
    <Content Include="Views\Newsletter\Index.aspx" />
    <Content Include="Views\Newsletter\ManageNewsletters.aspx" />
    <Content Include="Views\Poll\CreatePoll.aspx" />
    <Content Include="Views\Poll\Index.aspx" />
    <Content Include="Views\Poll\ManagePolls.aspx" />
    <Content Include="Views\Poll\RemovePoll.aspx" />
    <Content Include="Views\Shared\Article\AdminSidebar.ascx" />
    <Content Include="Views\Shared\Article\CategoryItem.ascx" />
    <Content Include="Views\Shared\Article\CommentItem.ascx" />
    <Content Include="Views\Shared\Commerce\CommerceSidebar.ascx" />
    <Content Include="Views\Shared\Commerce\DepartmentItem.ascx" />
    <Content Include="Views\Shared\Commerce\AdminProductItem.ascx" />
    <Content Include="Views\Shared\Commerce\ProductItem.ascx" />
    <Content Include="Views\Shared\Forum\AdminSidebar.ascx" />
    <Content Include="Views\Shared\Message.ascx" />
    <Content Include="Views\Shared\Newsletter\NewsletterStatus.ascx" />
    <Content Include="Views\Shared\Poll\AdminSidebar.ascx" />
    <Content Include="Views\Shared\Poll\PollItem.ascx" />
    <Content Include="Views\Shared\Poll\PollResultItem.ascx" />
    <Content Include="Views\User\ChangePassword.aspx" />
    <Content Include="Views\User\EditUser.aspx" />
    <Content Include="Views\User\ForgotPassword.aspx" />
```

```
    <Content Include="Views\User\Login.aspx" />
    <Content Include="Views\User\ManageRole.aspx" />
    <Content Include="Views\User\ManageUser.aspx" />
    <Content Include="Views\User\Register.aspx" />
    <Content Include="Views\User\UserProfile.aspx" />
    <Content Include="Web.config" />
    <Content Include="Views\Localization\App_LocalResources
\TestLocalization.it-IT.resx" />
    <Content Include="Views\Localization\App_LocalResources
\TestLocalization.resx" />
    <Content Include="Content\images\3beers.jpg" />
    <Content Include="Content\images\Beers.gif" />
    <Content Include="Content\images\Camera.gif" />
    <Content Include="Content\images\centerstage\img01.gif" />
    <Content Include="Content\images\centerstage\img02.jpg" />
    <Content Include="Content\images\centerstage\img03.jpg" />
    <Content Include="Content\images\centerstage\img04.gif" />
    <Content Include="Content\images\DeleteSymbol.png" />
    <Content Include="Content\images\Diary.gif" />
    <Content Include="Content\images\Diary2.gif" />
    <Content Include="Content\images\EditSymbol.png" />
    <Content Include="Content\images\error.png" />
    <Content Include="Content\images\feed.png" />
    <Content Include="Content\images\glass.jpg" />
    <Content Include="Content\images\Guitar.gif" />
    <Content Include="Content\images\info.png" />
    <Content Include="Content\images\lock.gif" />
    <Content Include="Content\images\News.gif" />
    <Content Include="Content\images\noimage.gif" />
    <Content Include="Content\images\PayPal.gif" />
    <Content Include="Content\images\poll-graph.gif" />
    <Content Include="Content\images\Question.gif" />
    <Content Include="Content\images\Question2.gif" />
    <Content Include="Content\images\stars10.gif" />
    <Content Include="Content\images\stars15.gif" />
    <Content Include="Content\images\stars20.gif" />
    <Content Include="Content\images\stars25.gif" />
    <Content Include="Content\images\stars30.gif" />
    <Content Include="Content\images\stars35.gif" />
    <Content Include="Content\images\stars40.gif" />
    <Content Include="Content\images\stars45.gif" />
    <Content Include="Content\images\stars50.gif" />
    <Content Include="Content\images\Store\can1_small.jpg" />
    <Content Include="Content\images\Store\cap1_small.jpg" />
    <Content Include="Content\images\Store\glass1_small.jpg" />
    <Content Include="Content\images\Store\glass2_small.jpg" />
    <Content Include="Content\images\Store\glass3_small.jpg" />
    <Content Include="Content\images\Store\glass4_small.jpg" />
    <Content Include="Content\images\Store\glass5_small.jpg" />
    <Content Include="Content\images\Store\glass6_small.jpg" />
    <Content Include="Content\images\Store\keychain1_full.jpg" />
    <Content Include="Content\images\Store\keychain1_small.jpg" />
    <Content Include="Content\images\Store\tshirt1_full.jpg" />
    <Content Include="Content\images\Store\tshirt1_small.jpg" />
    <Content Include="Content\images\Store\tshirt2_full.jpg" />
    <Content Include="Content\images\Store\tshirt2_small.jpg" />
    <Content Include="Content\images\Store\tshirt3_full.jpg" />
    <Content Include="Content\images\Store\tshirt3_small.jpg" />
    <Content Include="Content\images\success.png" />
    <Content Include="Content\images\thebeerpub_logo.png" />
    <Content Include="Content\images\Tshirt.gif" />
    <Content Include="Content\images\validation.png" />
    <Content Include="Content\images\warning.png" />
    <Content Include="Content\images\wiley.gif" />
    <Content Include="Content\images\wrox.gif" />
    <Content Include="Content\scripts\article.js" />
    <Content Include="Content\scripts\forums.js" />
    <Content Include="Content\scripts\global.js" />
```

```
    <Content Include="Content\scripts\jquery-1.3.2-vsdoc.js" />
    <Content Include="Content\scripts\jquery-1.3.2.js" />
    <Content Include="Content\scripts\jquery-1.3.2.min-vsdoc.js" />
    <Content Include="Content\scripts\jquery-1.3.2.min.js" />
    <Content Include="Content\scripts\manage-articles.js" />
    <Content Include="Content\scripts\manage-categories.js" />
    <Content Include="Content\scripts\manage-comments.js" />
    <Content Include="Content\scripts\manage-department.js" />
    <Content Include="Content\scripts\manage-forums.js" />
    <Content Include="Content\scripts\manage-newsletter.js" />
    <Content Include="Content\scripts\manage-polls.js" />
    <Content Include="Content\scripts\manage-product.js" />
    <Content Include="Content\scripts\MicrosoftAjax.debug.js" />
    <Content Include="Content\scripts\MicrosoftAjax.js" />
    <Content Include="Content\scripts\MicrosoftMvcAjax.debug.js" />
    <Content Include="Content\scripts\MicrosoftMvcAjax.js" />
    <Content Include="Content\scripts\poll.js" />
    <Content Include="Content\scripts\tiny_mce\langs\en.js" />
    <Content Include="Content\scripts\tiny_mce\license.txt" />
    <Content Include="Content\scripts\tiny_mce\plugins\advhr\css\advhr.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\advhr\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advhr
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advhr\js\rule.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advhr\langs\en_dlg.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advhr\rule.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\advimage\css\advimage.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\advimage\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advimage
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advimage\image.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\advimage\img\sample.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\advimage\js\image.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advimage\langs\en_dlg.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advlink\css\advlink.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\advlink\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advlink
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advlink\js\advlink.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advlink\langs\en_dlg.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\advlink\link.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\contextmenu
\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\contextmenu
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\img\alert.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\img\button.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\img\buttons.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\img\confirm.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\img\corners.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\img\horizontal.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\img\vertical.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\skins\clearlooks2\window.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\inlinepopups
\template.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\media\css\content.css" />
```

```
    <Content Include="Content\scripts\tiny_mce\plugins\media\css\media.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\img\flash.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\img\flv_player.swf" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\img\quicktime.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\img\realmedia.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\img\shockwave.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\img\trans.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\media
\img\windowsmedia.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\media\js\embed.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\media\js\media.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\media\langs\en_dlg.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\media\media.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\blank.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\css\blank.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\css\pasteword.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\js\pastetext.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\js\pasteword.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\langs\en_dlg.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\pastetext.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\paste\pasteword.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\safari\blank.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\safari
\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\safari
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\spellchecker\css
\content.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\spellchecker
\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\spellchecker
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\spellchecker\img
\wline.gif" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\abbr.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\acronym.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\attributes.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\cite.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\css\attributes.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\css\popup.css" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\del.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\editor_plugin.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\editor_plugin_src.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\ins.htm" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\js\abbr.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\js\acronym.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\js\attributes.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\js\cite.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\js\del.js" />
```

```
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\js\element_common.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras\js\ins.js" />
    <Content Include="Content\scripts\tiny_mce\plugins\xhtmlxtras
\langs\en_dlg.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\about.htm" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\anchor.htm" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\charmap.htm" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced
\color_picker.htm" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced
\editor_template.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced
\editor_template_src.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\image.htm" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced
\img\colorpicker.jpg" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\img\icons.gif" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\js\about.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\js\anchor.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\js\charmap.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced
\js\color_picker.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\js\image.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\js\link.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced
\js\source_editor.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\langs\en.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\langs\en_dlg.js" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\link.htm" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default
\content.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default
\dialog.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default\img
\buttons.png" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default\img
\items.gif" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default\img
\menu_arrow.gif" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default\img
\menu_check.gif" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default\img
\progress.gif" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default\img
\tabs.gif" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\default
\ui.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7
\content.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7
\dialog.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7\img
\button_bg.png" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7\img
\button_bg_black.png" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7\img
\button_bg_silver.png" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7
\ui.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7
\ui_black.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\skins\o2k7
\ui_silver.css" />
    <Content Include="Content\scripts\tiny_mce\themes\advanced\source_editor.htm" />
    <Content Include="Content\scripts\tiny_mce\themes\simple\editor_template.js" />
    <Content Include="Content\scripts\tiny_mce\themes\simple
\editor_template_src.js" />
```

```
    <Content Include="Content\scripts\tiny_mce\themes\simple\img\icons.gif" />
    <Content Include="Content\scripts\tiny_mce\themes\simple\langs\en.js" />
    <Content Include="Content\scripts\tiny_mce\themes\simple\skins\default
\content.css" />
    <Content Include="Content\scripts\tiny_mce\themes\simple\skins\default
\ui.css" />
    <Content Include="Content\scripts\tiny_mce\themes\simple\skins\o2k7
\content.css" />
    <Content Include="Content\scripts\tiny_mce\themes\simple\skins\o2k7\img
\button_bg.png" />
    <Content Include="Content\scripts\tiny_mce\themes\simple\skins\o2k7\ui.css" />
    <Content Include="Content\scripts\tiny_mce\tiny_mce.js" />
    <Content Include="Content\scripts\tiny_mce\tiny_mce_popup.js" />
    <Content Include="Content\scripts\tiny_mce\tiny_mce_src.js" />
    <Content Include="Content\scripts\tiny_mce\utils\editable_selects.js" />
    <Content Include="Content\scripts\tiny_mce\utils\form_utils.js" />
    <Content Include="Content\scripts\tiny_mce\utils\mctabs.js" />
    <Content Include="Content\scripts\tiny_mce\utils\validate.js" />
    <Content Include="Content\styles\modules.css" />
    <Content Include="Content\styles\site.css" />
    <Content Include="Views\Shared\Site.Master" />
    <Content Include="Views\Web.config" />
    <None Include="Models\ArticlesClassDiagram.cd" />
    <None Include="Models\ForumClassDiagram.cd" />
    <None Include="Models\PollsClassDiagram.cd" />
    <None Include="Models\TheBeerHouse.dbml">
      <Generator>MSLinqToSQLGenerator</Generator>
      <LastGenOutput>TheBeerHouse.designer.cs</LastGenOutput>
      <SubType>Designer</SubType>
    </None>
  </ItemGroup>
  <ItemGroup>
    <None Include="App_GlobalResources\Iso3166CountryCodes.csv" />
  </ItemGroup>
  <ItemGroup>
    <Service Include="{3259AA49-8AA1-44D3-9025-A0B520596A8C}" />
  </ItemGroup>
  <ItemGroup>
    <Content Include="App_GlobalResources\Messages.it-IT.resx" />
    <Content Include="App_GlobalResources\Messages.resx">
      <Generator>GlobalResourceProxyGenerator</Generator>
      <LastGenOutput>Messages.designer.cs</LastGenOutput>
    </Content>
    <Content Include="Content\scripts\commerece.js" />
    <Content Include="Content\scripts\manage-roles.js" />
    <Content Include="Content\scripts\manage-users.js" />
    <Content Include="Content\scripts\register.js" />
    <Content Include="Resources\Autumn_Leaves.jpg" />
    <Content Include="Resources\Creek.jpg" />
    <Content Include="Resources\Desert_Landscape.jpg" />
    <Content Include="Resources\Dock.jpg" />
    <Content Include="Resources\Forest_Flowers.jpg" />
    <Content Include="Resources\Forest.jpg" />
    <Content Include="Resources\Frangipani_Flowers.jpg" />
    <Content Include="Resources\Garden.jpg" />
    <Content Include="Resources\Green_Sea_Turtle.jpg" />
    <Content Include="Resources\Humpback_Whale.jpg" />
    <Content Include="Views\Article\ViewArticle.aspx" />
    <Content Include="Views\Shared\Article\ArticleItem.ascx" />
    <Content Include="Views\Shared\Pager.ascx" />
  </ItemGroup>
  <ItemGroup>
    <None Include="Models\CommerceClassDiagram.cd" />
    <None Include="Models\TheBeerHouse.dbml.layout">
      <DependentUpon>TheBeerHouse.dbml</DependentUpon>
    </None>
    <None Include="Resources\CI.txt" />
    <None Include="Resources\Image1.jpg" />
```

```
    <Content Include="Web.Membership.config" />
    <EmbeddedResource Include="Properties\Resources.resx">
      <Generator>ResXFileCodeGenerator</Generator>
      <LastGenOutput>Resources.Designer.cs</LastGenOutput>
    </EmbeddedResource>
  </ItemGroup>
  <ItemGroup>
    <Folder Include="App_Data\" />
    <Folder Include="App_LocalResources\" />
  </ItemGroup>
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
  <Import Project="$(MSBuildExtensionsPath)\Microsoft\VisualStudio\v9.0
\WebApplications
\Microsoft.WebApplication.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" Condition="'$(MvcBuildViews)'=='true'">
    <AspNetCompiler VirtualPath="temp" PhysicalPath="$(ProjectDir)\..
\$(ProjectName)" />
  </Target>
  <ProjectExtensions>
    <VisualStudio>
      <FlavorProperties GUID="{349c5851-65df-11da-9384-00065b846f21}">
        <WebProjectProperties>
          <UseIIS>False</UseIIS>
          <AutoAssignPort>False</AutoAssignPort>
          <DevelopmentServerPort>55600</DevelopmentServerPort>
          <DevelopmentServerVPath>/</DevelopmentServerVPath>
          <IISUrl>
          </IISUrl>
          <NTLMAuthentication>False</NTLMAuthentication>
          <UseCustomServer>False</UseCustomServer>
          <CustomServerUrl>
          </CustomServerUrl>
          <SaveServerSettingsInUserFile>False</SaveServerSettingsInUserFile>
        </WebProjectProperties>
      </FlavorProperties>
    </VisualStudio>
  </ProjectExtensions>
</Project>
```

Código fuente 14.4: Archivo `TheBeerHouse.csproj`

```
ï»¿<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="RunAll" xmlns="http://schemas.microsoft.com/developer
/msbuild/2003" ToolsVersion="4.0">
  <!-- imports -->
  <Import Project="$(MSBuildExtensionsPath)\MSBuildCommunityTasks
\MSBuild.Community.Tasks.Targets" />
  <!-- property groups -->
  <PropertyGroup>
    <ArtifactPath>D:\Ecommerce\v1.0\Feedback</ArtifactPath>
    <IntegrationPath>D:\CCNetProjects\Ecommerce-Client1-Integration-v1.0
</IntegrationPath>
    <ProjectName>TheBeerHouse</ProjectName>
    <DataBaseServer>VICEN1\SQLEXPRESS</DataBaseServer>
    <DataBaseName>$(ProjectName)</DataBaseName>
    <FileUpgradeFlags>
    </FileUpgradeFlags>
    <UpgradeBackupLocation>
    </UpgradeBackupLocation>
    <OldToolsVersion>3.5</OldToolsVersion>
  </PropertyGroup>
  <PropertyGroup>
    <DeployWebAppInputPath>D:\CCNetProjects\Ecommerce-Client1-Integration-v1.0
```

```
\$(ProjectName)</DeployWebAppInputPath>
    <DeployWebAppOutputPath>D:\Ecommerce\v1.0\Clients\Client1\Deploy
</DeployWebAppOutputPath>
    <DeployWebWorkingDirectory>C:\Windows\System32\inetsrv
</DeployWebWorkingDirectory>
  </PropertyGroup>
  <PropertyGroup>
    <FxCopInput>$(IntegrationPath)\$(ProjectName)\bin\$(ProjectName).dll
</FxCopInput>
    <FxCopOutput>$(ArtifactPath)\FxCop\FxCop.xml</FxCopOutput>
    <FxCopPath>C:\Program Files\Microsoft FxCop 1.36</FxCopPath>
    <FxCopProjectName>$(ProjectName).FxCop</FxCopProjectName>
    <FxCopProjectPath>$(IntegrationPath)\$(FxCopProjectName)</FxCopProjectPath>
  </PropertyGroup>
  <PropertyGroup>
    <MsTestInput>$(IntegrationPath)\TestProject\bin\Debug\TestProject.dll
</MsTestInput>
    <MsTestOutput>$(ArtifactPath)\MsTest\MsTest.xml</MsTestOutput>
    <MsTestPath>"C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE"
</MsTestPath>
  </PropertyGroup>
  <PropertyGroup>
    <NCoverAppToProfileArgs>"/testcontainer:$(MsTestInput)"</NCoverAppToProfileArgs>
    <NCoverAppToProfileExe>$(MsTestPath)\MSTest.exe</NCoverAppToProfileExe>
    <NCoverPath>"C:\Program Files\NCover"</NCoverPath>
    <NCoverInputPath>$(IntegrationPath)\$(ProjectName)\bin</NCoverInputPath>
    <NCoverOutput>$(ArtifactPath)\NCover\NCover.xml</NCoverOutput>
    <NCoverOutputLogFile>$(ArtifactPath)\NCover\NCover.log</NCoverOutputLogFile>
  </PropertyGroup>
  <PropertyGroup>
    <NDependInput>$(IntegrationPath)\$(ProjectName)\bin\$(ProjectName).dll
</NDependInput>
    <NDependOutputPath>$(ArtifactPath)\NDepend</NDependOutputPath>
    <NDependOutput>$(NDependOutputPath)\NDependReport.html</NDependOutput>
    <NDependPath>C:\Program Files\NDepend</NDependPath>
    <NDependProjectName>$(ProjectName).NDepend.xml</NDependProjectName>
    <NDependProjectPath>$(IntegrationPath)\$(NDependProjectName)
</NDependProjectPath>
  </PropertyGroup>
  <PropertyGroup>
    <NDocDocumenter>MSDN-CHM</NDocDocumenter>
    <NDocInput>$(IntegrationPath)\$(ProjectName)\bin\$(ProjectName).xml</NDocInput>
    <NDocOutputPath>$(ArtifactPath)\NDoc</NDocOutputPath>
    <NDocOutput>$(NDocOutputPath)\Documentation.chm</NDocOutput>
    <NDocPath>D:\Recursos\Investigacion\Doctorado\Software\NDoc2-Alpha3u</NDocPath>
    <NDocProjectName>$(ProjectName).NDoc</NDocProjectName>
    <NDocProjectPath>$(IntegrationPath)\$(NDocProjectName)</NDocProjectPath>
  </PropertyGroup>
  <PropertyGroup>
    <ArtifactPath>D:\Ecommerce\v1.0\Feedback</ArtifactPath>
    <UniqueSqlScriptsExecutorAssembly>D:\Recursos\Investigacion\Doctorado\Software
\MsBuild.Extends\UniqueSqlScriptsExecutor\bin\Release\UniqueSqlScriptsExecutor.dll
</UniqueSqlScriptsExecutorAssembly>
    <UniqueSqlScriptsExecutorInputPath>$(IntegrationPath)\$(ProjectName)\Scripts
</UniqueSqlScriptsExecutorInputPath>
    <UniqueSqlScriptsExecutorOutputPath>$(ArtifactPath)\Scripts
</UniqueSqlScriptsExecutorOutputPath>
    <WorkingDirectory>D:\Ecommerce\v1.0\Clients\Client1\Config</WorkingDirectory>
    <OrderFile>UniqueSqlScriptsExecutorTaskOrder.xml</OrderFile>
  </PropertyGroup>
  <!-- targets -->
  <Target Name="Begin">
    <Message Text="Initializing extra build tasks..." />
  </Target>
  <Target Name="DeployWebApp">
    <Exec WorkingDirectory="$(DeployWebWorkingDirectory)"
Command="appcmd stop site /site.name:$(ProjectName)" />
    <Exec WorkingDirectory="$(DeployWebAppInputPath)" Timeout="60000"
```

```
IgnoreExitCode="true" Command="robocopy.exe␣.␣$(DeployWebAppOutputPath)␣*.*
/E␣/XA:H␣/PURGE␣/XO␣/XD␣'.svn'␣/NDL␣/NC␣/NS␣/NP" />
    <Exec WorkingDirectory="$(DeployWebWorkingDirectory)"
Command="appcmd␣start␣site␣/site.name:$(ProjectName)" />
  </Target>
  <Target Name="End">
    <Message Text="Finishing␣extra␣build␣tasks..." />
  </Target>
  <Target Name="FxCop" Inputs="$(FxCopInput)" Outputs="$(FxCopOutput)">
    <Exec WorkingDirectory="$(FxCopPath)" Command="FxCopCmd.exe
/project:$(FxCopProjectPath)␣/out:$(FxCopOutput)" />
  </Target>
  <Target Name="MsTest" Inputs="$(MsTestInput)" Outputs="$(MsTestOutput)">
    <Delete Files="$(MsTestOutput)" />
    <Exec Command="$(MsTestPath)\MsTest.exe␣/testcontainer:$(MsTestInput)
/resultsfile:$(MsTestOutput)" />
  </Target>
  <Target Name="NCover" Inputs="$(MsTestInput)" Outputs="$(NCoverOutput)">
    <Exec Command="$(NCoverPath)\NCover.Console.exe␣//w␣$(NCoverInputPath)
//x␣$(NCoverOutput)␣//l␣$(NCoverOutputLogFile)␣$(NCoverAppToProfileExe)
$(NCoverAppToProfileArgs)" />
  </Target>
  <UsingTask AssemblyFile="$(NDependPath)\MSBuild\NDepend.Build.MSBuild.dll"
TaskName="NDependTask" />
  <Target Name="NDepend" Inputs="$(NDependInput)" Outputs="$(NDependOutput)">
    <NDependTask NDependConsoleExePath="$(NDependPath)"
ProjectFilePath="$(NDependProjectPath)" OutDir="$(NDependOutputPath)" />
  </Target>
  <Target Name="NDoc" Inputs="$(NDocInput)" Outputs="$(NDocOutput)">
    <Exec Command="$(NDocPath)\NDocConsole.exe␣-documenter=$(NDocDocumenter)
-project=$(NDocProjectPath)" />
  </Target>
  <Target Name="RunAll">
    <CallTarget Targets="Begin" />
    <!-- <CallTarget Targets="MsTest" ContinueOnError="true" />
    <CallTarget Targets="NDoc" ContinueOnError="true" />
    <CallTarget Targets="FxCop" ContinueOnError="true" />
    <CallTarget Targets="NDepend" ContinueOnError="true" />
    <CallTarget Targets="NCover" ContinueOnError="true" />-->
    <CallTarget Targets="UniqueSqlScriptsExecutor" ContinueOnError="true" />
    <!-- <CallTarget Targets="DeployWebApp" ContinueOnError="true" />-->
    <CallTarget Targets="End" />
  </Target>
  <!-- targets -->
  <UsingTask TaskName="UniqueSqlScriptsExecutorTask" AssemblyFile=
"$(UniqueSqlScriptsExecutorAssembly)" />
  <Target Name="UniqueSqlScriptsExecutor">
    <UniqueSqlScriptsExecutorTask DataBaseServer="$(DataBaseServer)"
DataBaseName="$(DataBaseName)" UniqueSqlScriptsExecutorInputPath=
"$(UniqueSqlScriptsExecutorInputPath)" UniqueSqlScriptsExecutorOutputPath=
"$(UniqueSqlScriptsExecutorOutputPath)" WorkingDirectory="$(WorkingDirectory)"
OrderFile="$(OrderFile)">
    </UniqueSqlScriptsExecutorTask>
  </Target>
</Project>
```

Código fuente 14.5: Archivo `build.csproj`

14.6. Conclusiones

Al comienzo de este trabajo existían diversas necesidades tecnológicas para realmente poder trabajar plenamente con el concepto de MDCI (Model-Driven Conti-

nuous Integration):

- Necesidad de elección de una iniciativa MDE.
- Necesidad de detección correcta de nuevas versiones en los VCSs.
- Necesidad de integración de MDE y CI.
- Necesidad de generación incremental de artefactos.

Sin embargo, se han ido solventando los problemas encontrados con el objetivo de crear una herramienta prototípica que cumpla con los propósitos de Model-Driven Continuous Integration. Gracias a ello, se han podido satisfacer los principales objetivos de la investigación:

1. Permitir que los desarrolladores apliquen integración continua durante el desarrollo.
2. Permitir que expertos en el dominio modifiquen una aplicación sin necesidad de personal técnico.

Para probar y mostrar las ventajas del uso de la herramienta MDCIP (Model-Driven Continuous Integration Prototype), en los próximos capítulos se detallan algunos de los ámbitos de aplicación en los que se podría utilizar, y se realizará una evaluación de la mejora ofrecida en el proceso de desarrollo de software. Finalmente, se definirán los elementos clave del proceso Model-Driven Continuous Integration, surgido en base al trabajo llevado a cabo para desarrollar la herramienta presentada en este capítulo.

CAPÍTULO 15

Especificación de ámbitos de aplicación para MDCI

En el capítulo anterior se ha detallado la estructura de MDCIP (Model-Driven Continuous Integration Prototype), surgida en base a la solución de los diferentes problemas planteados durante todo el trabajo. Sin embargo, para que el prototipo sea de utilidad, es necesario aplicarlo en diferentes dominios reales.

Es este capítulo se mostrarán algunos de los posibles ámbitos de aplicación del prototipo desarrollado, dando ejemplos representativos que muestran cómo el empleo de los conceptos subyacentes a MDCI (Model-Driven Continuous Integration) pueden ser beneficiosos para afrontar desarrollos de software.

* * * *

15.1. Ámbitos de aplicación verticales

Se refiere a aquellas aplicaciones que están definidas para un mercado estrechamente delimitado. A continuación se muestran algunos ejemplos representativos.

15.1.1. eGobierno

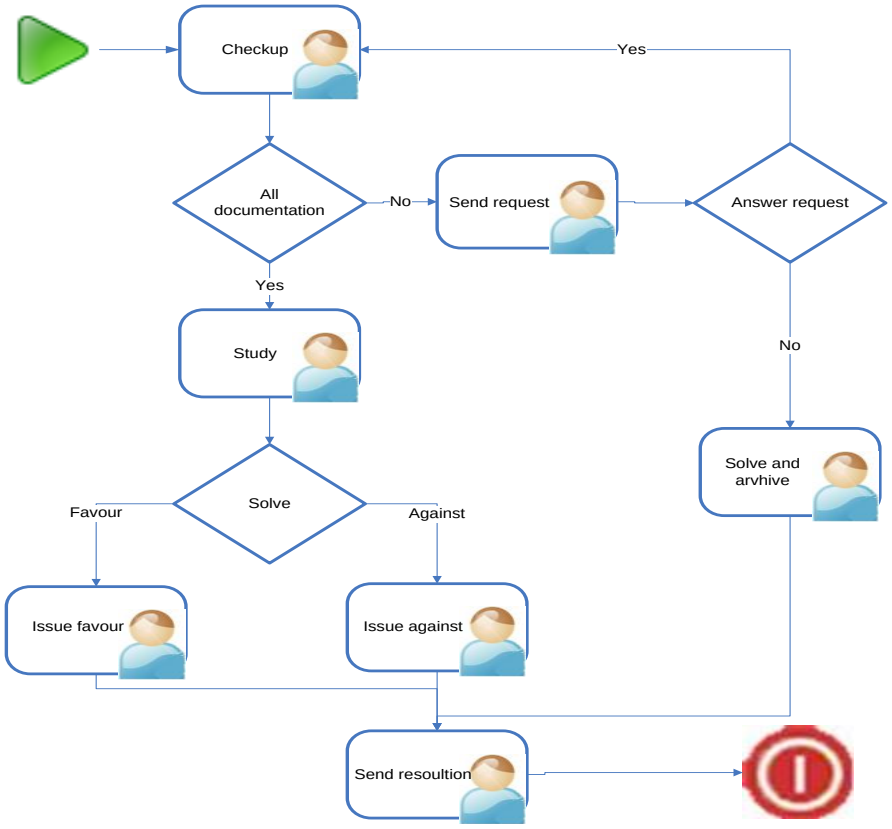


Figura 15.1: MDE. eGobierno. Ejemplo de DSL

En los últimos años, una de las mayores preocupaciones en el ámbito político es la presencia de diferentes servicios civiles en Internet con el objetivo de ahorrar costes en personal y el tiempo de los ciudadanos. Las aplicaciones Web son el tipo de aplicación más utilizada para este tipo de servicios porque son especialmente útiles para usuarios situados en diferentes localizaciones y para el empleo de diferentes plataformas informáticas.

Las aplicaciones para eGobierno no son especialmente diferentes de otras utilizadas en otros sectores, pero alguna características es especialmente crítica en este tipo de sistemas. Por ejemplo, la necesidad de un diseño centrado en el usuario, la

necesidad de realizar transacciones de forma segura y eficiente, el empleo de estándares [Alonso, 2008] o el cumplimiento de estrictos requisitos que frecuentemente pueden cambiar (p.e., cambios en la legislación). Algunos trabajos como [Mittal et al., 2004] han empezado a destacar la importancia de las aplicaciones para eGobierno. Sin embargo, el desarrollo de este tipo de sistemas aún no tiene un elevado nivel de abstracción.

Trabajos como [Figueiredo et al., 2004, Pontico et al., 2007] ya utilizan técnicas MDE en el desarrollo de sistemas para eGobierno. En [Palacios-González et al., 2009] se propone el empleo de herramientas MDE para generar, de forma completa y automática, este tipo de aplicaciones. El objetivo es que los expertos en el dominio puedan adaptar, cuando sea necesario, las aplicaciones a los cambiantes requisitos de las administraciones públicas. En la Fig. 15.1 se muestra un ejemplo de proceso definido con el lenguaje SBPMN (Simple Business Process Modeling Notation) [Fernández-Fernández et al., 2010], utilizado en la propuesta expuesta en [Palacios-González et al., 2009].

Una posibilidad con MDCI sería: Permitir que los empleados públicos sean capaces de modificar aplicaciones de eGobierno (p.e., por cambios que ocurran en la legislación vigente) sin necesidad de requerir la intervención de personal técnico que le ayude a desplegar los cambios realizados.

Además, al igual que podría ocurrir con los demás ámbitos de aplicación que se citarán en este capítulo, el equipo de desarrollo también podría emplear la práctica de integración continua durante el desarrollo de los productos de la línea de producción, aprovechándose de los beneficios de dicha práctica [Fowler, 2009a]

Se está trabajando para profundizar en este ámbito de aplicación - [Palacios-González et al., 2009]

15.1.2. Sistemas de gestión de aprendizaje

El aprendizaje electrónico (también llamado eAprendizaje o eLearning) está experimentando un gran auge debido a los progresos tecnológicos y a la aparición de herramientas como los LMSs (Learning Management Systems o Sistemas de Gestión de Aprendizaje) [Richard E and Clark, 2007], que facilitan las tareas tanto de los docentes como de los alumnos. Ofrecen varias ventajas, entre las que pueden citarse:

- Eliminación de barreras espaciales y temporales.

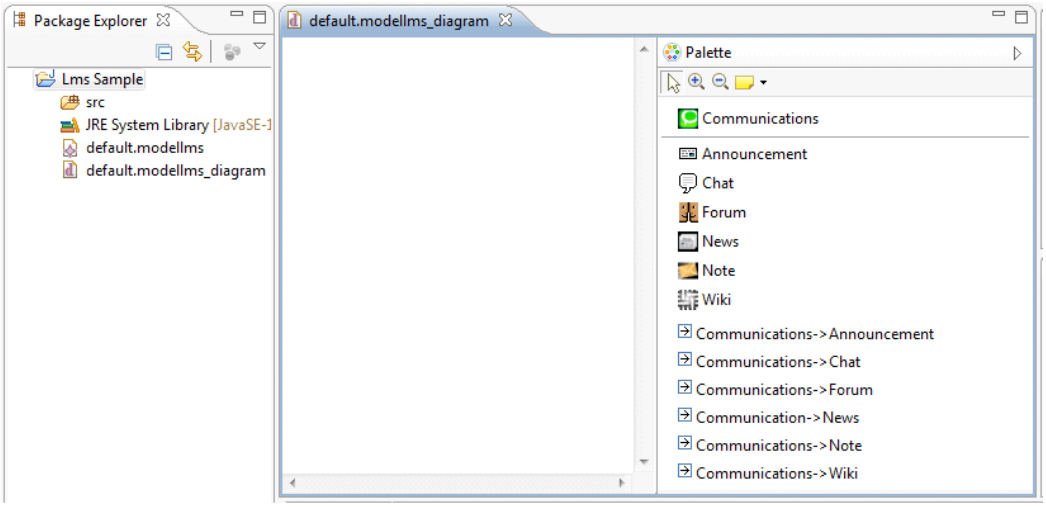


Figura 15.2: MDE. eAprendizaje. Aspecto del entorno de desarrollo

- Actualización constante de los contenidos.
- Reducción de costes logísticos.
- Prácticas en entornos simulados.
- Gestión real del conocimiento.

Los LMSs, que pueden ser vistos como un ámbito de aplicación vertical dentro de los sistemas de gestión de contenidos (véase sección 15.2.2), son los entornos diseñados para automatizar y gestionar el desarrollo de actividades formativas.

Actualmente, existe una gran cantidad de plataformas, tanto comerciales como de código abierto. Algunas de las más conocidas son Moodle[1], LRN[2], Blackboard[3], Ilias[4], Fronter[5] o eCollege[6]. Estos sistemas han de ser instalados y configurados como cualquier otro sistema de gestión de contenidos Web, requiriéndose la labor de administradores y docentes que configuren una serie de elementos que suelen ser comunes en todos los sistemas de gestión de aprendizaje (p.e., secciones de noticias, secciones teóricas, ejercicios, exámenes, salas de *chat*, foros). Como, a pesar de su

[1]http://www.moodle.org/
[2]http://www.dotlrn.org/
[3]http://www.blackboard.com/
[4]http://www.ilias.de/
[5]http://www.fronter.info/
[6]http://www.ecollege.com/

similitud, dependiendo de cada sistema concreto se utilizan diferentes tecnologías (lo que hace que en muchos casos personas sin conocimientos técnicos sean incapaces de configurar las plataformas), podría ser interesante una automatización de la gestión de dichos sistemas que sea independiente de cada tecnología subyacente.

Para lograr la automatización del trabajo con los LMSs, se podría crear un DSL en el que se incluyeran todas las opciones que pudieran ser modificadas por un administrador o docente en el futuro. Ya existen trabajos que relacionan MDE con el aprendizaje electrónico (p.e., [Iskander, 2007,Nodenot et al., 2008]). En [Montenegro-Marín et al., 2011] se propone un metamodelo, un DSL y un conjunto de transformaciones que permiten utilizar un entorno de desarrollo con un alto nivel de abstracción para crear LMSs. Dicho trabajo está enfocado a generar LMSs con el código fuente de Moodle, pero podría ser extendido para trabajar con otras plataformas de forma transparente para sus usuarios. En la Fig. 15.2 se muestra el aspecto del entorno de desarrollo presentado en [Montenegro-Marín et al., 2011].

Una posibilidad con MDCI sería: Permitir a los administradores de los LMSs gestionar de igual forma todos los cursos, independientemente de la tecnología subyacente, gracias a que la integración de los artefactos generados se haría de forma transparente para el personal no técnico.

Se está trabajando para profundizar en este ámbito de aplicación - [Montenegro-Marín et al., 2011]

15.1.3. Sistemas empotrados

Los sistemas empotrados son sistemas hardware/software de gran importancia en la actualidad [Vahid and Givargis, 2001]. A diferencia de las aplicaciones diseñadas y ejecutadas en ordenadores personales, los sistemas empotrados están formados por un hardware cuyo objetivo es desarrollar una determinada tarea, típicamente con restricciones de tiempo real, y por un software especialmente adaptado a dicho hardware. Algunos ejemplos se pueden encontrar en mecanismos de control, semáforos, electrodomésticos u otros dispositivos electrónicos de uso cotidiano. La gran cantidad de tipos de hardware y de software, así como la continua y rápida evolución de los mismos, hace que desarrollar y mantener este tipo de sistemas sea una tarea difícil y costosa. Además, los requisitos de tamaño y de coste del hardware de cada fabricante hace que, en muchas ocasiones, la única forma de desarrollar software sea utilizando lenguajes de bajo nivel de abstracción como C o C++. Ese es el motivo y la razón por la que hacen falta mecanismos que eleven el nivel de abstracción y

permitan una mejora de la productividad en un área tan competitiva [Hugues et al., 2007].

Existen ya multitud de trabajos cuyos objetivos están centrados en aplicar MDE al desarrollo del software de los sistemas empotrados. Por ejemplo, en [Oliver, 2003] se ofrece una visión orientada hacia un futuro en el que MDE, más concretamente MDA, será la clave en el desarrollo de sistemas embebidos. En [Bunse et al., 2009] se muestra como se aplica MDE para desarrollar diferentes variantes de un sistema orientado a la automoción, utilizando para ello un desarrollo basado en componentes. Además de otros muchos trabajos, también destaca el *Special issue on Model-driven Embedded System Design*[7], que refleja la importancia de MDE en el ámbito de este tipo de sistemas.

Una posibilidad con MDCI sería: Permitir que personas sin conocimientos de programación o sin conocimientos de lenguajes de bajo nivel de abstracción puedan modificar el comportamiento de un sistema empotrado sin ayuda de personal técnico especializado.

15.1.4. Trazabilidad alimentaria

Controlar el origen de los alimentos puede ofrecer una gran cantidad de beneficios [Golan et al., 2004], entre los que se encuentra el hecho de salvar vidas y ahorrar grandes cantidades de dinero. De hecho, están apareciendo nuevas leyes y reglamentos como el Artículo 18 de la Regulación Europea 178/2002[8], por el que desde el 1 de enero de 2005, todas las empresas europeas de la industria alimentaria tienen que tener implementado un sistema de trazabilidad. La trazabilidad alimentaria es, según el Artículo 3 de la Regulación Europea 178/2002, la habilidad de trazar y seguir un alimento, pienso, animal productor de alimentos o sustancias que serán incorporadas en cualquier alimento o pienso, a través de todas las etapas de su producción, procesamiento y distribución. Se podría decir que la trazabilidad alimentaria ha nacido para prevenir crisis como la de las vacas locas [Ratzan, 1998]. Los sistemas informáticos dedicados a gestionar la trazabilidad alimentaria son todavía muy recientes y carentes de una base que les permita automatizar su desarrollo eficaz y eficientemente.

En los trabajos [García-Díaz et al., 2008, García-Díaz et al., 2009a, García-Díaz et al., 2011] se detalla cómo es el primer generador automático de sistemas para gestionar la trazabilidad alimentaria. La idea es que, utilizando un DSL con un alto

[7] http://acmtecs.acm.org/mesd.htm
[8] http://ec.europa.eu/food/food/foodlaw/traceability/

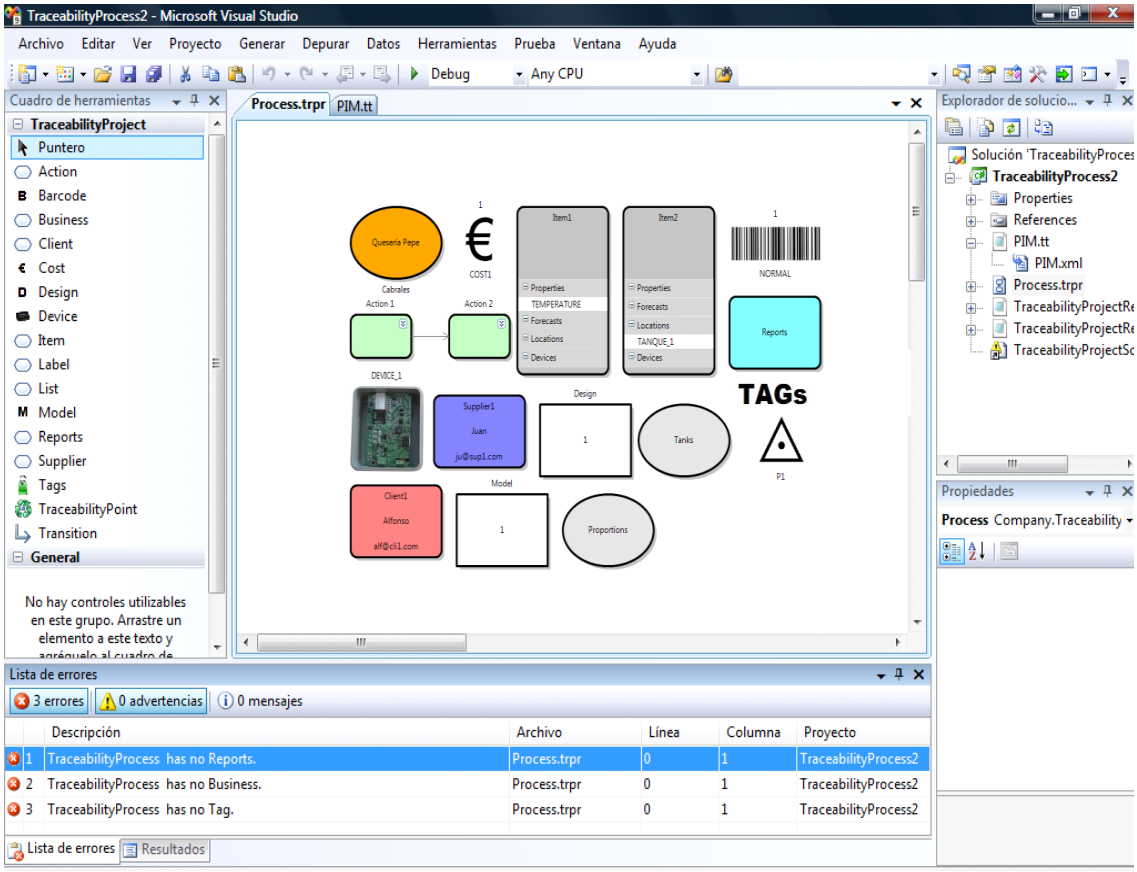


Figura 15.3: MDE. Trazabilidad alimentaria. Aspecto del entorno de desarrollo

nivel de abstracción, se puede lograr que los productores de alimentos sean capaces de definir cómo es el proceso de fabricación de su propia factoría. En la Fig. 15.3 se muestra el aspecto general del entorno de desarrollo del DSL y el anexo A muestra algunos procesos que podrían definirse con el DSL. De hecho, dichos procesos han sido elaborados mediante reuniones con los productores de alimentos durante el desarrollo de la herramienta.

Los trabajos actuales únicamente generan el código fuente del sistema específico de trazabilidad mediante el empleo del DSL. Así, por ejemplo en [García-Díaz et al., 2008], el código fuente obtenido, junto con los *scripts* SQL, tiene que ser desplegado manualmente en un servidor de aplicaciones. El despliegue no es una tarea sencilla porque una instalación de trazabilidad alimentaria estará compuesta por muy diverso hardware como etiquetadoras, terminales industriales, ordenadores, impresores, lectores RFID, etc.

Una posibilidad con MDCI sería: Permitir a los fabricantes de productos alimentarios modificar el proceso de fabricación, permitiendo variar la forma y el orden en el que las terminales industriales piden los datos a los trabajadores de la fábrica. Así, se logra que los analistas de negocio sean capaces, de forma transparente para ellos, de cambiar la lógica de negocio del software, pudiendo adaptar su negocio a nuevas necesidades de producción sin la necesidad de intervención de personal técnico informático.

Este ámbito de aplicación es el que ha motivado el trabajo expuesto en esta Tesis.

15.1.5. Videojuegos

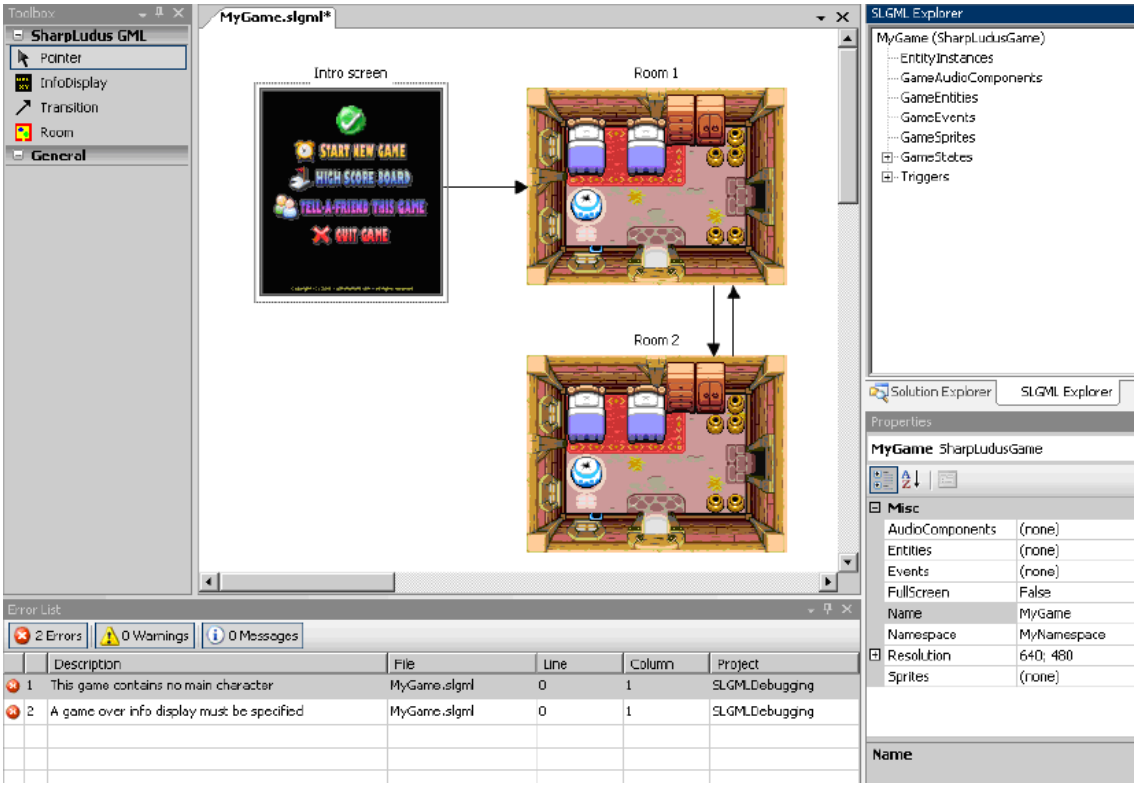


Figura 15.4: MDE. Videojuegos. Ejemplo de DSL

Los videojuegos son una de las mayores industrias del mundo [ESA, 2009], y es por ello que automatizar en la medida de lo posible su desarrollo puede aportar una gran ventaja frente a otros competidores. Sin embargo, a pesar de las muchas similitudes que se pueden encontrar en los aparentemente diferentes tipos de videojuegos [Djaouti et al., 2008], dicha industria aún carece de una forma aceptada que permita definir fácilmente nuevos productos.

Figura 15.5: MDE. Videojuegos. Ejemplos generados con el DSL

Por el anterior motivo, en [Furtado et al., 2007] se explica cómo han creado una pequeña línea de productos de videojuegos centrada en la plataforma .NET de Microsoft. Para ello, se emplea la iniciativa de las Software Factories [Greenfield and Short, 2003] y se utiliza un DSL de alto nivel de abstracción que permite desarrollar fácilmente videojuegos sin necesidad de conocimientos en lenguajes de programación tradicionales. Los juegos son aventuras gráficas en dos dimensiones, en las que como mínimo tiene que existir un personaje principal, un escenario, una pantalla de introducción y una pantalla de fin de juego. La Fig. 15.4 muestra el DSL utilizado y la Fig. 15.5 algunos ejemplos realizados con él. Otro interesante trabajo en esta línea es [Montero-Reyno and Cubel, 2009], que propone el diseño de un modelo para reemplazar el uso del lenguaje natural por parte de los diseñadores de videojuegos.

Los trabajos actuales son limitados en el sentido que únicamente generan el código fuente de cada juego mediante el empleo del entorno de desarrollo. Por ejemplo, en [Furtado et al., 2007] utilizan Visual Studio[9], obligando a que alguien del equipo técnico se encargue de crear un proyecto instalador con el código, con el objetivo de ser distribuido a cada máquina de cada potencial jugador.

[9]http://msdn.microsoft.com/en-us/vstudio/

Una posibilidad con MDCI sería: Permitir a cada jugador crear y actualizar sus propios juegos sin necesidad de tocar directamente el código fuente que se genera con la herramienta de desarrollo.

15.2. Ámbitos de aplicación horizontales

Se refiere a aplicaciones que proporcionan una solución general a todo un proceso o a un área de negocio específica. En las siguientes líneas se mencionan algunos posibles ejemplos.

15.2.1. Definición de procesos de negocio

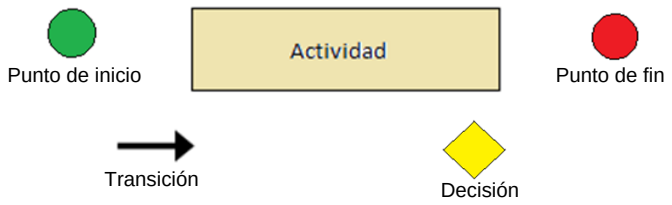


Figura 15.6: MDE. Procesos de negocio. Elementos del DSL

El BPM (Business Process Management o Modelado de Procesos de Negocio) se está convirtiendo en una práctica de uso creciente en el mundo empresarial e informático. Así, los expertos en un determinado proceso de negocio pueden definirlo de un modo formal utilizando un lenguaje informático, mientras permiten un mejor entendimiento por parte de personas que no están familiarizadas con dicho proceso.

Dada su creciente importancia, han ido apareciendo estándares y tecnologías como BPMN (Business Process Modeling Notation), XPDL (XML Process Definition Language), jPDL (jBPM Process Definition Language) o WWF (Windows Workflow Foundation) [Shukla and Schmidt, 2006]. A partir de un proceso de negocio definido utilizando alguna notación (p.e., el estándar BPMN o WWF), un equipo de desarrollo podría utilizar dicho proceso como motor para guiar la lógica de una aplicación. De ese modo, se puede ver a los lenguajes para modelar procesos de negocio como DSLs, que pueden ser personalizados con actividades específicas que dependan de cada dominio determinado. El inconveniente, desde el punto de vista de quien modela su negocio, es que necesita la colaboración de un equipo informático

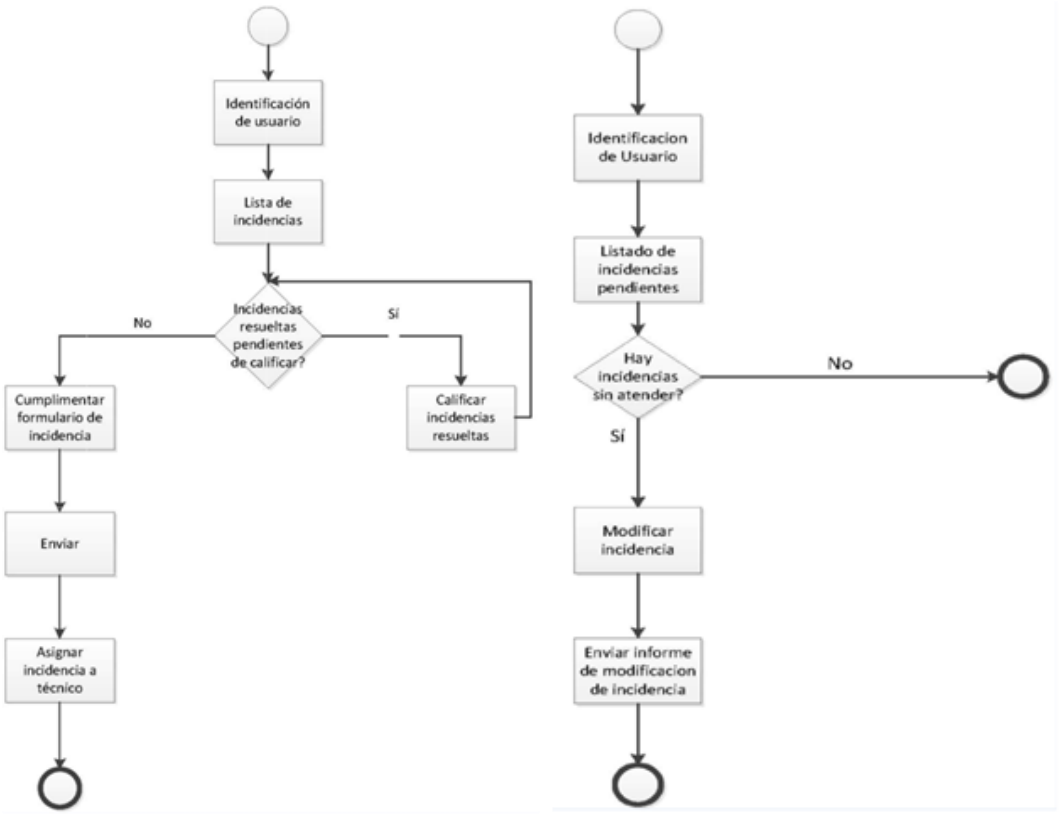


Figura 15.7: MDE. Procesos de negocio. Gestión de incidencias informáticas

que le ayude a integrar su proceso con el software y posteriormente desplegarlo en el entorno para el cual haya sido creado.

Existen muchos trabajos relacionados con la definición de procesos de negocio para diferentes dominios. Por ejemplo, en [Martínez et al., 2011b, Martínez et al., 2011a] se trabaja con los procesos de negocio de una empresa tecnológica para la que se ha creado un entorno de modelado que facilita la definición de dichos procesos mediante una notación muy simple. La Fig. 15.6 muestra los únicos elementos que tiene el DSL utilizado, mientras que la Fig. 15.7 muestra una definición preliminar en la que se modela el proceso de las incidencias informáticas (añadir incidencia y resolución de una incidencia). Dicho modelo podría ser utilizado como controlador para guiar la lógica de una aplicación informática de gestión de incidencias. Sin embargo y, al igual que ocurre en todos los demás posibles ámbitos de aplicación estudiados en este capítulo, la carencia de una unión entre la práctica de la integración continua y la ingeniería dirigida por modelos hace que no sea posible que los creadores de los modelos puedan trabajar de forma autónoma, sin la ayuda de

personal técnico.

Una posibilidad con MDCI sería: Permitir que los expertos en un dominio de conocimiento determinado modelen su proceso de negocio y modifiquen, de forma transparente para ellos, aplicaciones que utilicen dicho proceso modelado como motor para guiar la ejecución de la lógica del sistema informático.

Se está trabajando para profundizar en este ámbito de aplicación - [Martínez et al., 2011b, Martínez et al., 2011a]

15.2.2. Sistemas de gestión de contenidos

Los CMSs (Content Management Systems o Sistemas de Gestión de Contenidos) son aplicaciones informáticas, generalmente sitios Web, utilizadas para desarrollar una estructura común que sirve para crear y gestionar contenidos por parte de los diferentes usuarios y administradores involucrados. Normalmente existe una base de datos en la que se guarda todo el contenido del sitio, que es accedido a través de una interfaz amigable de cara al usuario final. Los CMSs tienen una estructura y características que les hacen ser similares incluso cuando se destinan para segmentos de mercado muy dispares. Por ese motivo, convendría una automatización de la gestión de dichos sistemas, de forma que independientemente de las tecnologías subyacentes (p.e., ASP.NET, PHP, MySQL, Oracle), pudieran ser gestionados de una forma común por parte de personal no técnico.

Para lograr la automatización del trabajo con los CMSs, se podría crear un DSL en el que se incluyeran todas las opciones que se estimen pueden ser modificadas por un administrador en el futuro. De hecho, ya han aparecido trabajos muy recientes como [de Sousa Saraiva and da Silva, 2009], en el que se implementan CMSs a partir de 2 lenguajes de alto nivel de abstracción o [Souer et al., 2009], en el que se utiliza MDE para configurar automáticamente CMSs. Johan den Haan[10], un experto en modelos, ha identificado similitudes entre los éxitos y los problemas de los CMSs y de MDE. Por todo ello, es muy probable que el futuro de los CMSs esté muy ligado a MDE.

[10]http://www.theenterprisearchitect.eu/archive/2010/01/26/what-model-driven-development-can-learn-from-content-management-systems/

Una posibilidad con MDCI sería: Permitir a los administradores de los CMSs gestionar de igual forma todos los sistemas, independientemente de la tecnología subyacente, gracias a que la integración de los artefactos generados se haría de forma transparente para el personal no técnico. Esta posibilidad es análoga a la propuesta para los LMSs, lo que podría permitir interpolar los resultados de las pruebas realizadas con los CMSs a los LMSs.

Se han realizado pruebas con un sistema de gestión de contenidos utilizando MDCIP (véase sección 16.2.1).

15.2.3. Sitios Web con estructura predefinida

Las aplicaciones Web han ido ganando importancia con el paso de los años. Las posibilidades colaborativas [Hsu and Lockwood, 1993] entre personas sin necesidad de utilizar máquinas de grandes presentaciones y su inmediatez lo han favorecido. Existen Webs sobre cualquier temática o dominio imaginable pero muchas de ellas tienen una estructura muy similar. Por ejemplo, los sitios de redes sociales, centros educativos, anuncios de congresos científicos, perfiles de usuario, etc. suelen tener una estructura muy similar entre ellos.

Sistemas como Google Sites[11] o SWS (Simple Website Software)[12] permiten crear aplicaciones Web sin necesidad de conocimientos informáticos, abriendo una puerta a que personas sin conocimientos de programación puedan diseñar y crear sus propios sitios Web de una forma bastante genérica. Sin embargo, si se busca una mayor personalización, aún habría que modificar manualmente el código HTML, CSS, Java, PHP o el de cualquiera de las tecnologías que se emplearan, siendo una tarea para la que muchas personas no están preparadas. El empleo de un DSL de alto nivel de abstracción en una capa superior a los sistemas como SWS, permitiría facilitar ciertos cambios lo máximo posible, evitando tener que programar en un lenguaje ajeno al del dominio de conocimiento del desarrollador. La importancia de MDE en la Web se ve reflejada con los innumerables trabajos de investigación realizados (p.e., The Model-Driven Web Engineering Workshop, ubicado en The International Conference On Web Engineering [Gaedke et al., 2009]).

[11] http://sites.google.com/
[12] http://phpsws.sourceforge.net/

Una posibilidad con MDCI sería: Permitir que los encargados de los sitios Web sean capaces de modificar el sitio, incluso concurrentemente si hay más de un encargado y, al mismo, tiempo desplegar los cambios en el servidor de forma transparente para ellos. Utilizando internamente lenguajes dinámicos, podrían modificar completamente sitios Web sin necesidad de saber programar en lenguajes tradicionales como HTML, CSS, SQL o PHP, únicamente mediante el empleo del DSL.

Se han realizado pruebas utilizando software para crear sitios Web con una estructura predefinida mediante el empleo de MDCIP (véase sección 16.2.2).

15.3. Ámbitos de aplicación transversales

Se refiere a software que por sus características no es en sí mismo ni una aplicación vertical, ni una aplicación horizontal. Por ejemplo, en esta categoría estarían las herramientas que únicamente generan el esqueleto de las clases de un lenguaje de programación o la generación automática de casos de prueba a partir de un modelo.

Generalmente, en este tipo de aplicaciones sólo se podría beneficiar, gracias a MDCI (Model-Driven Continuous Integration), el equipo de desarrollo software. En este caso, los usuarios finales, analistas o expertos en un dominio de conocimiento no tendrían cabida puesto que, en circunstancias normales, no tendrán acceso al DSL utilizado. Sin embargo, a continuación se muestra un caso de aplicación transversal en el que sí tendrían cabida.

15.3.1. Personalización de aplicaciones previamente desarrolladas

Existen millones de sistemas informáticos que continuamente evolucionan de múltiples formas [Chapin et al., 2001] o son sustituidos por otros con unas mayores prestaciones. Al mismo tiempo, las tecnologías utilizadas para desarrollar dichos sistemas también evolucionan y muy frecuentemente se ven superadas por otras que aparecen en el mercado.

Una posibilidad existente para permitir a terceras personas modificar aplicaciones desarrolladas previamente por otros, independientemente de la tecnología o arquitectura que se haya utilizado, es la creación de un DSL de alto nivel de abstracción entendible por las personas que podrían querer o necesitar modificar aspectos internos de las aplicaciones ya desarrolladas. Por ejemplo, se podrían querer modificar aspectos ya existentes (p.e., textos, logotipos, aspecto visual, archivos de

configuración) respetando la arquitectura básica desarrollada por los autores originales.

Una posibilidad con MDCI sería: Permitir que personas sin conocimientos de programación ni de la estructura interna de un determinado software, sean capaces de realizar modificaciones en él, con el mínimo impacto posible y sin necesidad de aprender ninguna de las tecnologías involucradas. Únicamente tendrían que utilizar un DSL especialmente diseñado para sus necesidades.

Se han realizado pruebas personalizando una aplicación desarrollada por terceros utilizando MDCIP (véase sección 16.2.3).

15.4. Conclusiones

El esfuerzo extra necesario para trabajar con modelos (p.e., definir un DSL o las transformaciones subyacentes) suele ser rentable cuando se trabaja con familias en las que se crean 3 o más productos [van der Linden et al., 2007] mediante el empleo de las posibilidades de variación definidas en el DSL. Así, tanto MDE en general, como MDCI (Model-Driven Continuous Integration) en particular pueden ser aplicados con éxito en infinidad de ámbitos.

En el próximo capítulo se realizará una evaluación de las mejoras ofrecidas por Model-Driven Continuous Integration mediante el desarrollo y empleo de algunos de los casos de aplicación citados en este capítulo.

CAPÍTULO 16

Evaluación de MDCI

Para realizar una evaluación de las potenciales ventajas obtenidas al trabajar con MDCI (Model-Driven Continuous Integration), se han realizado pruebas sobre el prototipo desarrollo a través del empleo de diferentes casos de estudio.

Las pruebas tienen por objeto demostrar que este proceso permite obtener mejores resultados en cuanto a los tiempos y recursos necesarios para desarrollar software utilizando la aproximación de desarrollo de la ingeniería dirigida por modelos. Cabe destacar que todas las pruebas se han realizado sobre aplicaciones reales de libre distribución y descargables desde Internet, respetando en todo caso su arquitectura. Es decir, las pruebas realizadas se han adaptado a las aplicaciones ya desarrolladas teniendo en cuenta las decisiones tomadas para su desarrollo, sean acertadas o no. Ese hecho permite una obtención de resultados más acordes con la realidad.

Este capítulo se destina a mostrar las pruebas realizadas, los resultados obtenidos y un análisis matemático que mostrará las ventajas de la propuesta realizada en esta Tesis.

⁎ ⁎ ⁎ ⁎

Técnica	Descripción
1	MDE sin integración continua
2	MDE con integración continua
3	MDE con integración continua y generación incremental de artefactos
4	MDE con integración continua y un sistema de control de versiones para modelos
5	MDCI. Es decir, MDE con integración continua, generación incremental de artefactos y un sistema de control de versiones para modelos

Tabla 16.1: Posibilidades de desarrollo con MDE

16.1. Presentación de las pruebas realizadas

Se han obtenido valores numéricos a partir de modificaciones en los modelos que guían la construcción o modificación de diverso software. Dichos parámetros se corresponden con:

1. El número de artefactos generados ante un cambio en un modelo.

2. El tamaño de dichos artefactos.

3. El tiempo de generación de los artefactos.

4. Su tiempo de despliegue en el entorno de producción.

La Tabla 16.1 representa las diferentes posibilidades de desarrollo utilizando MDE. Como sugieren algunos trabajos [Karlesky and Williams, 2007], actualmente una gran parte de desarrolladores de software ni siquiera conocen lo que es la integración continua y muchos de los que la conocen no la utilizan porque trabajar con las herramientas actuales suele ser bastante complejo. Así, los desarrollos actuales dirigidos por modelos se realizan con la técnica que la Tabla 16.1 refleja como técnica número 1. Sin embargo, y como se verá a continuación, el hecho de utilizar integración continua en MDE (técnica 2) es muy beneficioso para el desarrollo. La realización del prototipo del presente trabajo permite, además, evaluar las ventajas surgidas del empleo de las técnicas 3, 4 y, finalmente, la técnica 5 (Model-Driven Continuous Integration).

16.2. Datos obtenidos en las pruebas

A continuación se mostrarán los resultados obtenidos en los casos de prueba, utilizando las diferentes técnicas posibles.

16.2.1. Primer caso de prueba : The Beer House

Figura 16.1: Prueba 1. Aspecto general de *The Beer House*

Este caso de prueba se basa en una aplicación real denominada *The Beer House*[1], que es un completo *starter kit* para crear aplicaciones CMS (Content Management System) y de comercio electrónico. Dicha aplicación puede ser descargada y empleada libremente por quienes así lo deseen. Además, algunos libros [Bellinaso, 2006, Berardi et al., 2009, Love, 2009] utilizan a *The Beer House* como caso práctico a lo largo de su contenido. Por tanto, estamos ante una aplicación totalmente completa y funcional realizada con el framework ASP.NET MVC[2], utilizando el lenguaje C# y el sistema de gestión de bases de datos Sql Server. La Fig. 16.1 muestra el aspecto general de la página de inicio de una aplicación de ejemplo. El sistema ofrece características como las siguientes:

- Autenticación y autorización de usuarios.
- Noticias, artículos y *blogs*.

[1] http://www.codeplex.com/TheBeerHouse/
[2] http://www.asp.net/mvc/

- Encuestas.
- Boletines electrónicos.
- Foros.
- Comercio electrónico.
- Localización.

Para completar el caso de prueba, durante la integración continua, además de compilar, se realizan diversas acciones con el código de la aplicación: FxCop (análisis de ensamblados), MsTest (pruebas unitarias), NCover (cobertura de código), NDepend (optimización del código) y NDoc (documentación en función de los comentarios en el código).

Caso de prueba basado en el ámbito expuesto en la sección 15.2.2.

Lenguaje de dominio específico

Para el desarrollo, se ha construido un DSL especialmente pensado para realizar las tareas de configuración del CMS[3]. Así, en función de la entrada del usuario, se generarán diferentes archivos de configuración en formato XML y diferentes *scripts* SQL (Structured Query Language) que completarán la carga de datos de la aplicación en la base de datos correspondiente. Se trata así de un ejemplo muy típico de aplicación de MDE, en el que un usuario sin altos conocimientos de programación es capaz de programar en un determinado dominio mediante la utilización de un DSL determinado para sus necesidades.

La Fig. 16.2 muestra, por motivos de tamaño, únicamente un pequeño fragmento del metamodelo del DSL construido. Dicho metamodelo puede observarse con más detalle en las Figs. C.1, C.2, C.3 y C.4.

Acciones realizadas

Para la realización del caso de prueba número 1 se han hecho, en orden secuencial, una serie de modificaciones en el modelo de entrada utilizado por el generador de artefactos. Dichas modificaciones están reflejadas en la Tabla 16.2.

[3]El diseño de todos los DSLs utilizados en este capítulo se ve afectado por el carácter demostrativo de este trabajo, lo que implica que no pueden ser utilizados como ejemplo de las llamadas *mejores prácticas*

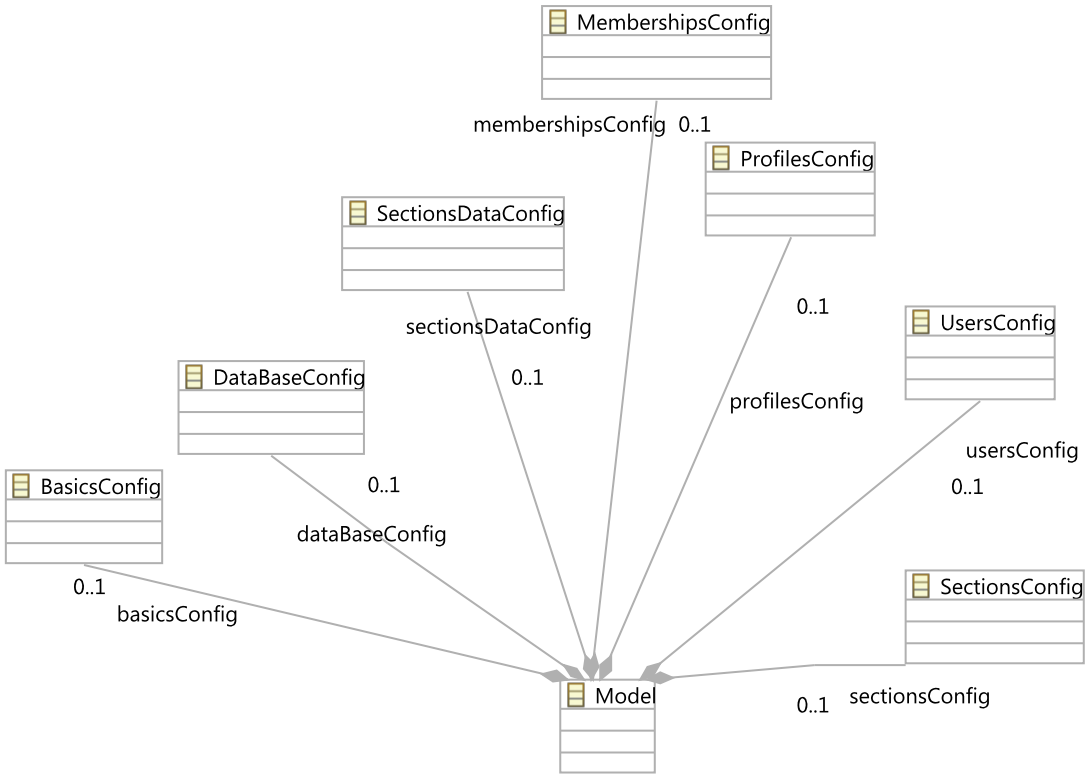


Figura 16.2: Prueba 1. Fragmento del metamodelo de CMSLanguage

Tabla de resultados

Las Figs. 16.3 y 16.4 muestran los resultados obtenidos en la evaluación del presente caso de prueba en función de la entrada. Las diferentes columnas representan:

1. La acción realizada según la Tabla 16.2.
2. La técnica empleada según la Tabla 16.1.
3. Si la unidad de versión es tal que se debiera detectar que se ha creado una nueva versión del modelo de entrada.
4. El número de artefactos generados.
5. El tamaño de la suma de los artefactos generados en bytes.
6. El tiempo de generación de los artefactos en milésimas de segundo.
7. El tiempo de despliegue de los artefactos generados en el entorno de producción en centésimas de segundo.

Número	Descripción
1	Generar todos los artefactos posibles a partir del modelo inicial
2	Añadir un usuario *newUser*
3	Cambiar el tamaño de los artículos y de las encuestas por página a 15
4	Añadir 3 nuevas categorías de elementos
5	Añadir el idioma italiano
6	Eliminar el idioma chino y el idioma francés
7	Crear una nueva encuesta
8	Añadir una nueva pregunta a la encuesta
9	Cambiar el nombre y el precio de los 2 tipos de envíos de productos
10	Cambiar el título, añadir un perfil y añadir un usuario
11	Cambiar el tamaño de página de los artículos, foros y encuestas
12	Cambiar toda la información de *Paypal* y del *newsletter*
13	Eliminar 3 categorías de elementos
14	Añadir un grupo de perfiles con 5 propiedades, añadir 3 profesiones y eliminar 2 departamentos
15	Modificar un elemento que no afecta a la generación de artefactos

Tabla 16.2: Prueba 1. Descripción de las acciones realizadas

N.	Técnica	UV	N.artefactos	Tam.artefactos	T.generación	T.despliegue
-	1	-	1090	29626368	16004	15566

Tabla 16.3: Prueba 1. Datos obtenidos sin CI

Hay que tener en cuenta que utilizando la técnica tradicional (MDE sin integración continua), no importa ni la acción realizada ni la configuración de la unidad de versión, ya que en todos los casos se volverán a regenerar todos los artefactos junto con su arquitectura básica. La Tabla 16.3 muestra el resultado obtenido, resultado medio que se reutiliza en las Figs. 16.3 y 16.4 para evitar pruebas redundantes.

Análisis de los resultados

Los datos mostrados en las Figs. 16.3 y 16.4 muestran los parámetros de salida (cuatro últimas columnas) obtenidos en función de las entradas (tres primeras columnas). Para poder interpretar los datos con una mayor facilidad, en las siguientes líneas se mostrarán diferentes gráficas con los valores resultantes.

El primer grupo de gráficas muestra el número de artefactos generados en función de los parámetros de entrada. La Fig. 16.5a muestra que cuando el sistema de control de versiones para modelos identifica que ha habido un cambio en el modelo, se regenerarán los artefactos pertinentes. En este caso, funcionarán mejor las técnicas 3 y 5 ya que incorporan la generación incremental de artefactos. Utilizando un sistema

N	Técnica	UC	Nº artefactos	Tamaño artefactos	T. generación (ms)	T. despliegue (cs)
1	1	1	1090	29626368	16004	15566
1	2	1	22	98304	465	4443
1	3	1	22	98304	471	4712
1	4	1	22	98304	581	4606
1	5	1	22	98304	567	5079
1	1	0	1090	29626368	16004	15566
1	2	0	22	98304	487	4578
1	3	0	22	98304	467	4039
1	4	0	0	0	0	0
1	5	0	0	0	0	0
2	1	1	1090	29626368	16004	15566
2	2	1	22	98304	533	4232
2	3	1	1	4096	378	121
2	4	1	22	98304	506	4897
2	5	1	1	4096	411	103
2	1	0	1090	29626368	16004	15566
2	2	0	22	98304	552	4190
2	3	0	1	4096	412	303
2	4	0	0	0	0	0
2	5	0	0	0	0	0
3	1	1	1090	29626368	16004	15566
3	2	1	22	98304	485	4488
3	3	1	2	8192	330	0
3	4	1	22	98304	472	4567
3	5	1	2	8192	353	0
3	1	0	1090	29626368	16004	15566
3	2	0	22	98304	489	4439
3	3	0	2	8192	327	0
3	4	0	0	0	0	0
3	5	0	0	0	0	0
4	1	1	1090	29626368	16004	15566
4	2	1	22	98304	469	4598
4	3	1	1	4096	407	156
4	4	1	22	98304	508	4761
4	5	1	1	4096	432	105
4	1	0	1090	29626368	16004	15566
4	2	0	22	98304	466	4081
4	3	0	1	4096	400	197
4	4	0	0	0	0	0
4	5	0	0	0	0	0
5	1	1	1090	29626368	16004	15566
5	2	1	22	98304	507	4138
5	3	1	1	4096	387	700
5	4	1	22	98304	467	4432
5	5	1	1	4096	390	409
5	1	0	1090	29626368	16004	15566
5	2	0	22	98304	467	4654
5	3	0	1	4096	452	229
5	4	0	0	0	0	0
5	5	0	0	0	0	0
6	1	1	1090	29626368	16004	15566
6	2	1	22	98304	476	4456
6	3	1	1	4096	382	242
6	4	1	22	98304	474	4543
6	5	1	1	4096	392	252
6	1	0	1090	29626368	16004	15566
6	2	0	22	98304	467	4491
6	3	0	1	4096	387	261
6	4	0	0	0	0	0
6	5	0	0	0	0	0
7	1	1	1090	29626368	16004	15566
7	2	1	22	98304	497	4767
7	3	1	1	4096	398	342
7	4	1	22	98304	556	4883
7	5	1	1	4096	394	327
7	1	0	1090	29626368	16004	15566
7	2	0	22	98304	515	4918
7	3	0	1	4096	380	332
7	4	0	0	0	0	0
7	5	0	0	0	0	0
8	1	1	1090	29626368	16004	15566
8	2	1	22	98304	550	4630
8	3	1	1	4096	396	269
8	4	1	22	98304	526	4424
8	5	1	1	4096	398	245

Figura 16.3: Prueba 1. Datos obtenidos 1/2

N	Técnica	UC	Nº artefactos	Tamaño artefactos	T. generación (ms)	T. despliegue (cs)
8	1	0	1090	29626368	16004	15566
8	2	0	22	98304	471	4877
8	3	0	1	4096	368	245
8	4	0	0	0	0	0
8	5	0	0	0	0	0
9	1	1	1090	29626368	16004	15566
9	2	1	22	98304	513	4107
9	3	1	1	4096	385	337
9	4	1	22	98304	480	4714
9	5	1	1	4096	387	314
9	1	0	1090	29626368	16004	15566
9	2	0	22	98304	488	4587
9	3	0	1	4096	391	322
9	4	0	0	0	0	0
9	5	0	0	0	0	0
10	1	1	1090	29626368	16004	15566
10	2	1	22	98304	472	4102
10	3	1	3	16384	436	679
10	4	1	22	98304	471	4458
10	5	1	3	16384	547	658
10	1	0	1090	29626368	16004	15566
10	2	0	22	98304	470	4398
10	3	0	3	16384	321	639
10	4	0	0	0	0	0
10	5	0	0	0	0	0
11	1	1	1090	29626368	16004	15566
11	2	1	22	98304	482	4612
11	3	1	3	12288	335	0
11	4	1	22	98304	471	4309
11	5	1	3	12288	336	0
11	1	0	1090	29626368	16004	15566
11	2	0	22	98304	472	4167
11	3	0	3	12288	337	0
11	4	0	0	0	0	0
11	5	0	0	0	0	0
12	1	1	1090	29626368	16004	15566
12	2	1	22	98304	502	4393
12	3	1	2	8192	340	0
12	4	1	22	98304	470	4103
12	5	1	2	8192	335	0
12	1	0	1090	29626368	16004	15566
12	2	0	22	98304	475	4883
12	3	0	2	8192	336	0
12	4	0	0	0	0	0
12	5	0	0	0	0	0
13	1	1	1090	29626368	16004	15566
13	2	1	22	98304	476	4676
13	3	1	1	4096	399	411
13	4	1	22	98304	475	4408
13	5	1	1	4096	396	407
13	1	0	1090	29626368	16004	15566
13	2	0	22	98304	535	4421
13	3	0	1	4096	392	405
13	4	0	0	0	0	0
13	5	0	0	0	0	0
14	1	1	1090	29626368	16004	15566
14	2	1	22	98304	534	5899
14	3	1	3	12288	402	540
14	4	1	22	98304	468	4901
14	5	1	3	12288	399	597
14	1	0	1090	29626368	16004	15566
14	2	0	22	98304	467	4511
14	3	0	3	12288	398	555
14	4	0	0	0	0	0
14	5	0	0	0	0	0
15	1	1	1090	29626368	16004	15566
15	2	1	22	98304	478	4864
15	3	1	0	0	356	0
15	4	1	22	98304	472	4676
15	5	1	0	0	328	0
15	1	0	1090	29626368	16004	15566
15	2	0	22	98304	485	4980
15	3	0	0	0	316	0
15	4	0	0	0	0	0
15	5	0	0	0	0	0

Figura 16.4: Prueba 1. Datos obtenidos 2/2

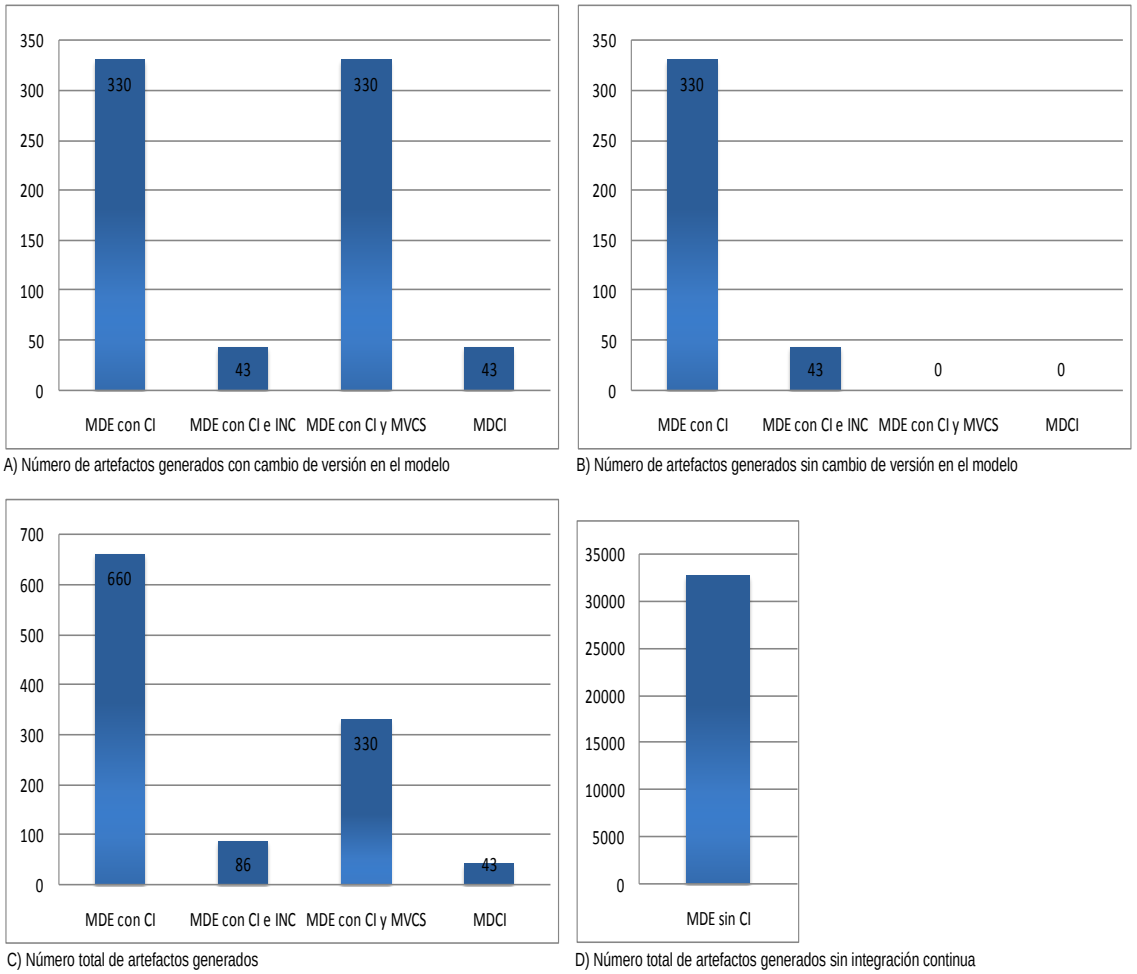


Figura 16.5: Prueba 1. Número de artefactos

de control de versiones para modelos (técnicas 4 y 5) y detectando que el cambio que ha hecho el usuario de la herramienta de modelo realmente no acarrea una nueva versión del software, se evita regenerar innecesariamente artefactos (Fig. 16.5b). Como la técnica 5 es la única que se comporta de la mejor forma posible en los dos casos anteriores, está claro que es con la que se han generado menos artefactos a lo largo de todas las acciones realizadas (Fig. 16.5c). La Fig. 16.5d muestra la cantidad de artefactos que se hubieran regenerado utilizando técnicas tradicionales en lugar de integración continua junto con MDE. La escala utilizada en dicha gráfica es mucho mayor que en los casos anteriores.

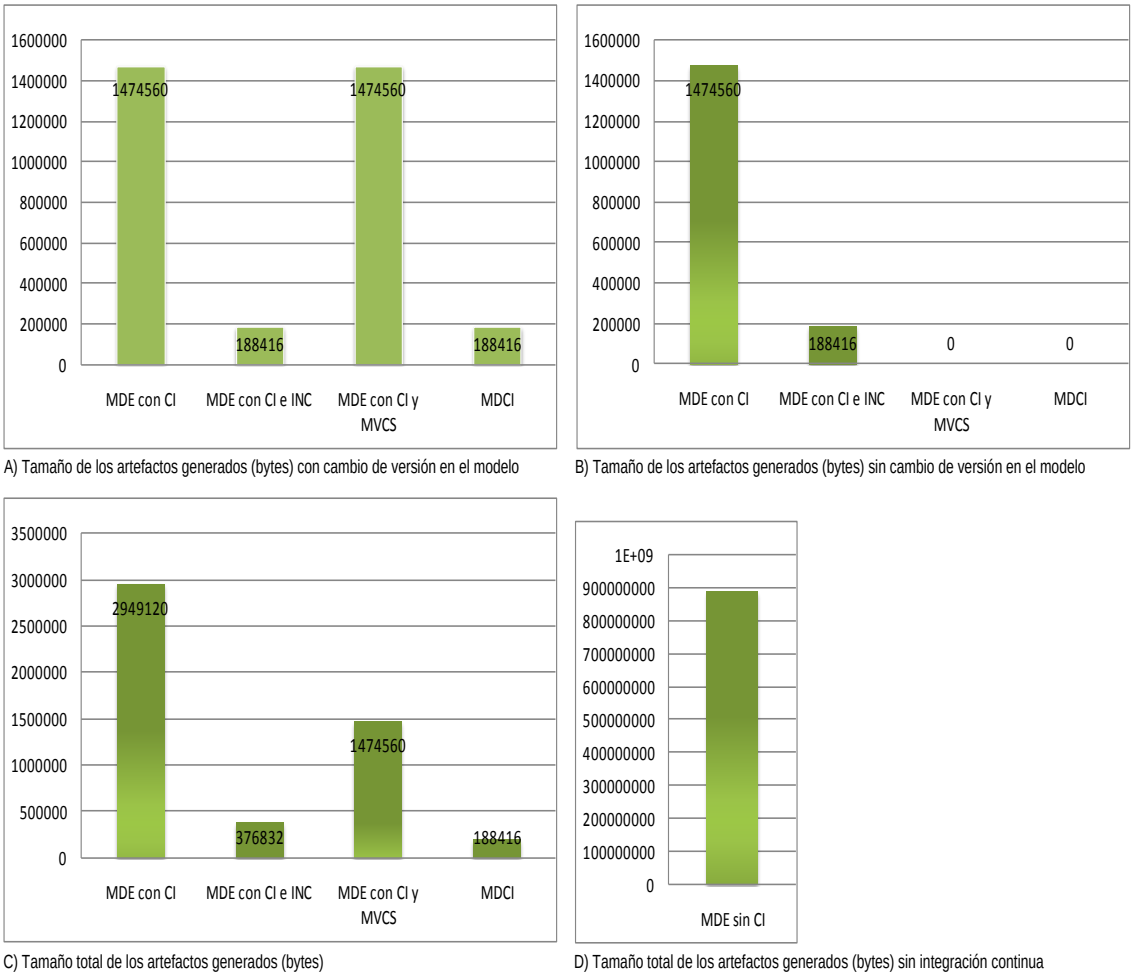


Figura 16.6: Prueba 1. Tamaño de los artefactos

El segundo grupo de gráficas muestra el tamaño total (en *bytes*) de los artefactos generados en función de la entrada. Por ello, la Fig. 16.6a muestra que cuando el sistema de control de versiones para modelos identifica que ha habido un cambio en el modelo, se regenerarán los artefactos necesarios para cada caso concreto. En este caso, funcionarán mejor las técnicas 3 y 5 ya que incorporan la generación incremental de artefactos. Como se puede observar, los resultados obtenidos en cuanto al tamaño de los artefactos en éste y en los demás casos (Figs. 16.6b, 16.6c y 16.6d), son directamente proporcionales al número de artefactos. Esto es así porque en este caso de prueba se ha trabajado siempre con archivos de similar tamaño.

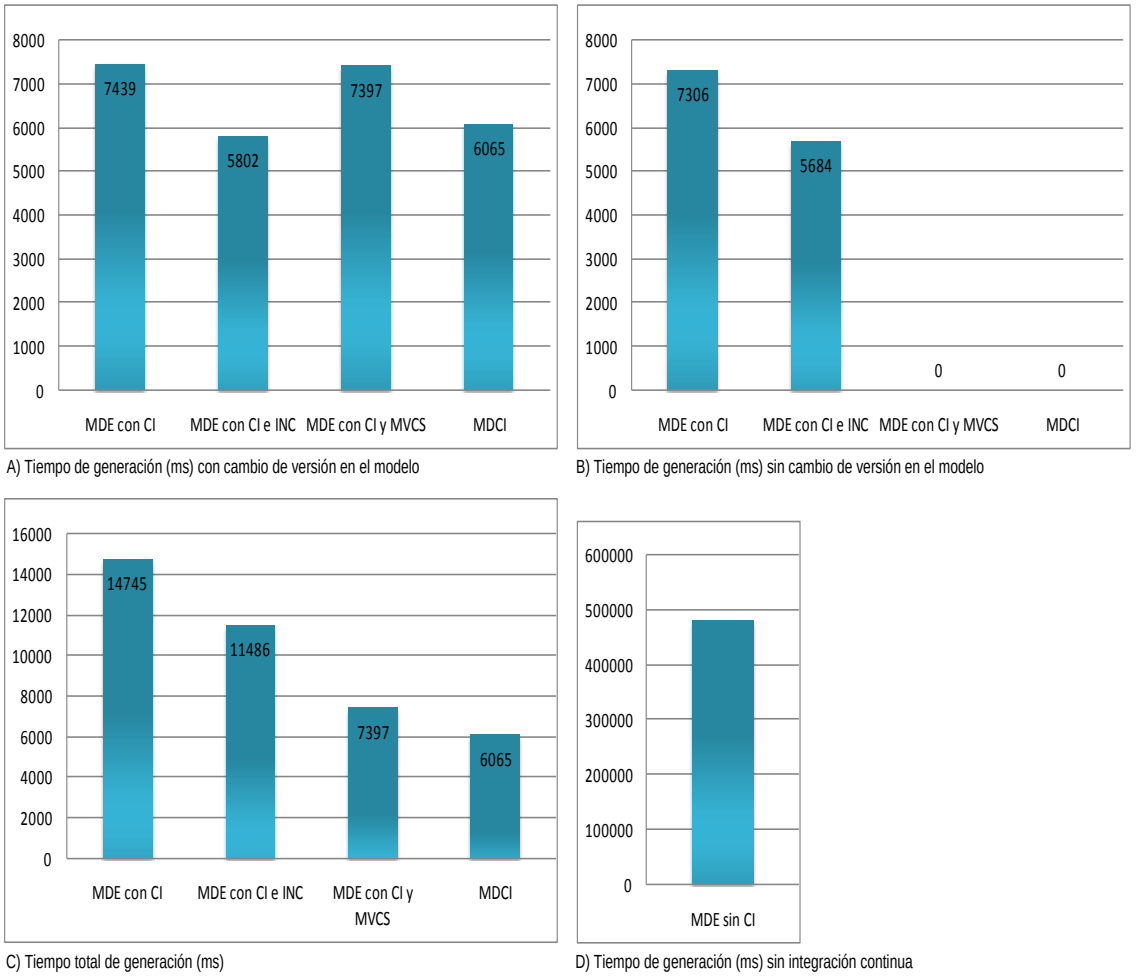


Figura 16.7: Prueba 1. Tiempo de generación

El tercer grupo de gráficas muestra el tiempo necesario (en milésimas de segundo) para generar artefactos cuando se manipulan los modelos de la entrada. Observando la Fig. 16.7a se puede ver que con este parámetro de salida los resultados son más igualados. Ello es debido a que con las técnicas previsiblemente mejores para este caso concreto en el que hay un cambio de versión en el modelo (técnicas 3 y 5), además de generar artefactos, habrá que realizar procesamiento extra debido al trabajo con el metamodelo extendido. También, se observa que para las técnicas 3 y 5 los resultados obtenidos, aunque muy similares, no son idénticos debido a que probabilísticamente es muy poco probable que diferentes ejecuciones en una máquina den exactamente los mismos tiempos. En las demás, (Figs. 16.7b, 16.7c y 16.7d) se pueden ver el resto de valores obtenidos.

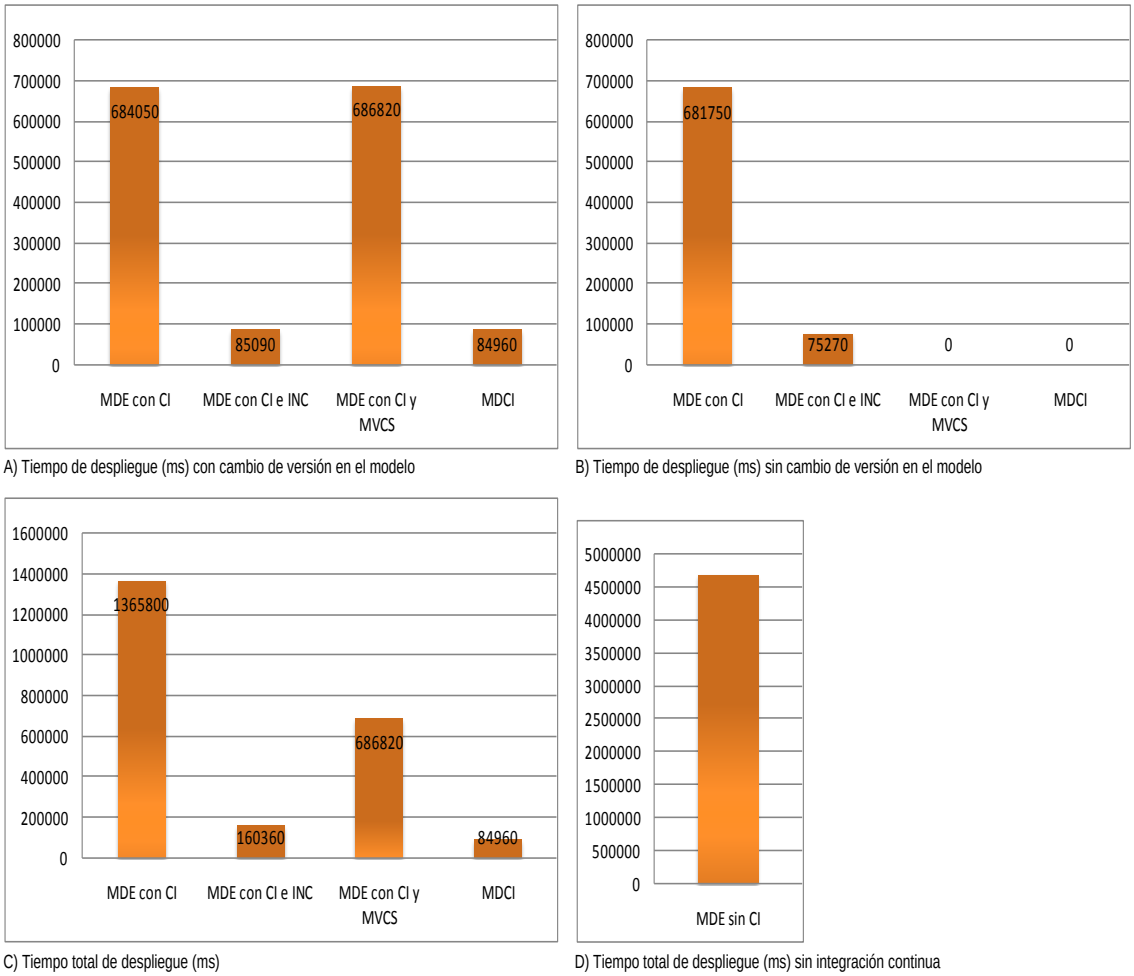


Figura 16.8: Prueba 1. Tiempo de despliegue

El cuarto grupo de gráficas muestra el tiempo necesario (en milésimas de segundo) para desplegar artefactos cuando se manipulan los modelos de la entrada. Con este parámetro de salida se han obtenido resultados muy proporcionales al número y al tamaño de los artefactos generados (en oposición al tiempo de generación que no era proporcional al número y al tamaño de los artefactos generados). Las Figs. 16.8a, 16.8b, 16.8c y 16.8d muestran los resultados obtenidos.

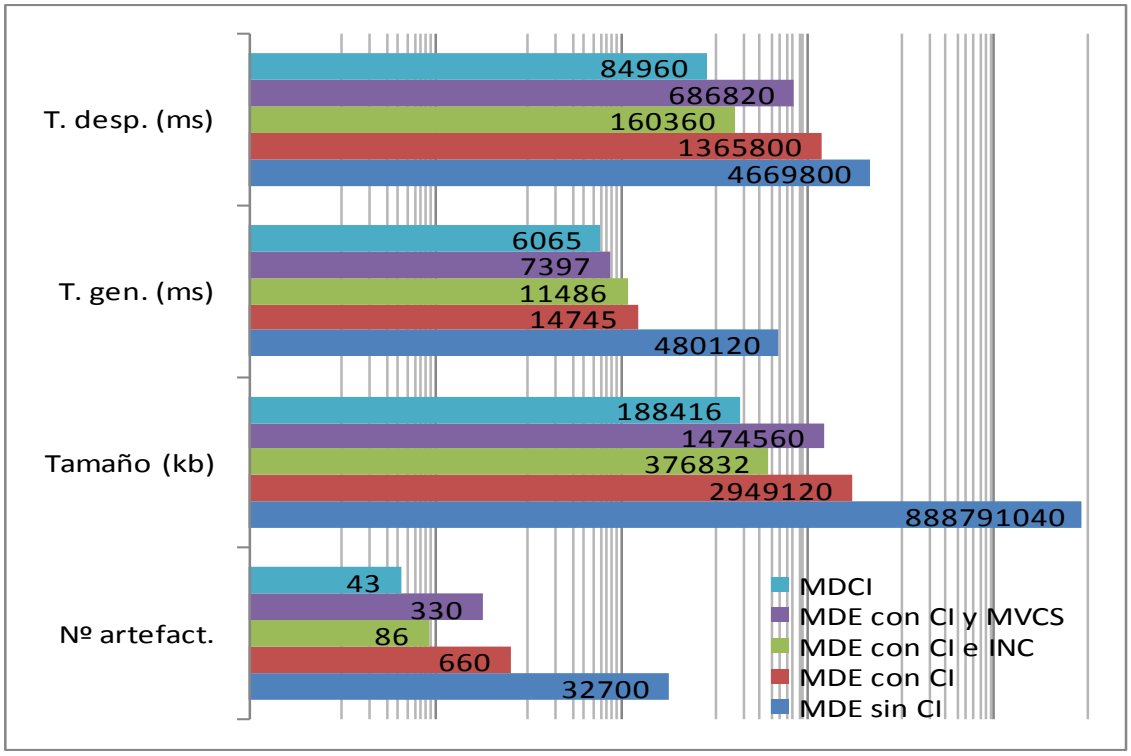


Figura 16.9: Prueba 1. Tabla logarítmica resumen de resultados obtenidos

Debido a que los datos obtenidos con la técnica 1 están en otra escala y a que estamos hablando de cuatro tipos de parámetros de salida con rangos de resultados muy diversos, la Fig. 16.9, que sirve para observar todos los resultados al mismo tiempo, está realizada en escala logarítmica. Se puede ver con claridad que la técnica 5 (Model-Driven Continuous Integration) es la mejor de todas en las acciones realizadas para los cuatro parámetros de salida. Del mismo modo, la técnica 1 (tradicional) es la que peor se comporta. Además, se puede observar que con incorporar únicamente la integración continua en el desarrollo, ya se consigue mejorar los resultados del mismo. El hecho de incorporar, además, un sistema de control de versiones específico de modelos o un sistema de generación incremental para modelos hace que los resultados también mejoren, siendo mejor una u otra herramienta según cada caso concreto de aplicación.

16.2.2. Segundo caso de prueba : Simple Website Software

Figura 16.10: Prueba 2. Aspecto general de *Simple Website Software*

Este caso de prueba se basa en una aplicación real denominada SWS (*Simple WebSite Software*)[4], cuyo objetivo es proporcionar lo necesario para generar aplicaciones Web con una pequeña cantidad de esfuerzo. SWS está realizado utilizando el lenguaje PHP[5] junto con páginas HTML y hojas de estilo CSS para controlar el *look and feel* del sitio. La Fig. 16.10 muestra el aspecto general de una de las numerosas Webs que han sido creadas utilizando SWS. Los componentes básicos que pueden ser utilizados con SWS son:

- Anuncios. Una caja opcional en la parte superior de la página.
- Menús de la izquierda. Enlaces obligatorios organizados en uno o varios bloques. Cada bloque tiene un título y consiste en cero o más enlaces.

[4]http://phpsws.sourceforge.net/
[5]http://www.php.net/

- Menús de la derecha. Enlaces opcionales.
- Cuerpo de la página. Contenido básico del sitio.
- Pie de página. El *copyright* y la fecha de cada página del sitio.

Durante la integración continua, se introducirán en un servidor Web los archivos generados en cada caso, acción que dará lugar al consumo de tiempo durante la etapa de despliegue. Hay que tener en cuenta que para alterar las mediciones lo menos posible, siempre se utilizará el mismo servidor con igual carga de trabajo.

Caso de prueba basado en el ámbito expuesto en la sección 15.2.3.

Lenguaje de dominio específico

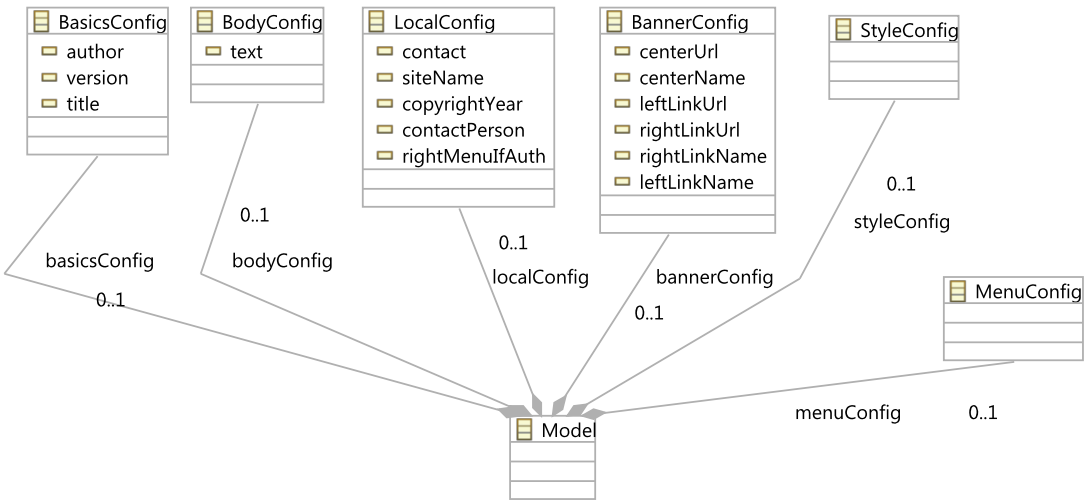


Figura 16.11: Prueba 2. Fragmento del metamodelo de SWSLanguage

SWS se utiliza para elevar el nivel de abstracción en el desarrollo de sitios Web dinámicos de pequeño tamaño debido a que es una herramienta que sirve para automatizar el desarrollo de sitios que siguen un patrón de diseño similar. Sin embargo, aún sería necesario que el editor del sitio tenga conocimientos de PHP, HTML y CSS para poder adaptar el sitio a sus preferencias personales y con los contenidos deseados. Para simplificar dicha tarea, se ha creado un DSL que permite elevar todavía más el nivel de abstracción, evitando tener que modificar manualmente los archivos creados con lenguajes de programación que pueden obligar a los editores a tener conocimientos excesivamente técnicos.

La Fig. 16.11 muestra únicamente un pequeño fragmento del metamodelo del DSL construido. Dicho metamodelo puede observarse con más detalle en las Figs. C.5 y C.6.

Acciones realizadas

Para la realización del caso de prueba número 2 se han hecho, en orden secuencial, una serie de modificaciones en el modelo de entrada utilizado por el generador de artefactos. Dichas modificaciones están reflejadas en la tabla 16.4.

Número	Descripción
1	Generar todos los artefactos posibles a partir del modelo inicial
2	Cambiar el nombre del autor
3	Cambiar el título y la versión del sito
4	Cambiar el texto del cuerpo del sitio por uno más largo
5	Eliminar el texto del cuerpo del sitio
6	Cambiar la fecha del *copyright*, la persona y la forma de contacto
7	Cambiar el nombre del sitio
8	Cambiar el nombre y el enlace del anuncio central
9	Cambiar el nombre y el enlace de los 3 anuncios del sitio
10	Cambiar el color del fondo y el color del texto del pie
11	Cambiar el tamaño del texto utilizado en el anuncio de información
12	Cambiar el título del primer bloque del menú de la izquierda
13	Añadir un bloque con 3 enlaces en el menú de la derecha
14	Eliminar el segundo bloque de enlaces del menú de la izquierda
15	Eliminar el primer bloque de enlaces del menú de la derecha, cambiar el color del texto principal del sitio y obligar a que el usuario esté registrado para que pueda visualizar el menú de la derecha

Tabla 16.4: Prueba 2. Descripción de las acciones realizadas

Tabla de resultados

Las Figs. 16.12 y 16.13 muestran los resultados obtenidos en la evaluación del presente caso de prueba en función de la entrada. Las diferentes columnas representan los mismos parámetros que para el primer caso de prueba, a excepción de la primera columna, que representa la acción realizada según la Tabla 16.4.

En el caso de prueba número 2, los valores medios utilizando únicamente MDE sin CI son mostrados en la Tabla 16.5.

N	Técnica	UC	Nº artefactos	Tamaño artefactos	T. generación (ms)	T. despliegue (cs)
1	1	1	11	33535	297	907
1	2	1	8	25556	345	630
1	3	1	8	25556	376	621
1	4	1	8	25556	373	642
1	5	1	8	25556	356	641
1	1	0	11	33535	297	907
1	2	0	8	25556	336	680
1	3	0	8	25556	334	677
1	4	0	0	0	0	0
1	5	0	0	0	0	0
2	1	1	11	33535	297	907
2	2	1	8	25556	337	643
2	3	1	2	19240	315	298
2	4	1	8	25556	339	656
2	5	1	2	19240	298	287
2	1	0	11	33535	297	907
2	2	0	8	25556	338	664
2	3	0	2	19240	324	296
2	4	0	0	0	0	0
2	5	0	0	0	0	0
3	1	1	11	33535	297	907
3	2	1	8	25556	373	645
3	3	1	2	19240	308	277
3	4	1	8	25556	372	698
3	5	1	2	19240	301	287
3	1	0	11	33535	297	907
3	2	0	8	25556	347	677
3	3	0	2	19240	302	260
3	4	0	0	0	0	0
3	5	0	0	0	0	0
4	1	1	11	33535	297	907
4	2	1	8	25556	344	663
4	3	1	1	103	320	61
4	4	1	8	25556	346	662
4	5	1	1	103	310	60
4	1	0	11	33535	297	907
4	2	0	8	25556	342	665
4	3	0	1	103	337	57
4	4	0	0	0	0	0
4	5	0	0	0	0	0
5	1	1	11	33535	297	907
5	2	1	8	25556	367	629
5	3	1	1	1	301	45
5	4	1	8	25556	362	612
5	5	1	1	1	299	46
5	1	0	11	33535	297	907
5	2	0	8	25556	336	670
5	3	0	1	1	299	45
5	4	0	0	0	0	0
5	5	0	0	0	0	0
6	1	1	11	33535	297	907
6	2	1	8	25556	341	667
6	3	1	1	2056	312	179
6	4	1	8	25556	356	666
6	5	1	1	2056	307	182
6	1	0	11	33535	297	907
6	2	0	8	25556	358	654
6	3	0	1	2056	316	190
6	4	0	0	0	0	0
6	5	0	0	0	0	0
7	1	1	11	33535	297	907
7	2	1	8	25556	393	680
7	3	1	1	2056	312	167
7	4	1	8	25556	354	678
7	5	1	1	2056	349	156
7	1	0	11	33535	297	907
7	2	0	8	25556	369	662
7	3	0	1	2056	308	171
7	4	0	0	0	0	0
7	5	0	0	0	0	0
8	1	1	11	33535	297	907
8	2	1	8	25556	370	661
8	3	1	1	411	311	71
8	4	1	8	25556	367	659
8	5	1	1	411	306	72

Figura 16.12: Prueba 2. Datos obtenidos 1/2

N	Técnica	UC	Nº artefactos	Tamaño artefactos	T. generación (ms)	T. despliegue (cs)
8	1	0	11	33535	297	907
8	2	0	8	25556	358	653
8	3	0	1	411	301	69
8	4	0	0	0	0	0
8	5	0	0	0	0	0
9	1	1	11	33535	297	907
9	2	1	8	25556	339	662
9	3	1	1	425	302	71
9	4	1	8	25556	338	637
9	5	1	1	425	298	73
9	1	0	11	33535	297	907
9	2	0	8	25556	338	649
9	3	0	1	425	300	74
9	4	0	0	0	0	0
9	5	0	0	0	0	0
10	1	1	11	33535	297	907
10	2	1	8	25556	344	666
10	3	1	1	2126	303	201
10	4	1	8	25556	343	670
10	5	1	1	2126	304	198
10	1	0	11	33535	297	907
10	2	0	8	25556	340	669
10	3	0	1	2126	303	188
10	4	0	0	0	0	0
10	5	0	0	0	0	0
11	1	1	11	33535	297	907
11	2	1	8	25556	377	640
11	3	1	1	425	309	74
11	4	1	8	25556	371	648
11	5	1	1	425	300	72
11	1	0	11	33535	297	907
11	2	0	8	25556	338	609
11	3	0	1	425	299	69
11	4	0	0	0	0	0
11	5	0	0	0	0	0
12	1	1	11	33535	297	907
12	2	1	8	25556	379	690
12	3	1	1	927	315	101
12	4	1	8	25556	337	687
12	5	1	1	927	316	120
12	1	0	11	33535	297	907
12	2	0	8	25556	376	658
12	3	0	1	927	319	132
12	4	0	0	0	0	0
12	5	0	0	0	0	0
13	1	1	11	33535	297	907
13	2	1	8	25556	352	663
13	3	1	1	1111	317	132
13	4	1	8	25556	351	629
13	5	1	1	1111	327	140
13	1	0	11	33535	297	907
13	2	0	8	25556	361	620
13	3	0	1	1111	320	138
13	4	0	0	0	0	0
13	5	0	0	0	0	0
14	1	1	11	33535	297	907
14	2	1	8	25556	359	646
14	3	1	1	734	330	99
14	4	1	8	25556	371	686
14	5	1	1	734	323	97
14	1	0	11	33535	297	907
14	2	0	8	25556	340	669
14	3	0	1	734	218	95
14	4	0	0	0	0	0
14	5	0	0	0	0	0
15	1	1	11	33535	297	907
15	2	1	8	25556	345	664
15	3	1	3	5066	328	322
15	4	1	8	25556	373	657
15	5	1	3	5066	332	298
15	1	0	11	33535	297	907
15	2	0	8	25556	346	673
15	3	0	3	5066	329	301
15	4	0	0	0	0	0
15	5	0	0	0	0	0

Figura 16.13: Prueba 2. Datos obtenidos 2/2

N.	Técnica	UV	N.artefactos	Tam.artefactos	T.generación	T.despliegue
-	1	-	11	33535	297	907

Tabla 16.5: Prueba 2. Datos obtenidos sin CI

Análisis de los resultados

Los datos mostrados en las Figs. 16.12 y 16.13 muestran los parámetros de salida (cuatro últimas columnas) obtenidos en función de las entradas (tres primeras columnas). Para poder interpretar los datos con una mayor facilidad, en las siguientes líneas se mostrarán diferentes gráficas con los valores resultantes.

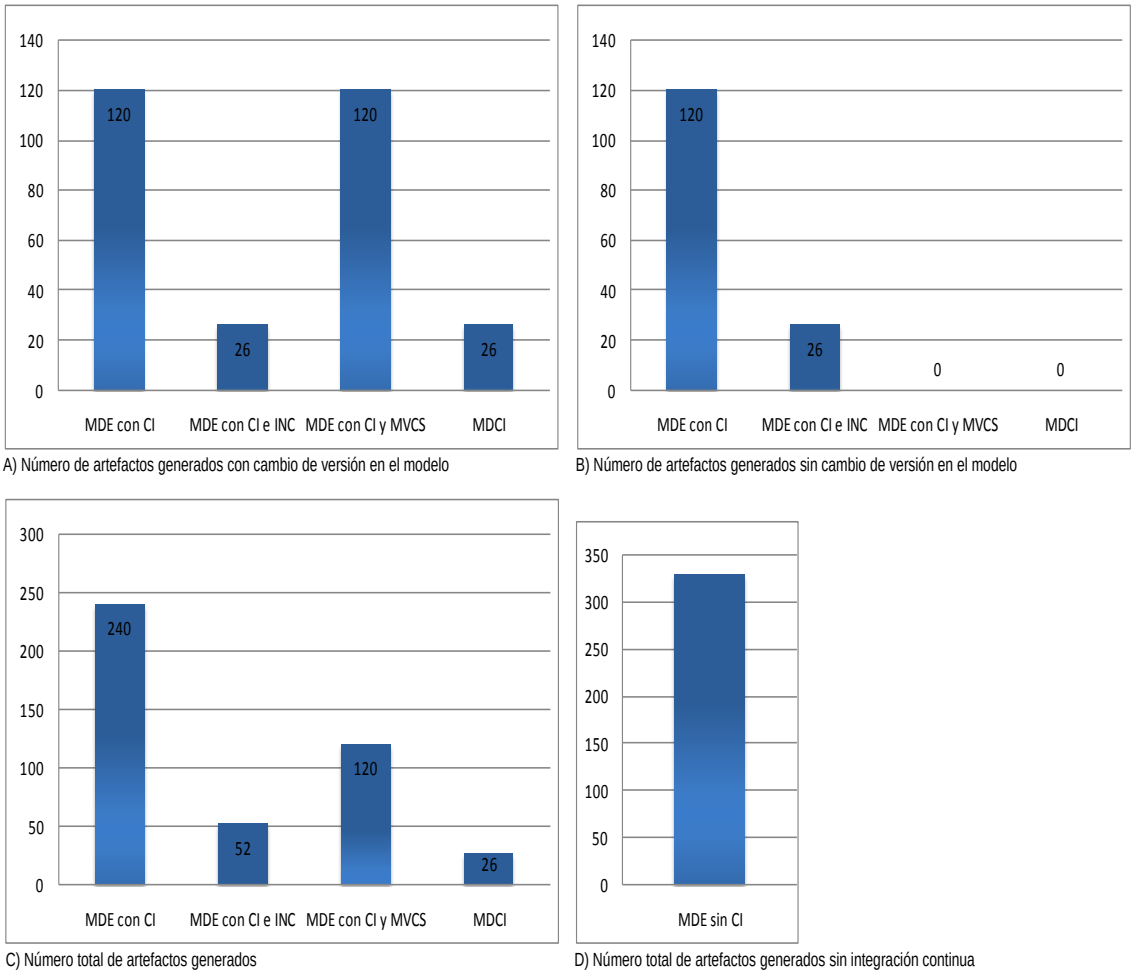


Figura 16.14: Prueba 2. Número de artefactos

El primer grupo de gráficas muestra el número de artefactos generados en función de los parámetros de entrada. Se observa que las Figs. 16.14a, 16.14b, 16.14c y 16.14d muestran resultados proporcionales a los obtenidos en el primer caso de prueba (véase sección 16.2.1). La única diferencia a efectos prácticos es el cambio de escala debido a las diferencias intrínsecas entre un caso de prueba y otro.

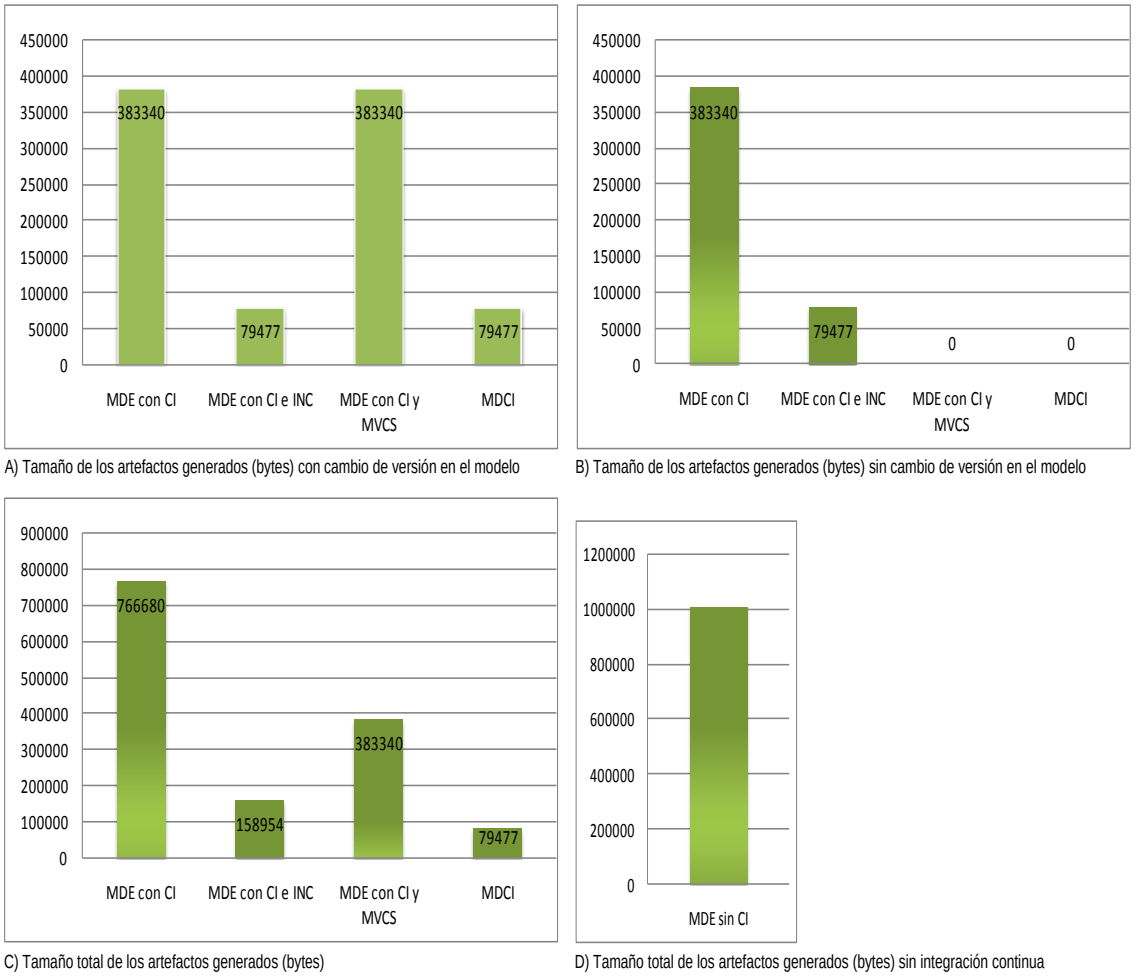


Figura 16.15: Prueba 2. Tamaño de los artefactos

El segundo grupo de gráficas muestra el tamaño total (en *bytes*) de los artefactos generados en función de la entrada. Las Figs. 16.15a. 16.15b, 16.15c y 16.15d muestran los resultados obtenidos, análogos a los del primer caso de prueba (véase sección 16.2.1).

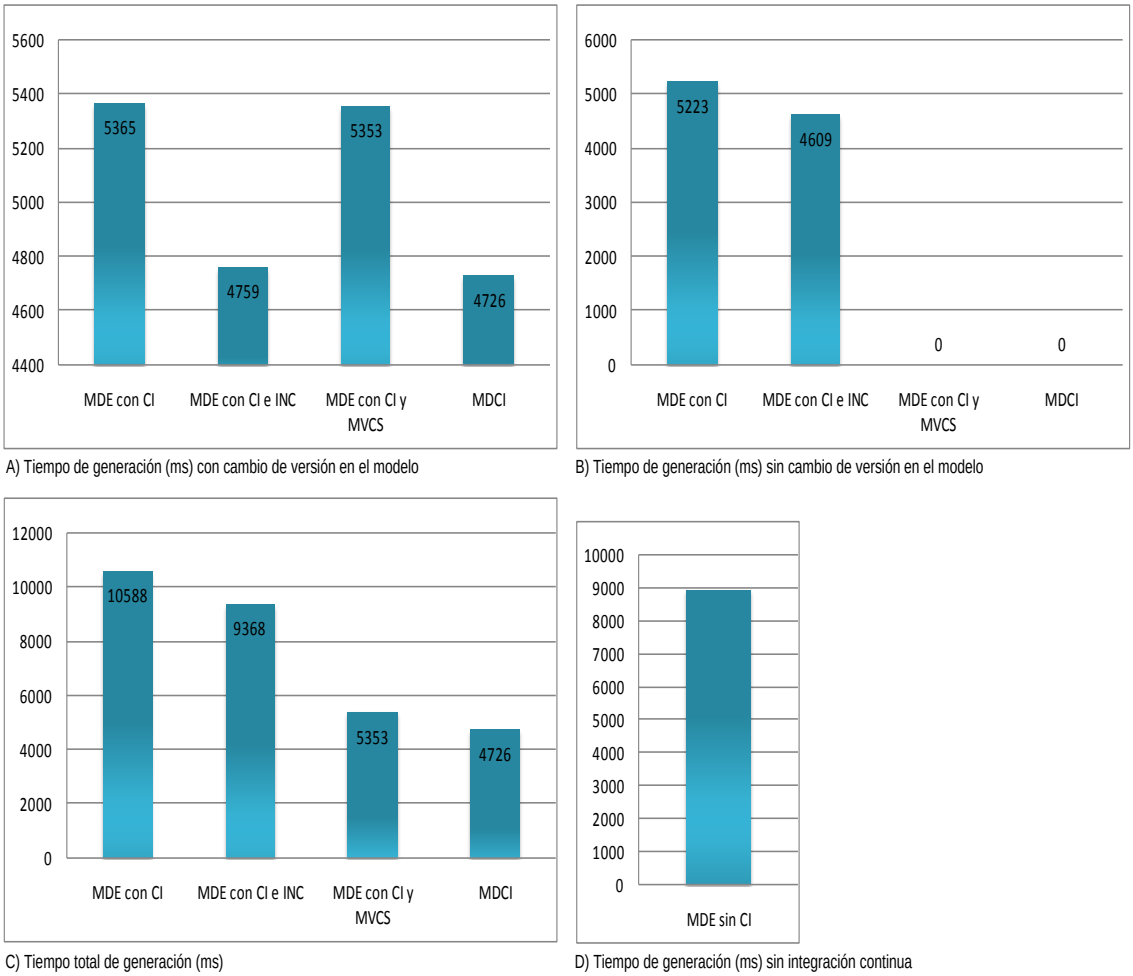


Figura 16.16: Prueba 2. Tiempo de generación

El tercer grupo de gráficas muestra el tiempo necesario (en milésimas de segundo) para generar artefactos cuando se manipulan los modelos de la entrada. Las Figs. 16.16a. 16.16b, 16.16c y 16.16d muestran los resultados obtenidos. En este caso existe alguna diferencia respecto al primer caso de prueba (véase sección 16.2.1). Por una parte, en la Fig. 16.16a se observa que con la técnica 3 los tiempos de generación son mayores que con la técnica 5 (debido únicamente a pequeñas diferencias que pueden considerarse aleatorias entre generaciones). Por otra parte, el tiempo de generación utilizando la técnica 1 es menor que el tiempo de generación utilizando las técnicas 2 y 3. Ello es debido a: 1- factores aleatorios, 2- a que cuando se genera el sistema final sin CI, únicamente hay que incorporar 3 archivos extra (lo cual consume muy poco tiempo - en el primer caso eran más de 1000) y 3- a que se evita el procesamiento extra para trabajar con el metamodelo extendido.

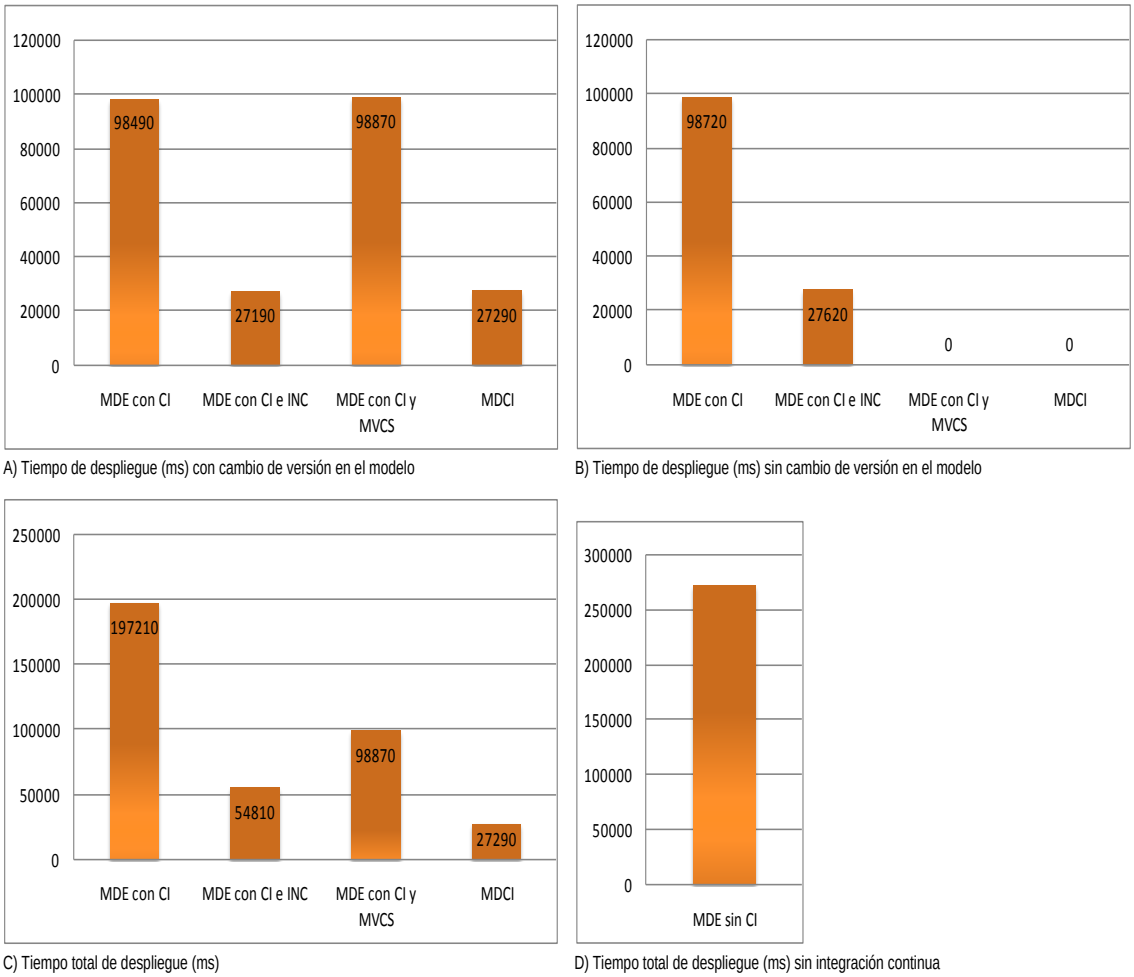


Figura 16.17: Prueba 2. Tiempo de despliegue

El cuarto grupo de gráficas muestra el tiempo necesario (en milésimas de segundo) para desplegar artefactos cuando se manipulan los modelos de la entrada. Las Figs. 16.17a. 16.17b, 16.17c y 16.17d muestran los resultados obtenidos, de nuevo análogos a los del primer caso de prueba (véase sección 16.2.1).

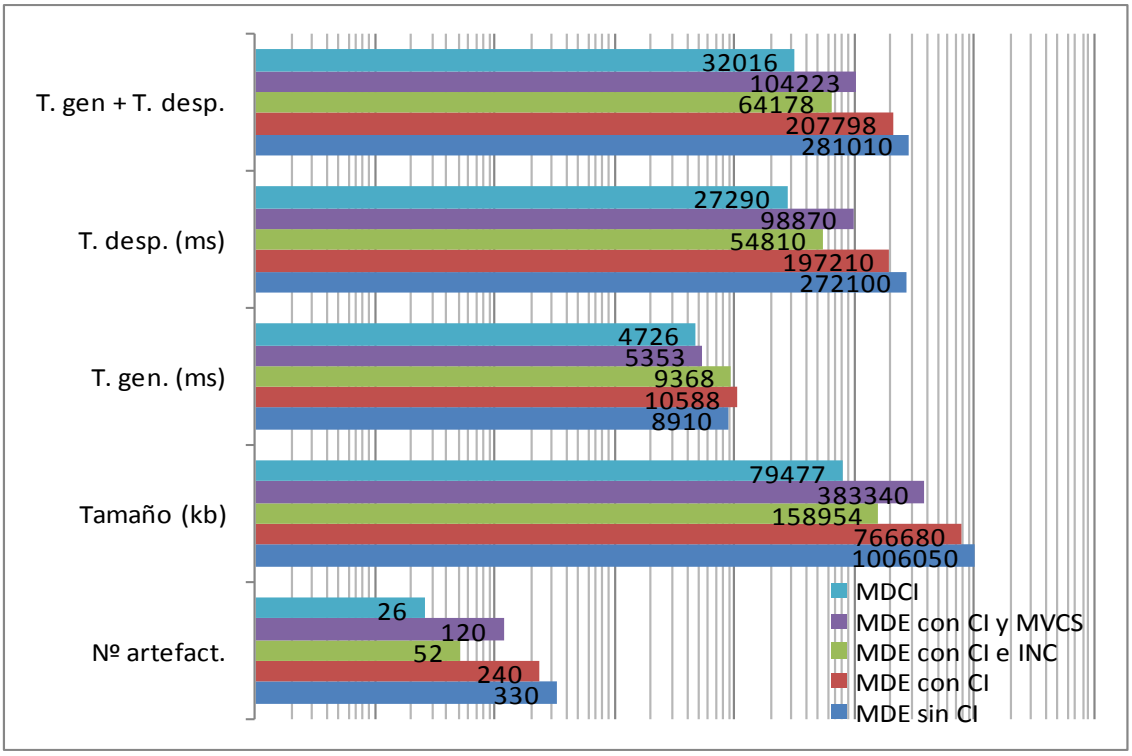


Figura 16.18: Prueba 2. Tabla logarítmica resumen de resultados obtenidos

Se observa en la Fig. 16.18 que la técnica 5 (Model-Driven Continuous Integration) es la mejor de todas en las acciones realizadas para los cuatro parámetros de salida. Del mismo modo, la técnica 1 (tradicional) es la que peor se comporta (a excepción puntualmente del tiempo de generación). Sin embargo, en este caso de prueba se ha añadido una nueva fila a la gráfica (en la parte superior) para ilustrar que sumando el tiempo de generación y el tiempo de despliegue, la técnica 1 es la que nuevamente peor se comporta de todas.

16.2.3. Tercer caso de prueba : Tweeterati

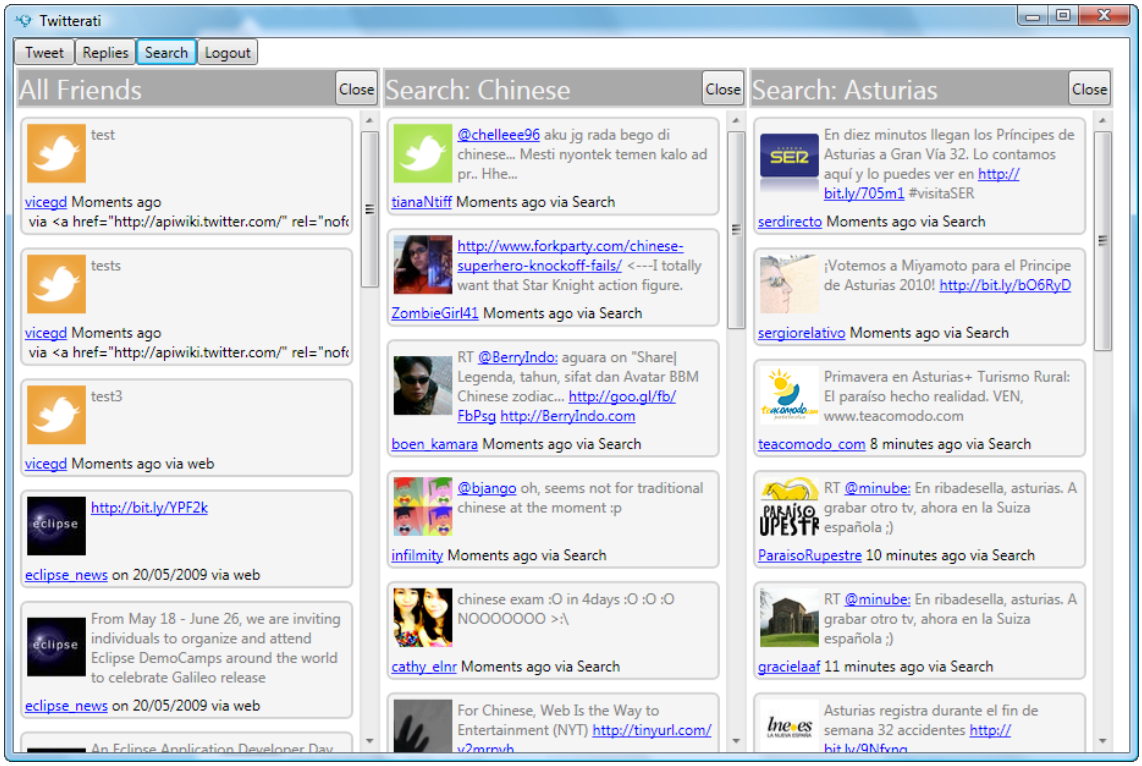


Figura 16.19: Prueba 3. Aspecto general de *Tweeterati*

El tercer caso de prueba se basa en una aplicación real denominada Tweeterati[6], la cual es un cliente de Twitter[7] desarrollado utilizando la tecnología WPF (Windows Presentation Foundation) [Nathan, 2006]. La Fig. 16.19 muestra el aspecto general de una de las pantallas de Tweeterati. Dicha aplicación ofrece características como las siguientes:

- Entrada y salida de cuentas de Twitter mediante el sistema de autenticación.
- Comprobación de los mensajes enviados.
- Lectura de los mensajes de otros contactos.
- Realización de búsquedas en Twitter.

[6] http://tweeterati.codeplex.com/

[7] Twitter es un servicio gratuito de *microblogging* que permite enviar mensajes de texto, denominados *tweets*, de una longitud máxima de 140 caracteres. El envío de estos mensajes se puede realizar por el sitio web de Twitter, desde un teléfono móvil, desde programas de mensajería instantánea o desde aplicaciones de terceros como Tweeterati

- Respuesta a mensajes.
- Envío de mensajes directos a otras personas.

Al igual que en el segundo caso de prueba, durante la integración continua se introducirán en un servidor Web los archivos generados, acción que dará lugar al consumo de tiempo durante la etapa de despliegue. En este caso, al tratarse de código compilado y no interpretado, habrá que compilar también los ensamblados regenerados cuando corresponda.

Caso de prueba basado en el ámbito expuesto en la sección 15.3.1.

Lenguaje de dominio específico

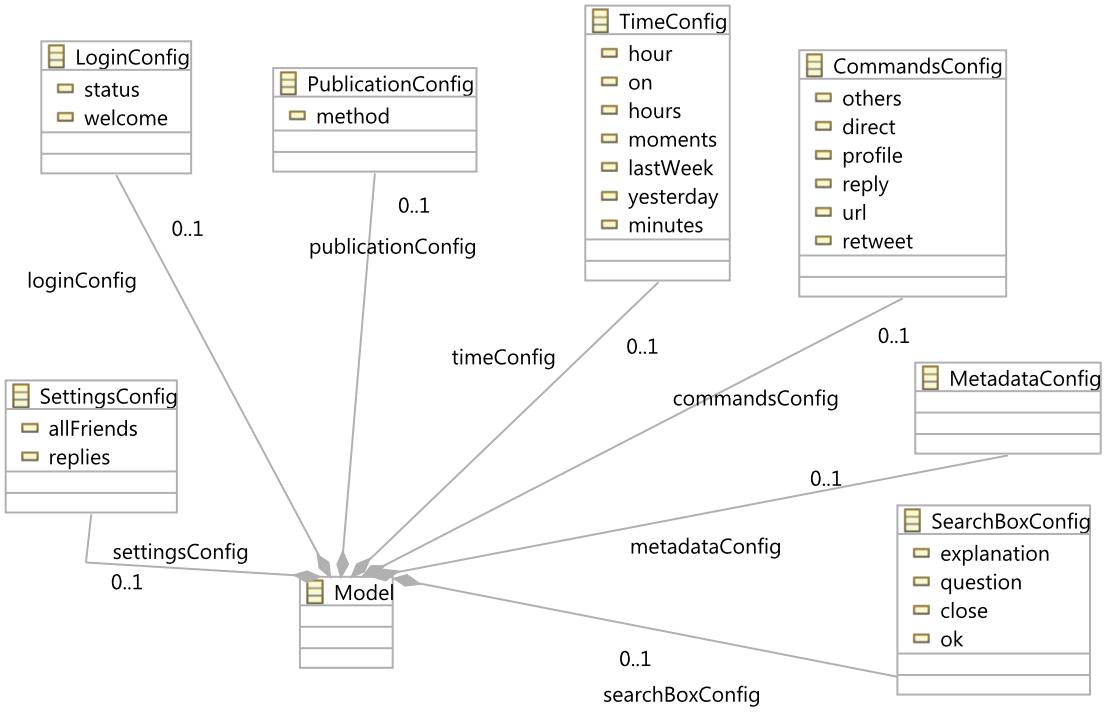


Figura 16.20: Prueba 3. Fragmento del metamodelo de TWILanguage

Tweeterati es una aplicación ejecutable compuesta por librerías que le dan funcionalidad. Como la aplicación no ofrece mecanismos de configuración de bajo nivel, se ha optado por construir un DSL que permita modificar únicamente el código fuente necesario para cambiar ciertos parámetros, de forma que únicamente haya que recompilar el código estrictamente necesario.

La Fig. 16.20 muestra, por motivos de tamaño, únicamente un pequeño fragmento del metamodelo del DSL construido. Dicho metamodelo puede observarse con más detalle en las Figs. C.7 y C.8.

Acciones realizadas

Para la realización de esta evaluación se han hecho, en orden secuencial, una serie de modificaciones en el modelo de entrada utilizado por el generador de artefactos. Dichas modificaciones están reflejadas en la tabla 16.6.

Número	Descripción
1	Generar de todos los artefactos posibles a partir del modelo inicial
2	No realizar ninguna modificación en el modelo
3	Cambiar la cultura del sitio
4	Cambiar la configuración para que se muestren por defecto todos los amigos
5	Añadir 3 pestañas de búsqueda en Twitter
6	Eliminar una pestaña de búsqueda
7	Cambiar el texto para referirse al día de ayer
8	Cambiar la información de la descripción del ensamblado *Knowledgecast.Twitter*
9	No realizar ninguna modificación en el modelo
10	Añadir el nombre de la marca en el ensamblado *Knowledgecast.Twitter.WPF*
11	Cambiar la configuración para que se muestren por defecto las respuestas y cambiar la palabra utilizada para referirse al método de publicación de los mensajes
12	Cambiar los 4 parámetros de meta información del ensamblado *Knowledgecast.Twitter.WPF.App*
13	Cambiar el mensaje de bienvenida inicial y cambiar las 4 frases referentes a la caja de búsqueda
14	Cambiar el mensaje asociado a los 6 comandos, añadir una pestaña de búsqueda y modificar la pregunta de ¿qué estás haciendo en ese momento?
15	Modificar un elemento que no afecta a la generación de artefactos

Tabla 16.6: Prueba 3. Descripción de las acciones realizadas

Tabla de resultados

Las Figs. 16.21 y 16.22 muestran los resultados obtenidos en la evaluación del presente caso de prueba en función de la entrada. Las diferentes columnas representan los mismos parámetros que para los anteriores casos de prueba, a excepción de

N.	Técnica	UV	N.artefactos	Tam.artefactos	T.generación	T.despliegue
-	1	-	54	697517	398	566,6

Tabla 16.7: Prueba 3. Datos obtenidos sin CI

la primera columna, que en este caso representa la acción realizada según los datos de la Tabla 16.6.

En el caso de prueba número 3, los valores medios utilizando únicamente MDE sin CI son mostrados en la Tabla 16.7.

Análisis de los resultados

Los datos mostrados en las Figs. 16.21 y 16.22 muestran, para este caso de prueba, los parámetros de salida obtenidos en función de las entradas. A continuación se mostrarán diferentes gráficas con los valores resultantes.

N	Técnica	UC	Nº artefactos	Tamaño artefactos	T. generación (ms)	T. despliegue (cs)
1	1	1	54	697517	398	532,9
1	2	1	11	19324	307	462,2
1	3	1	11	19324	312	461,8
1	4	1	11	19324	311	463
1	5	1	11	19324	313	486,9
1	1	0	54	697517	398	531,8
1	2	0	11	19324	312	500,8
1	3	0	11	19324	311	530
1	4	0	0	0	0	0
1	5	0	0	0	0	0
2	1	1	54	697517	398	552,4
2	2	1	11	19324	313	522,2
2	3	1	0	0	295	69
2	4	1	11	19324	319	523,2
2	5	1	0	0	290	66
2	1	0	54	697517	398	527,9
2	2	0	11	19324	319	520,2
2	3	0	0	0	286	67
2	4	0	0	0	0	0
2	5	0	0	0	0	0
3	1	1	54	697517	398	583,2
3	2	1	11	19324	318	483,2
3	3	1	1	1339	290	92,5
3	4	1	11	19324	320	472,1
3	5	1	1	1339	281	95
3	1	0	54	697517	398	496,9
3	2	0	11	19324	327	493,2
3	3	0	1	1339	289	90,3
3	4	0	0	0	0	0
3	5	0	0	0	0	0
4	1	1	54	697517	398	580,8
4	2	1	11	19324	319	527,6
4	3	1	1	1253	289	291,5
4	4	1	11	19324	313	524,1
4	5	1	1	1253	293	230,6
4	1	0	54	697517	398	606,7
4	2	0	11	19324	319	496,1
4	3	0	1	1253	292	212,2
4	4	0	0	0	0	0
4	5	0	0	0	0	0
5	1	1	54	697517	398	592
5	2	1	11	19324	316	490,1
5	3	1	1	1404	297	233
5	4	1	11	19324	318	525,3
5	5	1	1	1404	298	242,8
5	1	0	54	697517	398	596,4
5	2	0	11	19324	319	514,6
5	3	0	1	1404	309	251
5	4	0	0	0	0	0
5	5	0	0	0	0	0
6	1	1	54	697517	398	615,2
6	2	1	11	19324	314	453,9
6	3	1	1	1362	297	231
6	4	1	11	19324	323	476,9
6	5	1	1	1362	301	228,9
6	1	0	54	697517	398	664,3
6	2	0	11	19324	315	473,9
6	3	0	1	1362	297	225,1
6	4	0	0	0	0	0
6	5	0	0	0	0	0
7	1	1	54	697517	398	680,9
7	2	1	11	19324	311	475,8
7	3	1	1	2516	282	407,8
7	4	1	11	19324	328	479
7	5	1	1	2516	290	434,2
7	1	0	54	697517	398	600,9
7	2	0	11	19324	324	499,1
7	3	0	1	2516	289	428,8
7	4	0	0	0	0	0
7	5	0	0	0	0	0
8	1	1	54	697517	398	596,6
8	2	1	11	19324	318	500,2
8	3	1	1	1447	289	295,8
8	4	1	11	19324	307	495,6
8	5	1	1	1447	310	286,5

Figura 16.21: Prueba 3. Datos obtenidos 1/2

N	Técnica	UC	Nº artefactos	Tamaño artefactos	T. generación (ms)	T. despliegue (cs)
8	1	0	54	697517	398	566,6
8	2	0	11	19324	311	484,3
8	3	0	1	1447	292	301,5
8	4	0	0	0	0	0
8	5	0	0	0	0	0
9	1	1	54	697517	398	597,7
9	2	1	11	19324	313	538,6
9	3	1	0	0	286	67
9	4	1	11	19324	312	499,1
9	5	1	0	0	288	66
9	1	0	54	697517	398	522,2
9	2	0	11	19324	334	492,2
9	3	0	0	0	294	66
9	4	0	0	0	0	0
9	5	0	0	0	0	0
10	1	1	54	697517	398	601,7
10	2	1	11	19324	319	499,1
10	3	1	1	1470	293	269,2
10	4	1	11	19324	309	462,2
10	5	1	1	1470	293	262,1
10	1	0	54	697517	398	600,8
10	2	0	11	19324	315	465,2
10	3	0	1	1470	301	257,6
10	4	0	0	0	0	0
10	5	0	0	0	0	0
11	1	1	54	697517	398	575,6
11	2	1	11	19324	314	488,8
11	3	1	2	3810	300	398,4
11	4	1	11	19324	318	483,1
11	5	1	2	3810	307	433
11	1	0	54	697517	398	542,1
11	2	0	11	19324	314	533,1
11	3	0	2	3810	298	238
11	4	0	0	0	0	0
11	5	0	0	0	0	0
12	1	1	54	697517	398	577,9
12	2	1	11	19324	313	447,2
12	3	1	1	2309	292	362,4
12	4	1	11	19324	313	456,2
12	5	1	1	2309	296	347
12	1	0	54	697517	398	626,3
12	2	0	11	19324	312	460,3
12	3	0	1	2309	289	352
12	4	0	0	0	0	0
12	5	0	0	0	0	0
13	1	1	54	697517	398	614
13	2	1	11	19324	316	512,9
13	3	1	3	5243	291	355,9
13	4	1	11	19324	313	496,2
13	5	1	3	5243	304	371,2
13	1	0	54	697517	398	576,8
13	2	0	11	19324	317	494,2
13	3	0	3	5243	294	369
13	4	0	0	0	0	0
13	5	0	0	0	0	0
14	1	1	54	697517	398	597,9
14	2	1	11	19324	312	524,5
14	3	1	4	6488	298	406
14	4	1	11	19324	312	497,3
14	5	1	4	6488	305	406,1
14	1	0	54	697517	398	549,4
14	2	0	11	19324	320	516,5
14	3	0	4	6488	299	419,9
14	4	0	0	0	0	0
14	5	0	0	0	0	0
15	1	1	54	697517	398	579,8
15	2	1	11	19324	310	474,1
15	3	1	0	0	286	67
15	4	1	11	19324	325	460,2
15	5	1	0	0	288	65
15	1	0	54	697517	398	546,2
15	2	0	11	19324	314	498,1
15	3	0	0	0	284	66
15	4	0	0	0	0	0
15	5	0	0	0	0	0

Figura 16.22: Prueba 3. Datos obtenidos 2/2

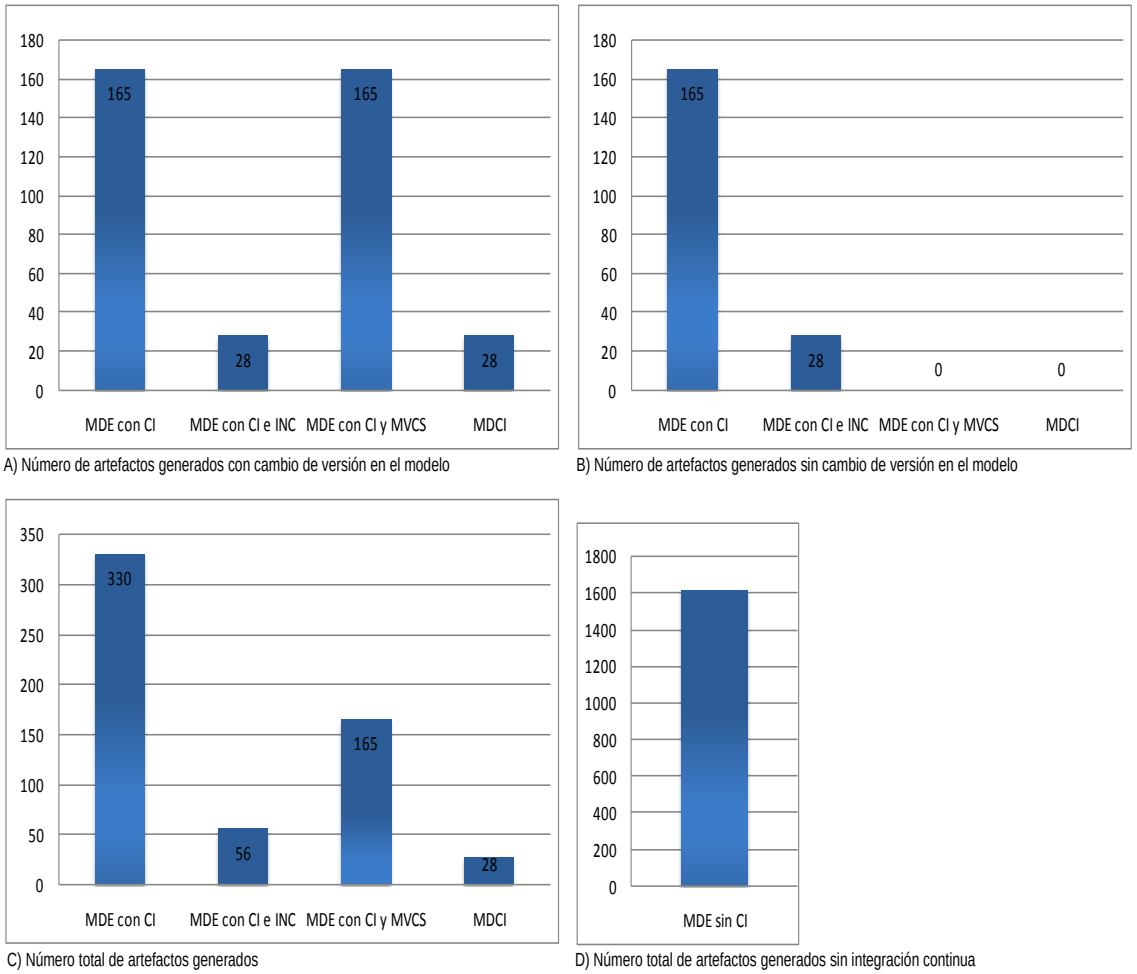


Figura 16.23: Prueba 3. Número de artefactos

El primer grupo de gráficas muestra el número de artefactos generados en función de los parámetros de entrada. Los resultados, análogos a los anteriores casos de prueba, están reflejados en las Figs. 16.23a, 16.23b, 16.23c y 16.23d

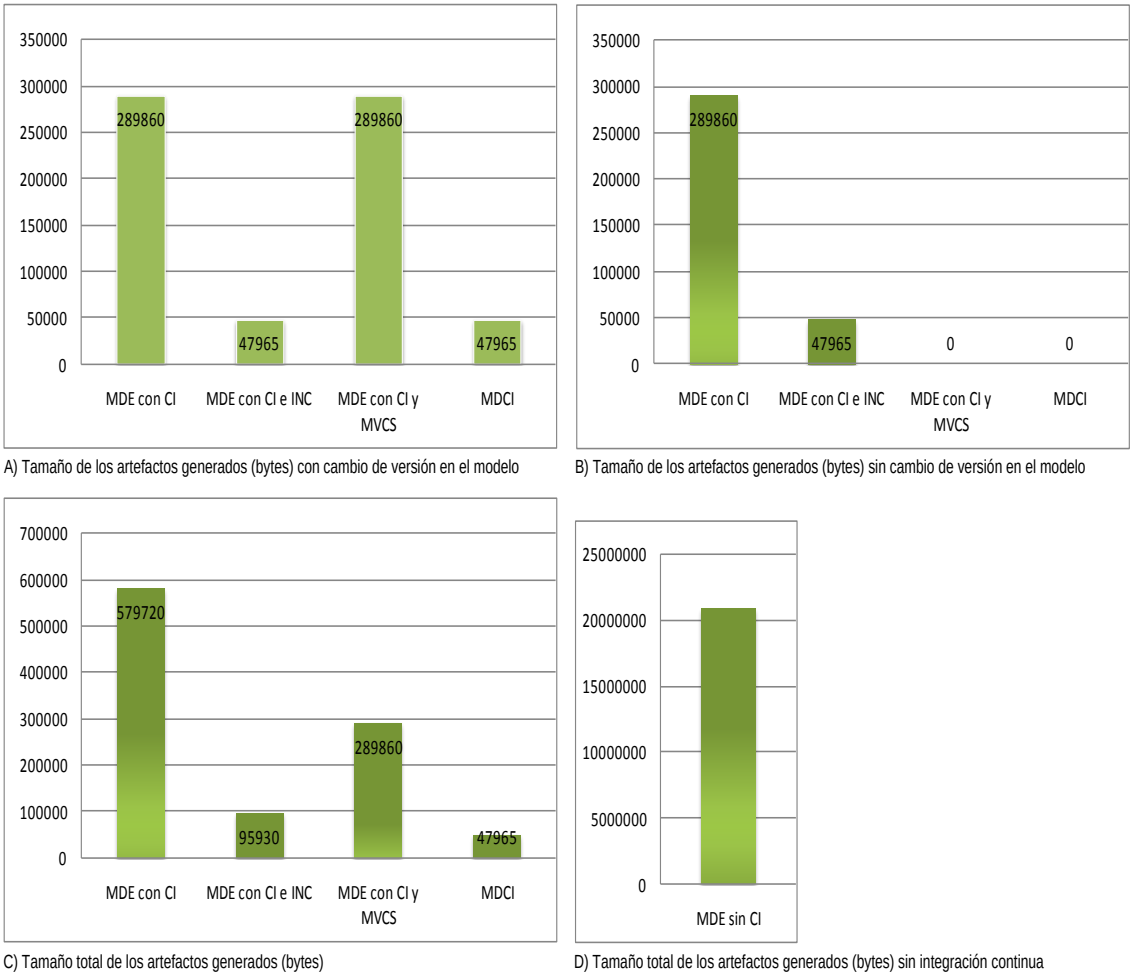


Figura 16.24: Prueba 3. Tamaño de los artefactos

El segundo grupo de gráficas muestra el tamaño total (en *bytes*) de los artefactos generados en función de la entrada. Las Figs. 16.24a. 16.24b, 16.24c y 16.24d muestran los resultados obtenidos. Dichos resultados también son análogos a los de los anteriores casos de prueba.

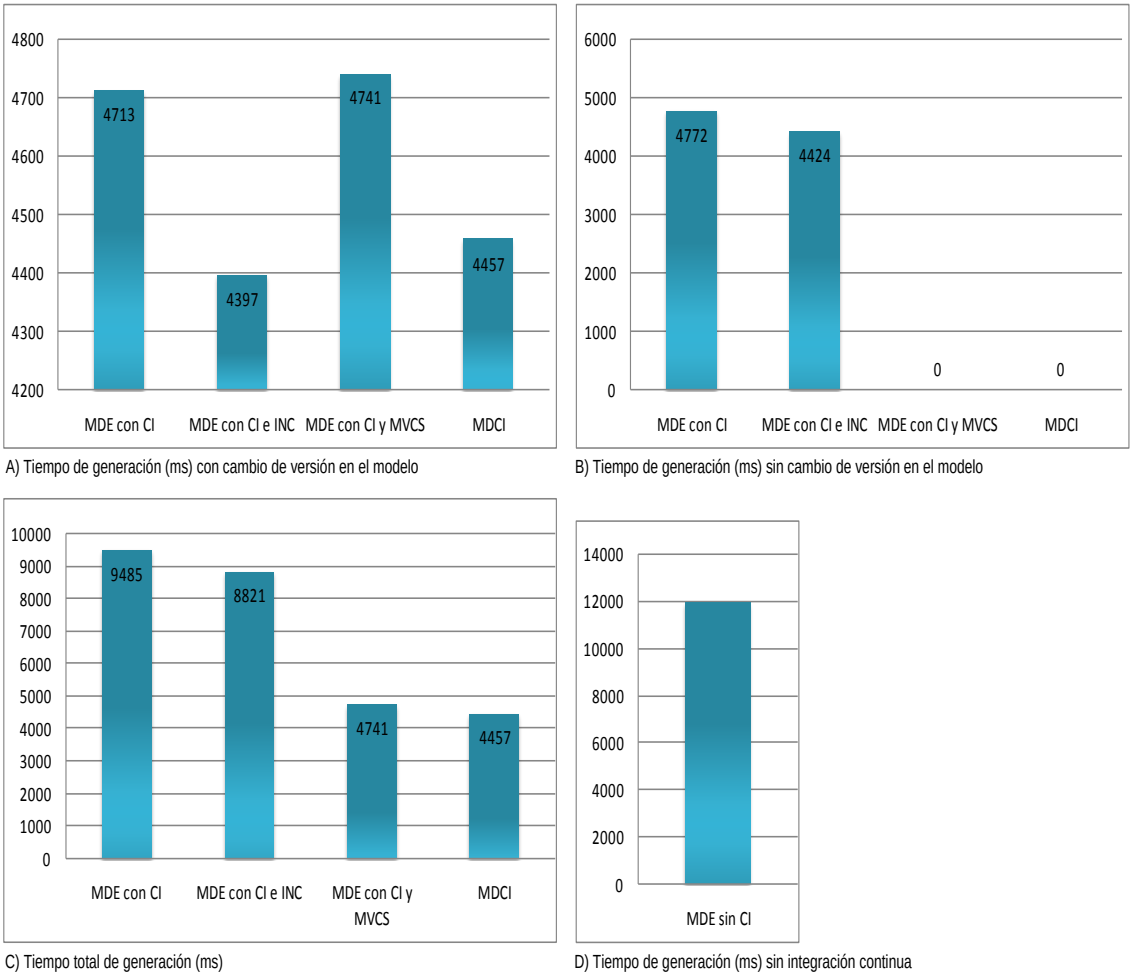


Figura 16.25: Prueba 3. Tiempo de generación

El tercer grupo de gráficas muestra el tiempo necesario (en milésimas de segundo) para generar artefactos cuando se manipulan los modelos de la entrada. Las Figs. 16.25a. 16.25b, 16.25c y 16.25d muestran los resultados obtenidos. La única diferencia destacable es que en este caso, la técnica 4 consume más tiempo que la técnica 2 en la Fig. 16.25a. Este hecho puede ser debido a un componente aleatorio inevitable.

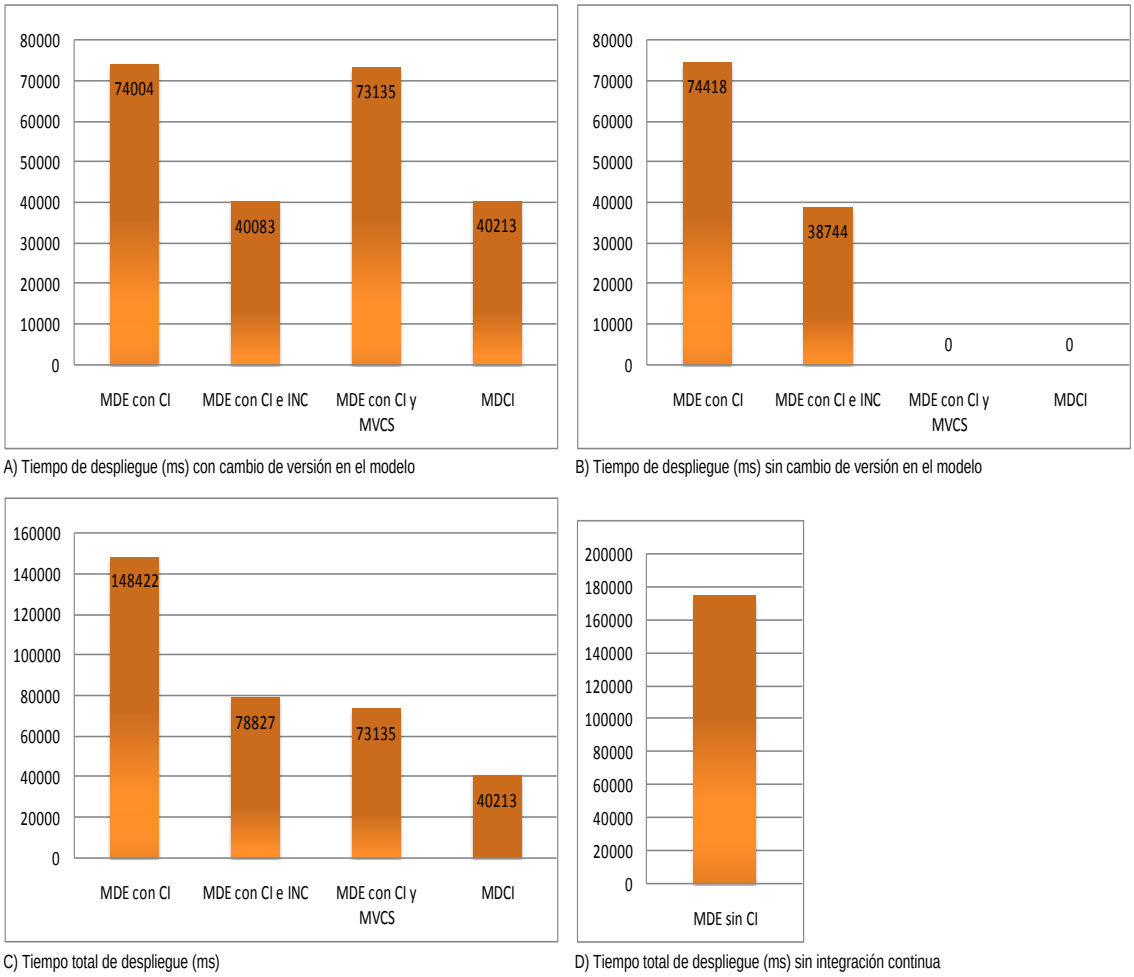


Figura 16.26: Prueba 3. Tiempo de despliegue

El cuarto grupo de gráficas muestra el tiempo necesario (en milésimas de segundo) para desplegar artefactos cuando se manipulan los modelos de la entrada. Las Figs. 16.26a. 16.26b, 16.26c y 16.26d muestran los resultados obtenidos, una vez más, análogos a los de los anteriores casos de prueba.

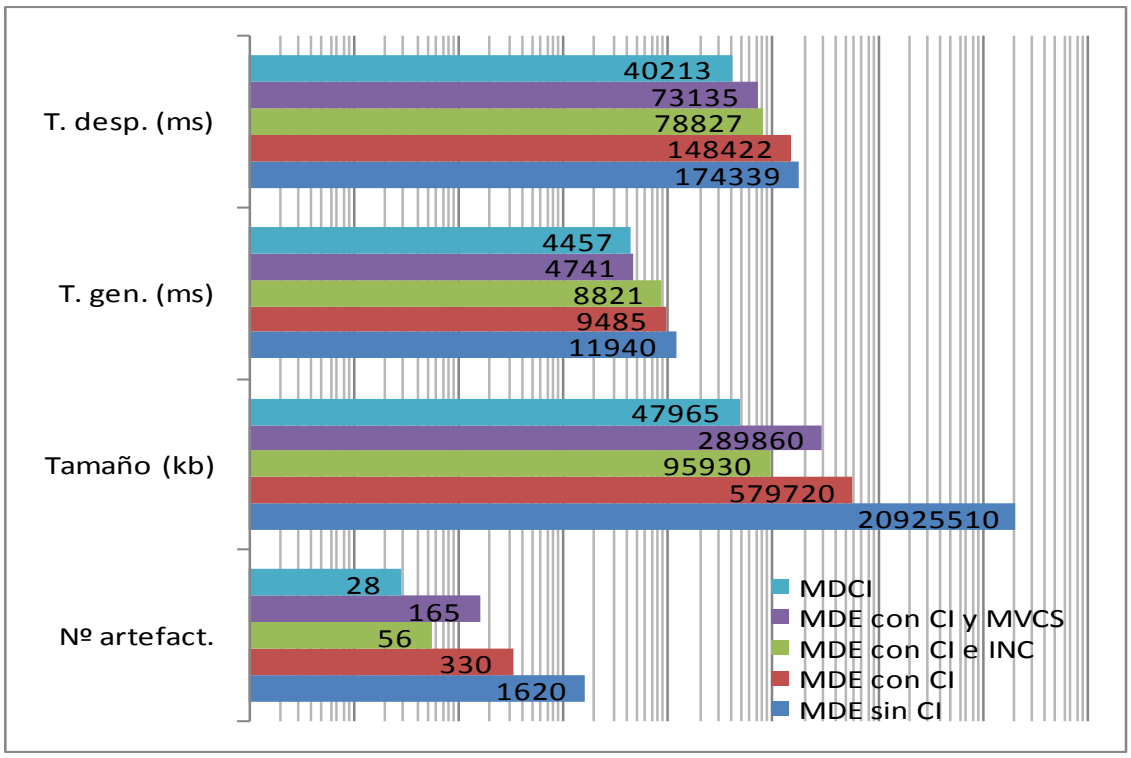


Figura 16.27: Prueba 3. Tabla logarítmica resumen de resultados obtenidos

Se observa en la Fig. 16.27 que la técnica 5 (Model-Driven Continuous Integration) es la mejor de todas en las pruebas realizadas para los cuatro parámetros de salida. Del mismo modo, la técnica 1 (tradicional) es la que peor se comporta. El resto de técnicas (2, 3 y 4) se mantienen con unos valores intermedios entre la técnica tradicional y MDCI (Model-Driven Continuous Integration), que dependen de cada caso concreto. La obtención de resultados tan homogéneos en todos los casos de prueba estudiados sugiere que los beneficios obtenidos por Model-Driven Continuous Integration respecto a otras técnicas son relevantes.

16.3. Análisis descriptivo

Debido a que la estadística descriptiva es la parte de la Estadística que resume los datos obtenidos y fundamenta todo estudio posterior, en esta sección realizaremos un análisis descriptivo de los valores resultantes de los 3 casos de prueba realizados. La Tabla 16.8 muestra un resumen de todos los parámetros de salida obtenidos[8] (tamaño de la muestra $n = 450$).

			Cuantiles				
Parámetro	Media	Desv. Típ.	0 %	25 %	50 %	75 %	100 %
Nº. artefactos	81,746	270,112	0	1	8	11	1090
Tam. artefactos	2040252	7382881,46	0	425	19324	33535	29626368
T. generación	1329,338	3929,341	0	293	318	398	16004
T. despliegue	18073,24	39146,74	0	662,5	4810,5	6847,25	155660

Tabla 16.8: Resumen de los parámetros de salida

En las Tablas 16.9, 16.10, 16.11 y 16.12 se muestra un resumen de los datos obtenidos, dividiéndolos según la variable de salida y la técnica utilizada[9] (tamaño de la muestra para cada técnica $n = 90$).

				Cuantiles				
T	Total	Media	Desv. Típ.	0 %	25 %	50 %	75 %	100 %
1	34650	385	501,613	11	11	54	1090	1090
2	1230	13,666	6,052	8	8	11	22	22
3	194	2,155	3,572	0	1	1	2	22
4	615	6,833	8,095	0	0	4	11	22
5	97	1,077	2,748	0	0	0	1	22

Tabla 16.9: Resumen del número de artefactos en función de la técnica

A partir de la información dada, la Tabla 16.13 muestra el factor de mejora obtenido utilizando MDCI (Model-Driven Continuous Integration) respecto las demás técnicas para cada una de las 4 variables de salida estudiadas. Por su parte, la Fig. 16.28 representa visualmente dicho factor de mejora.

[8] El tamaño de los artefactos se mide en bytes y los tiempos en milisegundos

[9] El número de la técnica se corresponde con los valores dados en la Tabla 16.1

				Cuantiles				
T	Total	Media	Desv. Típ.	0 %	25 %	50 %	75 %	100 %
1	910722600	10119140	13873647,55	33535	33535	697517	29626368	29626368
2	4295520	47728	36053,88	19324	19324	25556	98304	98304
3	631716	7019,067	15140,84	0	1111	2516	5243	98304
4	2147760	23864	35011,86	0	0	9662	25556	98304
5	315858	3509,533	11272,88	0	0	0	2464,25	98304

Tabla 16.10: Resumen del tamaño en función de la técnica (bytes)

				Cuantiles				
T	Total	Media	Desv. Típ.	0 %	25 %	50 %	75 %	100 %
1	500970	5566,333	7422,008	297	297	398	16004	16004
2	34818	386,866	78	307	319	345,5	470,75	552
3	29675	329,722	47,326	218	297,25	312	352	471
4	17491	194,344	203,292	0	0	153,5	355,5	581
5	15248	169,422	175,934	0	0	140,5	310	567

Tabla 16.11: Resumen del tiempo de generación en función de la técnica (ms)

				Cuantiles				
T	Total	Media	Desv. Típ.	0 %	25 %	50 %	75 %	100 %
1	5116239	56847,1	70275,822	4969	6011	9070	155660	155660
2	1711432	19015,911	18976,832	4472	5150,75	6625	43527,5	58990
3	293997	3266,633	6399,242	0	740	2320	3407,5	47120
4	858825	9542,5	16536,476	0	0	2281	6585	49010
5	152463	1694,033	5496,254	0	0	0	2301,75	50790

Tabla 16.12: Resumen del tiempo de despliegue en función de la técnica (ms)

Técnica	Nº. artefactos	Tam. artefactos	T. generación	T. despliegue
1	357,216	2883,329	32,854	33,557
2	12,68	13,599	2,283	11,225
3	2	2	1,946	1,928
4	6,34	6,799	1,147	5,633
5	1	1	1	1

Tabla 16.13: Factor de mejora de MDCI en función de la técnica

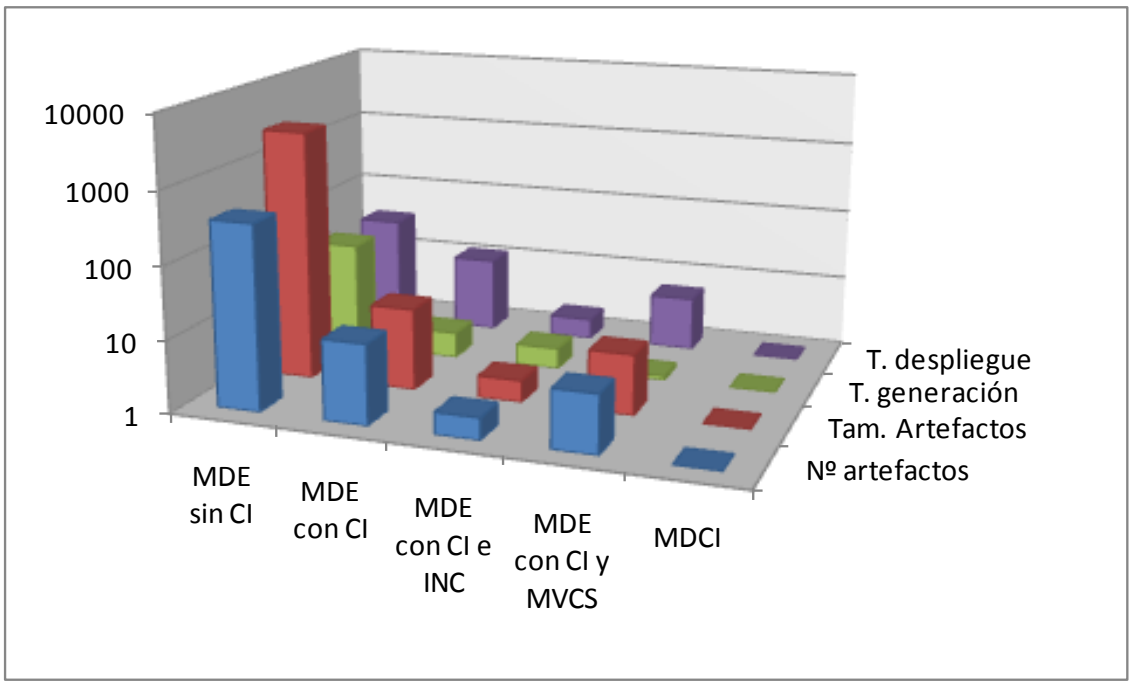


Figura 16.28: Factor de mejora de MDCI en función de la técnica

16.4. Contrastes de hipótesis

El análisis descriptivo proporciona una idea de cómo es la muestra. Sin embargo, para obtener conclusiones relativas a toda la población es necesario utilizar técnicas de inferencia estadística, siendo la más habitual el contraste de hipótesis.

16.4.1. Test para la media

Debido a que el tamaño de la muestra para cada técnica es $n = 90$, es posible utilizar el *Test t para una muestra* para realizar contrastes a cerca de la media de la población. En las Tablas 16.14, 16.15, 16.16, 16.17 y 16.18 se muestran los resultados en función de la técnica utilizada para las 4 variables de salida estudiadas. Se ha utilizado un nivel de significación $\alpha = 0,05$, que es el empleado habitualmente en la mayoría de los análisis estadísticos. Con él, se obtienen intervalos con una confianza del 95 % para todos los parámetros estudiados.

Como todos los p-values $\geq \alpha$, no se rechaza la hipótesis nula en ningún caso, con lo que no hay indicios de que las medias de los valores resultantes vayan a diferir de las muestras. Así, el factor de mejora obtenido en las muestras podría inferirse para el conjunto de la población.

Param.	Nº. artefactos	Tam. artefactos	T. generación	T. despliegue
H_0	$= 385$	$= 10119140$	$= 5566{,}333$	$= 56847{,}1$
H_1	$\neq 385$	$\neq 10119140$	$\neq 5566{,}333$	$\neq 56847{,}1$
α	0,05	0,05	0,05	0,05
p-value	1	1	1	1
int. conf.	(79,939 490,061)	(7213361 13024919)	(4011,824 7120,843)	(42128,11 71566,09)
☞	No rechazo H_0	No rechazo H_0	No rechazo H_0	No rechazo H_0

Tabla 16.14: Test para la media (Técnica 1)

Param.	Nº. artefactos	Tam. artefactos	T. generación	T. despliegue
H_0	$= 13{,}666$	$= 47728$	$= 386{,}866$	$= 19015{,}911$
H_1	$\neq 13{,}666$	$\neq 47728$	$\neq 386{,}866$	$\neq 19015{,}911$
α	0,05	0,05	0,05	0,05
p-value	0,999	1	1	1
int. conf.	(12,399 14,934)	(40176,66 55279,34)	(370,529 403,203)	(15041,29 22990,53)
☞	No rechazo H_0	No rechazo H_0	No rechazo H_0	No rechazo H_0

Tabla 16.15: Test para la media (Técnica 2)

La Fig. 16.29 muestra gráficamente un aspecto muy interesante, inferido para toda la población a partir de la muestra estudiada. Dicho aspecto son los valores medios, con una confianza del 95 %, que serán obtenidos en el futuro para nuevas muestras, en cuanto al número y el tamaño de los artefactos, el tiempo de generación y el tiempo de despliegue.

16.4.2. Relaciones entre variables

Sería interesante un estudio de las relaciones entre las 4 variables de salida. De esa forma, se mostraría matemáticamente si dichas variables están relacionadas. Así, un aumento en una de ellas podría provocar que aumenten las otras, dando lugar a

Param.	Nº. artefactos	Tam. artefactos	T. generación	T. despliegue
H_0	$= 2{,}155$	$= 7019{,}067$	$= 329{,}722$	$= 3266{,}633$
H_1	$\neq 2{,}155$	$\neq 7019{,}067$	$\neq 329{,}722$	$\neq 3266{,}633$
α	0,05	0,05	0,05	0,05
p-value	0,998	1	1	1
int. conf.	(1,407 2,903)	(3847,879 10190,255)	(319,809 339,634)	(1926,338 4606,929)
☞	No rechazo H_0	No rechazo H_0	No rechazo H_0	No rechazo H_0

Tabla 16.16: Test para la media (Técnica 3)

Param.	Nº. artefactos	Tam. artefactos	T. generación	T. despliegue
H_0	= 6,833	= 23864	= 194,344	= 9542,5
H_1	≠ 6,833	≠ 23864	≠ 194,344	≠ 9542,5
α	0,05	0,05	0,05	0,05
p-value	0.999	1	1	1
int. conf.	(5,137 8,528)	(16530,91 31197,09)	(151,765 236,923)	(6079,002 13005,998)
☞	No rechazo H_0	No rechazo H_0	No rechazo H_0	No rechazo H_0

Tabla 16.17: Test para la media (Técnica 4)

Param.	Nº. artefactos	Tam. artefactos	T. generación	T. despliegue
H_0	= 1,077	= 3509,533	= 169,422	= 1694,033
H_1	≠ 1,077	≠ 3509,533	≠ 169,422	≠ 1694,033
α	0,05	0,05	0,05	0,05
p-value	0,997	1	1	1
int. conf.	(0,502 1,653)	(1148,475 5870,592)	(132,573 206,271)	(542,865 2845,201)
☞	No rechazo H_0	No rechazo H_0	No rechazo H_0	No rechazo H_0

Tabla 16.18: Test para la media (Técnica 5)

peores resultados.

El *coeficiente de correlación lineal de Pearson* es el estadístico utilizado para medir el grado de covariación entre distintas variables relacionadas linealmente. Para ello se emplea la hipótesis nula H_0*: no existe relación entre las variables.*

	Nº. artefactos	T. despliegue	T. generación	Tam. artefactos
Nº. artefactos	1	0,946	0,998	0,999
T. despliegue	0,946	1	0,947	0,941
T. generación	0,998	0,947	1	0,999
Tam. artefactos	0,999	0,941	0,999	1

Tabla 16.19: Matriz de coeficientes de correlación (R)

En la Tabla 16.19 se muestran los coeficientes de correlación, que indican una alta correlación positiva (son todos $> 0{,}8$) entre todas las variables de salida. Este resultado avala el interés de este trabajo en reducir el valor de estas variables, ya que al estar tan íntimamente relacionadas, un mal dato en una de las variables podría reducir la calidad de los valores del resto de variables.

En la Tabla 16.20 se muestran los coeficientes de determinación, que indican el porcentaje de variación de una variable que se infiere a partir del valor de otra de las variables. Así, se observa que se puede explicar entre el 88 y el 99 % de una variable a partir de las otras.

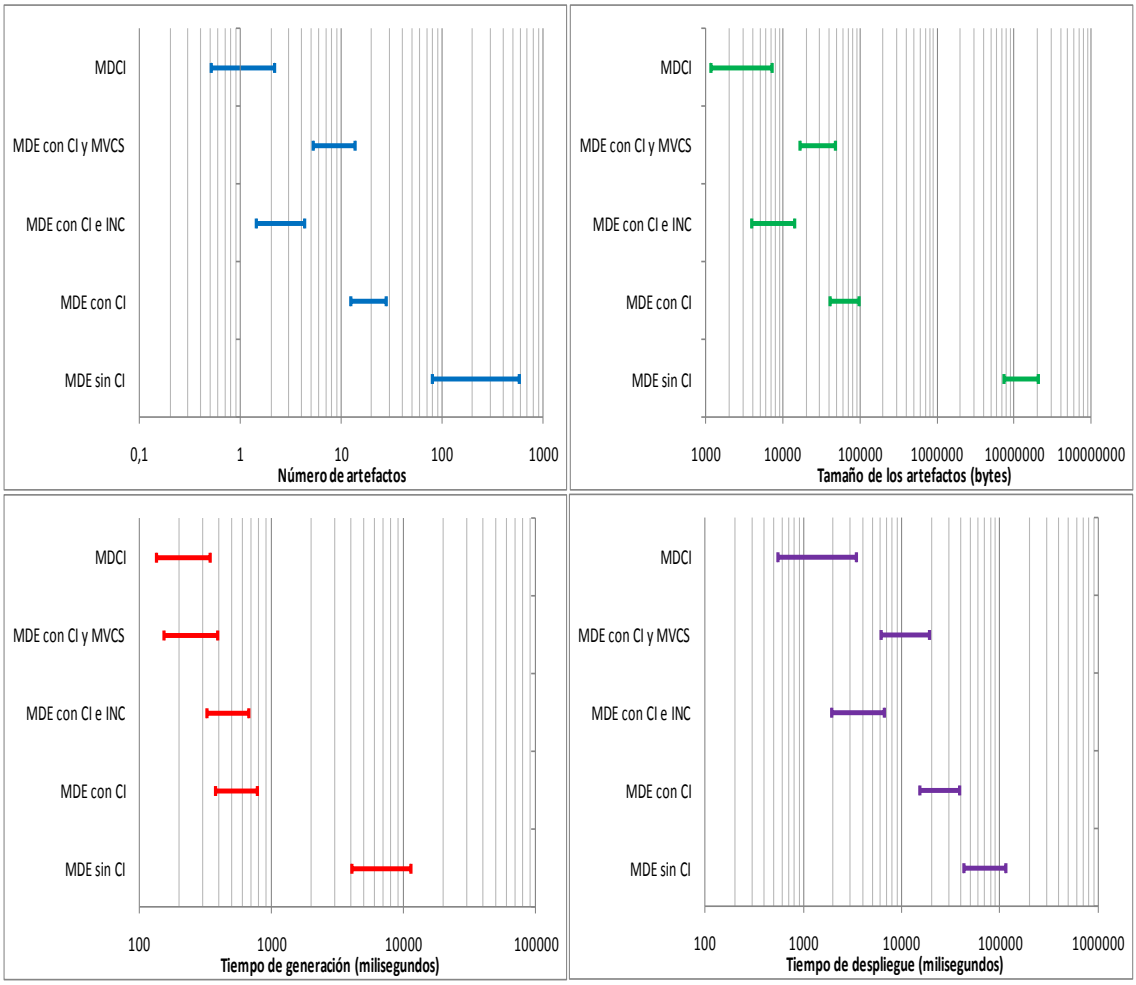


Figura 16.29: Intervalos de confianza de valores medios ($\alpha = 0,05$)

	Nº. artefactos	T. despliegue	T. generación	Tam. artefactos
Nº. artefactos	1	0,894	0,997	0.999
T. despliegue	0,894	1	0,897	0,885
T. generación	0,997	0,897	1	0,998
Tam. artefactos	0.999	0,885	0,998	1

Tabla 16.20: Matriz de coeficientes de determinación (R^2)

16.5. Regresión lineal

De entre las variables de salida estudiadas, el tiempo de despliegue es el que más puede influir en una hipotética aplicación ya en funcionamiento debido a que la modificación del modelo y generación de artefactos puede realizarse sin detener la aplicación en ninguno de los casos. Sin embargo, durante el despliegue de los artefactos generados, es posible que, por ejemplo, tenga que detenerse la aplicación (p.e., para recompilar) o el sistema de gestión de bases de datos que dicha aplicación pueda estar utilizando (p.e., para volver a un estado anterior). Por todo ello, y dada la alta correlación entre las variables de salida, se va a estudiar cómo el número de artefactos, el tamaño de los artefactos y el tiempo de generación pueden influir en el tiempo de despliegue.

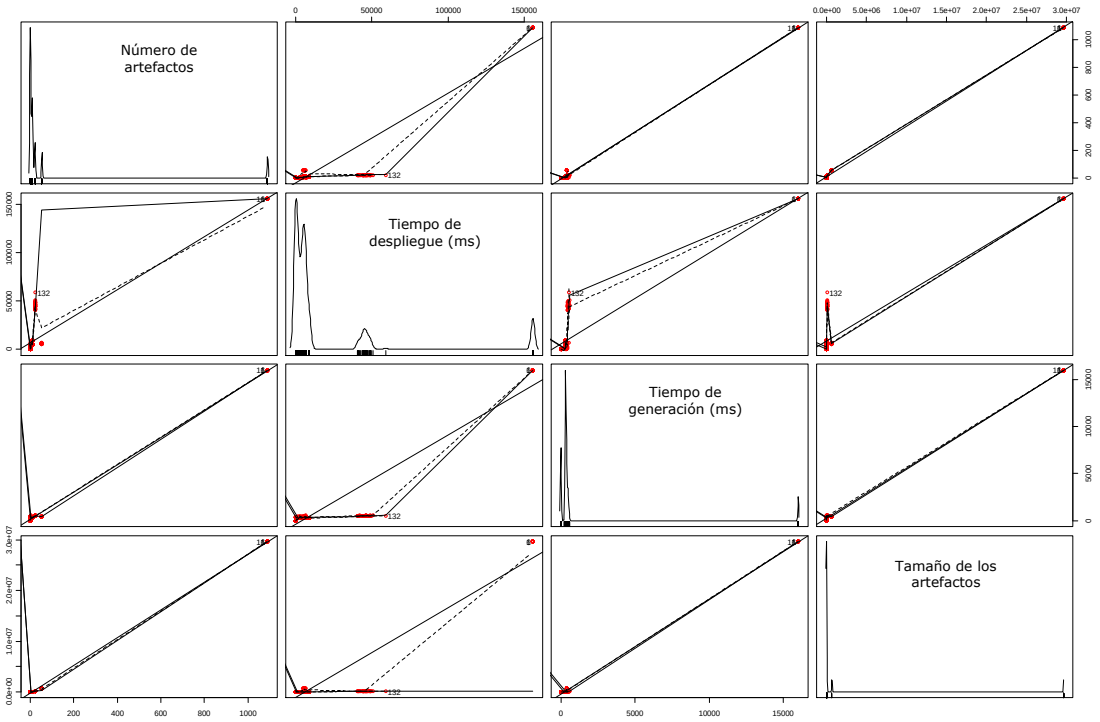


Figura 16.30: Diagrama de dispersión de las variables estudiadas

La Fig. 16.30 muestra un diagrama de dispersión de las variables principales. Cada variable está asociada con una fila y en cada columna de cada fila se representa la relación con otra de las variables. La diagonal principal no representa ninguna relación. El tiempo de despliegue está situado en la segunda fila pero parece que no hay ningún patrón claro de relación con las demás variables debido a los valores atípicos que se obtienen cuando se trabaja con la técnica número 1 (MDE sin CI).

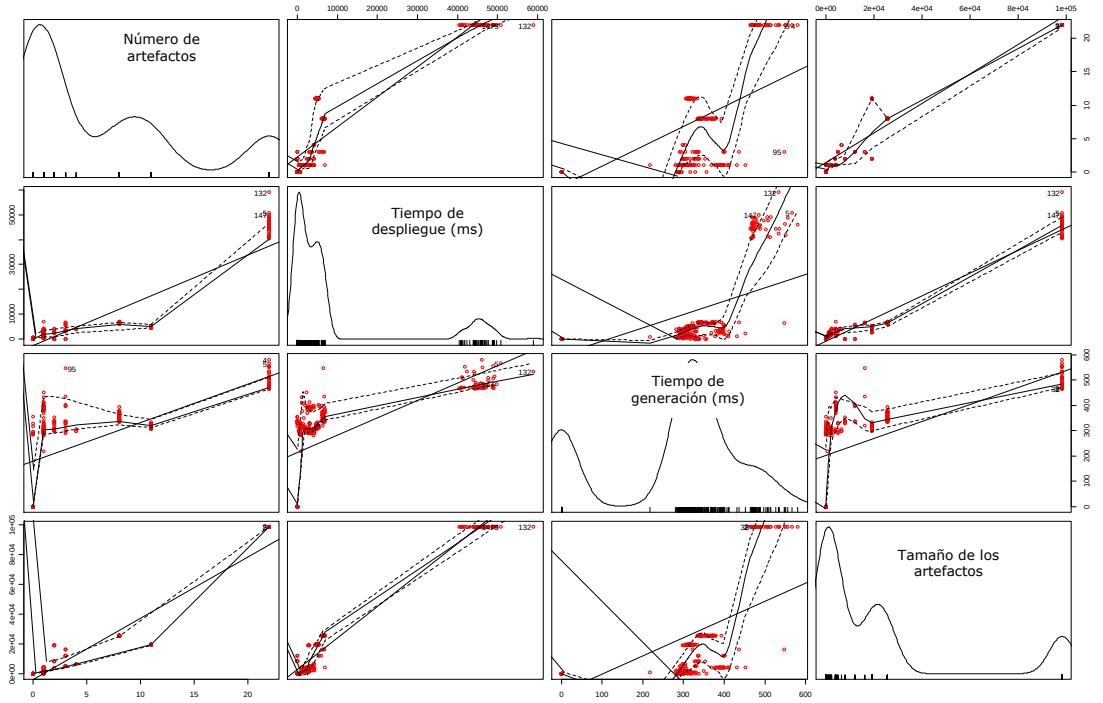


Figura 16.31: Diagrama de dispersión de las variables estudiadas (sin técnica 1)

En la Fig. 16.31 se puede ver la misma imagen pero eliminando los valores asociados a la técnica número 1. Así, se observan patrones de relaciones más claros entre las variables[10].

16.5.1. Modelos lineales simples

Debido a la alta correlación entre las variables, se pueden intentar construir modelos lineales que cuantifiquen la relación entre el tiempo de despliegue y el resto de variables. Estos modelos, a priori, parecen demasiados sencillos, ya que dependen sólo de una variable simultáneamente.

$$T.des = -2389,9 + 1815,11 * N.art$$

$$T.des = -1007 + 0,4572 * Tam.art$$

$$T.des = -6139,988 + 53,759 * T.gen$$

Todos los coeficientes del modelo son significativos porque en todos los casos el p-value $< 0{,}05$, dando por válido al modelo. Sin embargo, habrá que verificar

[10]Las líneas de puntos representan al modelo más simple. Las otras líneas representan al mejor ajuste posible. Si ambas coinciden significa que el ajuste lineal resulta adecuado

otros requisitos para estar seguros de que el modelo ajusta correctamente los datos. Por ejemplo, determinando si los residuos del modelo son homocedásticos mediante el test de *Breusch-Pagan*. En este caso el valor del test es muy próximo a 0 en todos los casos, siendo inferior al nivel de significación $\alpha = 0,05$, lo que indica que hay una variabilidad no constante en el ajuste (heterocedástico), por lo que sería recomendable buscar otros modelos mejores.

16.5.2. Modelos lineales simples - transformaciones

Hasta este momento tan sólo se han considerado los datos originales, llegando a la conclusión de que el modelo lineal no ajusta correctamente. Se pueden realizar planteamientos en los que aunque la relación no sea lineal, mediante alguna transformación, se pueda convertir a lineal. Por ejemplo, mediante el empleo de logaritmos, si $y = x^2 * b$ entonces $log(y) = 2 * log(x) + log(b)$.

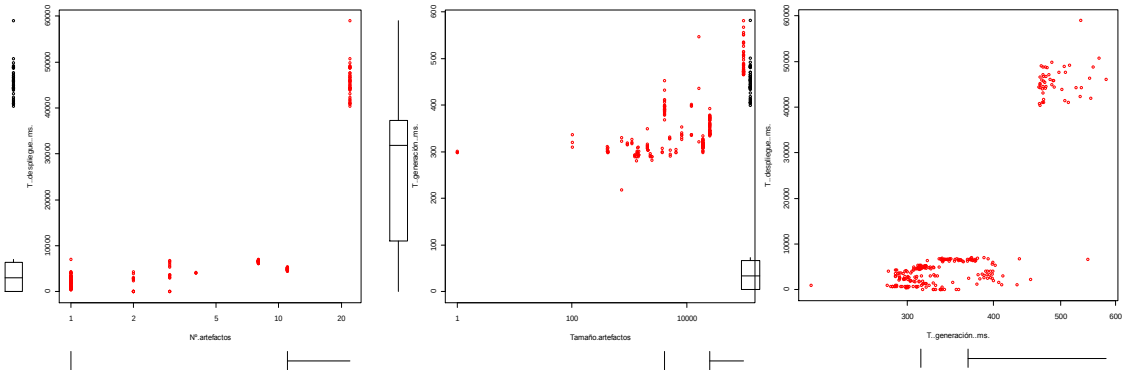


Figura 16.32: Diagrama de dispersión de las variables (utilizando log(x))

En las Figs. 16.32 y 16.33 se comprueba visualmente que estas nuevas relaciones lineales no son adecuadas[11]. Por lo tanto se desechan tales transformaciones.

Existen otras posibilidades que podrían utilizarse como la transformación de *Box-Cox*. Sin embargo, para evitar un análisis excesivamente largo, se realizará directamente un modelo de regresión lineal múltiple, en el que se utilizarán simultáneamente todas las variables cuantitativas del análisis.

[11]El eje de ordenadas es el tiempo de despliegue y los distintos ejes de abscisas son, de izquierda a derecha, el número de artefactos, el tamaño de los artefactos y el tiempo de generación

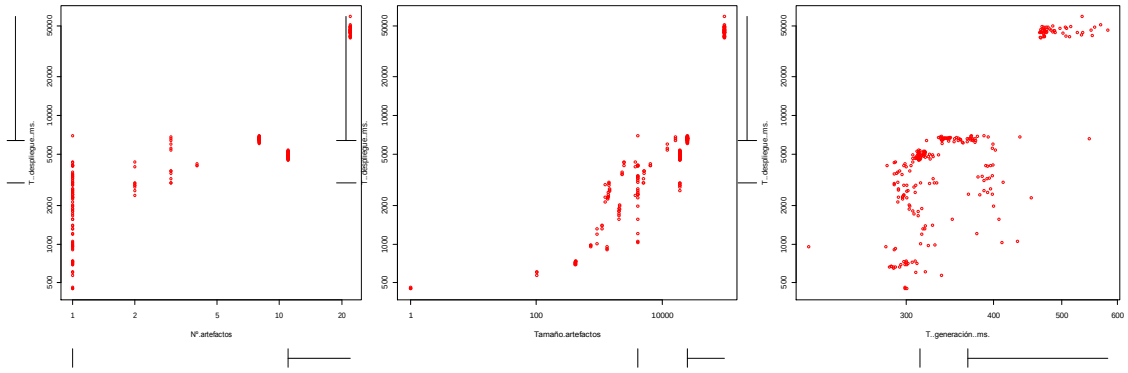


Figura 16.33: Diagrama de dispersión de las variables (utilizando log(x) y log(y))

16.5.3. Modelo lineal múltiple

La regresión lineal múltiple generaliza al modelo anterior al incorporar varias variables independientes. La siguiente ecuación es una primera aproximación del modelo.

$$T.des = -90,31339 - 634,61798 * N.art + 0,59802 * Tam.art - 0,15537 * T.gen$$

El tiempo de generación no es un coeficiente significativo (p-value = 0,87), por lo que habrá que simplificar el modelo quedando de la siguiente forma.

$$T.des = -118,1179 - 637,05956 * N.art + 0,59804 * Tam.art$$

Una vez estimado el modelo, habrá que verificar si ajusta bien los datos. Determinando la bondad del modelo mediante el estudio de los factores de inflación de varianza obtenemos unos valores superiores a 4, concretamente 10, lo cual indica que hay colinealidad, sobrando una variable del modelo. Esto implica que el modelo quedaría nuevamente reducido a un modelo lineal simple, y como ya se ha indicado, dicho modelo no ajustará bien los datos.

16.5.4. Modelo de ecuaciones estructurales

Para definir relaciones entre una variable descriptiva o predictora (X) y una respuesta cuantitativa (Y) se suelen usar *modelos de regresión lineales* [Montgomery et al., 2007]. Así, se estiman los parámetros del modelo, se verifica su validez y precisión y se descarta o se acepta ese modelo. También existen los *modelos de regresión lineales múltiples* para los casos en los que haya más de una variable descriptiva. Sin

embargo, para los casos en los que las relaciones pueden ser más complejas se utilizan *los modelos de ecuaciones estructurales* debido a que las relaciones entre las variables predictoras hacen difícil un modelamiento mediante regresión lineal múltiple. Además, algunas de las variables que son consideradas predictoras de la respuesta de interés pueden ser reflejo de un factor subyacente o variable latente [Zamora and Lemus, 2008].

Los modelos de ecuaciones estructurales son una combinación entre el análisis factorial y la regresión lineal múltiple y tienen una gran importancia en la investigación. Se pueden definir como una secuencia de modelos matemáticos-estadísticos útiles para describir las relaciones entre varias variables que permiten analizar un fenómeno de interés, teniendo en cuenta la estructura de covarianzas existentes entre ellas [Zamora and Lemus, 2008]. Estas ideas comenzaron con Sewall Wright [Wright, 1932] en su análisis de trayectorias y se reafirmaron con Karl Jöreskog [Joreskog, 1971, Joreskog and Goldberger, 1975], que le dieron una base más teórica.

El objetivo fundamental es que el modelo sea lo suficientemente aproximado para representar la realidad, ya que no tiene sentido plantear una hipótesis de si el modelo es correcto o no porque al hacer una simplificación ya se sabe que realmente no puede ser correcto. Para hacer la simplificación se eliminan elementos que dan lugar al *término de error*. Sin embargo, los modelos de ecuaciones estructurales presuponen una relación lineal entre las variables, lo cual, como se verá en el análisis de la covarianza (véase sección 16.7), será descartado, no siendo recomendable utilizar este tipo de modelos en el análisis.

16.6. Análisis de la varianza

El análisis de la varianza particulariza el modelo de regresión lineal cuando las variables independientes son cualitativas y la dependiente es cuantitativa. Es decir, las variables explicativas son categóricas y la explicada es un valor numérico. En este apartado se estudiará cómo la técnica y la unidad de control empleada influyen en las 4 variables principales de salida estudiadas.

16.6.1. En función de la técnica empleada

Las siguientes ecuaciones sirven para estimar las variables de salida en función de la técnica empleada, teniendo en cuenta que el número de la técnica se corresponde con los valores dados en la Tabla 16.1. Como en todos los casos el p-value $< 0{,}05$, entonces las diferencias son significativas, dando por válido al modelo.

$$N.art = \begin{cases} 385 & \text{si técnica} = 1 \\ 385 - 371,33 & \text{si técnica} = 2 \\ 385 - 382,84 & \text{si técnica} = 3 \\ 385 - 378,17 & \text{si técnica} = 4 \\ 385 - 383,92 & \text{si técnica} = 5 \end{cases}$$

$$Tam.art = \begin{cases} 10119140 & \text{si técnica} = 1 \\ 10119140 - 10071412 & \text{si técnica} = 2 \\ 10119140 - 10112121 & \text{si técnica} = 3 \\ 10119140 - 10095276 & \text{si técnica} = 4 \\ 10119140 - 10115631 & \text{si técnica} = 5 \end{cases}$$

$$T.gen = \begin{cases} 5566,3 & \text{si técnica} = 1 \\ 5566,3 - 5179,5 & \text{si técnica} = 2 \\ 5566,3 - 5179,5 & \text{si técnica} = 3 \\ 5566,3 - 5179,5 & \text{si técnica} = 4 \\ 5566,3 - 5396,9 & \text{si técnica} = 5 \end{cases}$$

$$T.des = \begin{cases} 56847 & \text{si técnica} = 1 \\ 56847 - 37831 & \text{si técnica} = 2 \\ 56847 - 53581 & \text{si técnica} = 3 \\ 56847 - 47305 & \text{si técnica} = 4 \\ 56847 - 55153 & \text{si técnica} = 5 \end{cases}$$

Estos resultados están influidos por el aumento de escala cuando se utiliza la primera técnica. Así, para una mayor claridad, las siguientes ecuaciones aunque equivalentes a las primeras, no muestran los resultados cuando se trabaja con la técnica número 1 (MDE sin CI).

$$N.art = \begin{cases} 13,6667 & \text{si técnica} = 2 \\ 13,6667 - 11,5111 & \text{si técnica} = 3 \\ 13,6667 - 6,8333 & \text{si técnica} = 4 \\ 13,6667 - 12,5889 & \text{si técnica} = 5 \end{cases}$$

$$Tam.art = \begin{cases} 47728 & \text{si técnica} = 2 \\ 47728 - 40709 & \text{si técnica} = 3 \\ 47728 - 23864 & \text{si técnica} = 4 \\ 47728 - 44219 & \text{si técnica} = 5 \end{cases}$$

$$T.gen = \begin{cases} 386,87 & \text{si técnica} = 2 \\ 386,87 - 57,14 & \text{si técnica} = 3 \\ 386,87 - 192,52 & \text{si técnica} = 4 \\ 386,87 - 217,44 & \text{si técnica} = 5 \end{cases}$$

$$T.des = \begin{cases} 19016 & \text{si técnica} = 2 \\ 19016 - 15749 & \text{si técnica} = 3 \\ 19016 - 9473 & \text{si técnica} = 4 \\ 19016 - 17322 & \text{si técnica} = 5 \end{cases}$$

16.6.2. En función de la unidad de control

Las siguientes ecuaciones servirían para estimar las variables de salida en función de la unidad de control empleada, teniendo en cuenta que UC = 0 se refiere a que la unidad de control es tal que con un MVCS no se debería detectar que ha habido ningún cambio en el modelo de entrada. Sin embargo, todos los p-values $\geq$ 0,05, por lo que en este caso no podemos dar por válido al modelo.

$$N.art = \begin{cases} 80,164 & \text{si UC} = 0 \\ 80,164 + 3,164 & \text{si UC} = 1 \end{cases}$$

p-value: 0,9013

$$Tam.art = \begin{cases} 2034777 & \text{si UC} = 0 \\ 2034777 + 10949 & \text{si UC} = 1 \end{cases}$$

p-value: 0,9875

$$T.gen = \begin{cases} 1255,6 & \text{si UC} = 0 \\ 1255,6 + 147,5 & \text{si UC} = 1 \end{cases}$$

p-value: 0,6909

$$T.des = \begin{cases} 15791 & \text{si UC} = 0 \\ 15791 + 4564 & \text{si UC} = 1 \end{cases}$$

p-value: 0,2166

16.6.3. En función de la técnica y de la unidad de control

Cómo se ha visto, el valor de la UC por sí solo no es suficiente como para poder estimar una variable de salida. Sin embargo, si puede utilizarse para obtener ecuaciones más ajustadas en cuanto al valor que tendrán las 4 variables de salida en función de la técnica utilizada. A continuación se muestran dichas ecuaciones.

$$N.art = \begin{cases} 385 & \text{si técnica} = 1 \text{ y UC} = 1 \\ 385 - 371,3 & \text{si técnica} = 2 \text{ y UC} = 1 \\ 385 - 382,8 & \text{si técnica} = 3 \text{ y UC} = 1 \\ 385 - 371,3 & \text{si técnica} = 4 \text{ y UC} = 1 \\ 385 - 382,8 & \text{si técnica} = 5 \text{ y UC} = 1 \\ 385 & \text{si técnica} = 1 \text{ y UC} = 0 \\ 385 - 371,3 & \text{si técnica} = 2 \text{ y UC} = 0 \\ 385 - 382,8 & \text{si técnica} = 3 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 4 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 5 \text{ y UC} = 0 \end{cases}$$

$$Tam.art = \begin{cases} 1,012e+07 & \text{si técnica} = 1 \text{ y UC} = 1 \\ 1,012e+07-1,007e+07 & \text{si técnica} = 2 \text{ y UC} = 1 \\ 1,012e+07-1,011e+07 & \text{si técnica} = 3 \text{ y UC} = 1 \\ 1,012e+07-1,007e+07 & \text{si técnica} = 4 \text{ y UC} = 1 \\ 1,012e+07-1,011e+07 & \text{si técnica} = 5 \text{ y UC} = 1 \\ 1,012e+07 & \text{si técnica} = 1 \text{ y UC} = 0 \\ 1,012e+07-1,007e+07 & \text{si técnica} = 2 \text{ y UC} = 0 \\ 1,012e+07-1,011e+07 & \text{si técnica} = 3 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 4 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 5 \text{ y UC} = 0 \end{cases}$$

$$T.gen = \begin{cases} 5566 & \text{si técnica} = 1 \text{ y UC} = 1 \\ 5566-5177 & \text{si técnica} = 2 \text{ y UC} = 1 \\ 5566-5234 & \text{si técnica} = 3 \text{ y UC} = 1 \\ 5566-5178 & \text{si técnica} = 4 \text{ y UC} = 1 \\ 5566-5227 & \text{si técnica} = 5 \text{ y UC} = 1 \\ 5566 & \text{si técnica} = 1 \text{ y UC} = 0 \\ 5566-5182 & \text{si técnica} = 2 \text{ y UC} = 0 \\ 5566-5239 & \text{si técnica} = 3 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 4 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 5 \text{ y UC} = 0 \end{cases}$$

$$T.des = \begin{cases} 56883,02 & \text{si técnica} = 1 \text{ y UC} = 1 \\ 56883,02-37848,71 & \text{si técnica} = 2 \text{ y UC} = 1 \\ 56883,02-53497,18 & \text{si técnica} = 3 \text{ y UC} = 1 \\ 56883,02-37798,02 & \text{si técnica} = 4 \text{ y UC} = 1 \\ 56883,02-53494,96 & \text{si técnica} = 5 \text{ y UC} = 1 \\ 56883,02 & \text{si técnica} = 1 \text{ y UC} = 0 \\ 56883,02-37885,51 & \text{si técnica} = 2 \text{ y UC} = 0 \\ 56883,02-53735,60 & \text{si técnica} = 3 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 4 \text{ y UC} = 0 \\ 0 & \text{si técnica} = 5 \text{ y UC} = 0 \end{cases}$$

16.7. Análisis de la covarianza

Se procederá a crear un modelo completo basándose en el análisis de la covarianza (particularización del modelo de regresión lineal) y en el que se utilizarán como variables dependientes elementos cualitativos y cuantitativos al mismo tiempo[12] con el objetivo de crear una estimación fiable del tiempo de despliegue. Para ello, se emplearan las demás variables de salida: el número de artefactos, el tamaño de

[12]Los estudios realizados utilizando regresión lineal y análisis de varianza no eran completos, puesto que estaban basados únicamente en variables cuantitativas o cualitativas respectivamente

los artefactos y el tiempo de generación. También se emplearán las 2 variables de entrada: la técnica utilizada y la unidad de control empleada.

$$T.des = \begin{cases} -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen & \text{si téc.} = 1 \text{ y UC} = 0 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen - 2766 & \text{si téc.} = 2 \text{ y UC} = 0 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen + 3,693 & \text{si téc.} = 3 \text{ y UC} = 0 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen + 48400 & \text{si téc.} = 4 \text{ y UC} = 0 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen + 48400 & \text{si téc.} = 5 \text{ y UC} = 0 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen + 71,84 & \text{si téc.} = 1 \text{ y UC} = 1 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen - 696,7 & \text{si téc.} = 2 \text{ y UC} = 1 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen - 571,6 & \text{si téc.} = 3 \text{ y UC} = 1 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen - 51730 & \text{si téc.} = 4 \text{ y UC} = 1 \\ -48400 + 1712 * N.art - 0,130 * Tam.art + 137,8 * T.gen - 46160 & \text{si téc.} = 5 \text{ y UC} = 1 \end{cases}$$

Una vez estimado el modelo, como en los casos anteriores, habrá que verificar si ajusta bien los datos. Determinando la bondad del modelo mediante el estudio de los factores de inflación de varianza obtenemos unos resultados que en ningún caso superan el valor de 4, lo cual indica que no se aprecia colinealidad. Es decir, el test ha sido superado con éxito.

Ahora se determinará si todos los residuos del modelo son homocedásticos mediante el test de *Breusch-Pagan*. En este caso el valor del test es 0,5157, siendo superior al nivel de significación $\alpha = 0,05$, lo que indica que no se rechaza la hipótesis de homocedasticidad, o lo que es lo mismo, que se ha superado el test.

Respecto a la linealidad o no del modelo, el resultado es un p-value $<$ 0,05. Por ese motivo habría que rechazar la hipótesis nula y tratar de aumentar el grado del modelo buscando un ajuste mejor. Sin embargo, el modelo podría complicarse excesiva e innecesariamente. Además, los resultados obtenidos en la evaluación son ya suficientes para mostrar los grandes factores de mejora obtenidos en todos los casos en los que se utiliza MDCI (Model-Driven Continuous Integration).

16.8. Conclusiones

Cuando se trabaja con MDE, se puede o no utilizar integración continua durante el desarrollo. Si se considera su empleo, también se pueden utilizar los sistemas de control de versiones especialmente orientados para trabajar con modelos y los generadores incrementales de artefactos. La unión de todas estas tecnologías hace que los desarrollos puedan mejorarse considerablemente y facilita la modificación de aplicaciones ya en producción con un menor impacto sobre ellas.

Los factores de mejora obtenidos sobre otras técnicas son superiores a 2 en todos los casos estudiados respecto al número y el tamaño de los artefactos. El factor de mejora respecto al tiempo de generación oscila entre 1,14 y 32,85. Por último, el factor de mejora varía entre 1,92 y 33,55 respecto el tiempo de despliegue de las aplicaciones desarrolladas.

En el próximo capítulo se mostrarán los fundamentos del proceso MDCI (Model-Driven Continuous Integration), surgido a partir de todo el trabajo realizado durante esta Tesis.

CAPÍTULO 17

Definición de un proceso MDCI

En los anteriores capítulos se ha mostrado cómo es MDCIP (Model-Driven Continuous Integration Prototype) y se han realizado múltiples pruebas utilizando algunos de los ámbitos de aplicación planteados. El objetivo ha sido dejar patente las ventajas de MDCI (Model-Driven Continuous Integration) frente a otras posibilidades de desarrollo [García-Díaz et al., 2011a].

En este capítulo se presentarán los fundamentos en los que se basa el proceso Model-Driven Continuous Integration planteado y el conjunto de elementos que lo forman, dando respuestas a las preguntas que han de responderse para definir cualquier proceso.

⁂ ⁂ ⁂ ⁂

17.1. Proceso de Integración Continua tradicional

La Fig. 17.1 muestra el proceso de integración continua cuando se utiliza una aproximación de desarrollo tradicional:

- La idea es que los diferentes miembros de un equipo de desarrollo trabajen contra un repositorio gestionado a través de un VCS, que facilita el trabajo colaborativo y distribuido.

- La herramienta de CI permanece a la espera de que se produzcan cambios en el repositorio de la aplicación.

- Pueden existir diferentes tipos de *desencadenantes o disparadores* que hagan a la herramienta de CI comprobar el estado del repositorio. Por ejemplo, un intervalo de tiempo, un instante determinado, el éxito de una compilación de otro programa, etc.

- Cuando se detectan cambios en el repositorio, es cuando las diferentes tareas de construcción de artefactos tienen lugar por medio de *scripts* automatizados gracias al uso de herramientas como Ant[1]:

  - Durante la *compilación* se generan las librerías o aplicaciones.
  - Durante las *pruebas* se realizan comprobaciones sobre el código fuente o sobre la aplicación.
  - Durante la etapa de *informes y métricas* se hacen estudios sobre, por ejemplo, el empleo de guías de estilo o el porcentaje de cobertura del código.
  - Durante la etapa de *documentación* se crean los archivos de ayuda a partir de los comentarios introducidos en el código.
  - Pueden existir otras etapas como por ejemplo la construcción de un instalador para la aplicación.

- Si existiera algún error o problema durante la construcción de los artefactos, la herramienta de CI informaría al responsable mediante algún sistema (p.e., Web, correo electrónico, RSS) para que se corrigiera tan pronto como sea posible y se vuelva a lanzar el proceso de integración continua.

[1] http://ant.apache.org/

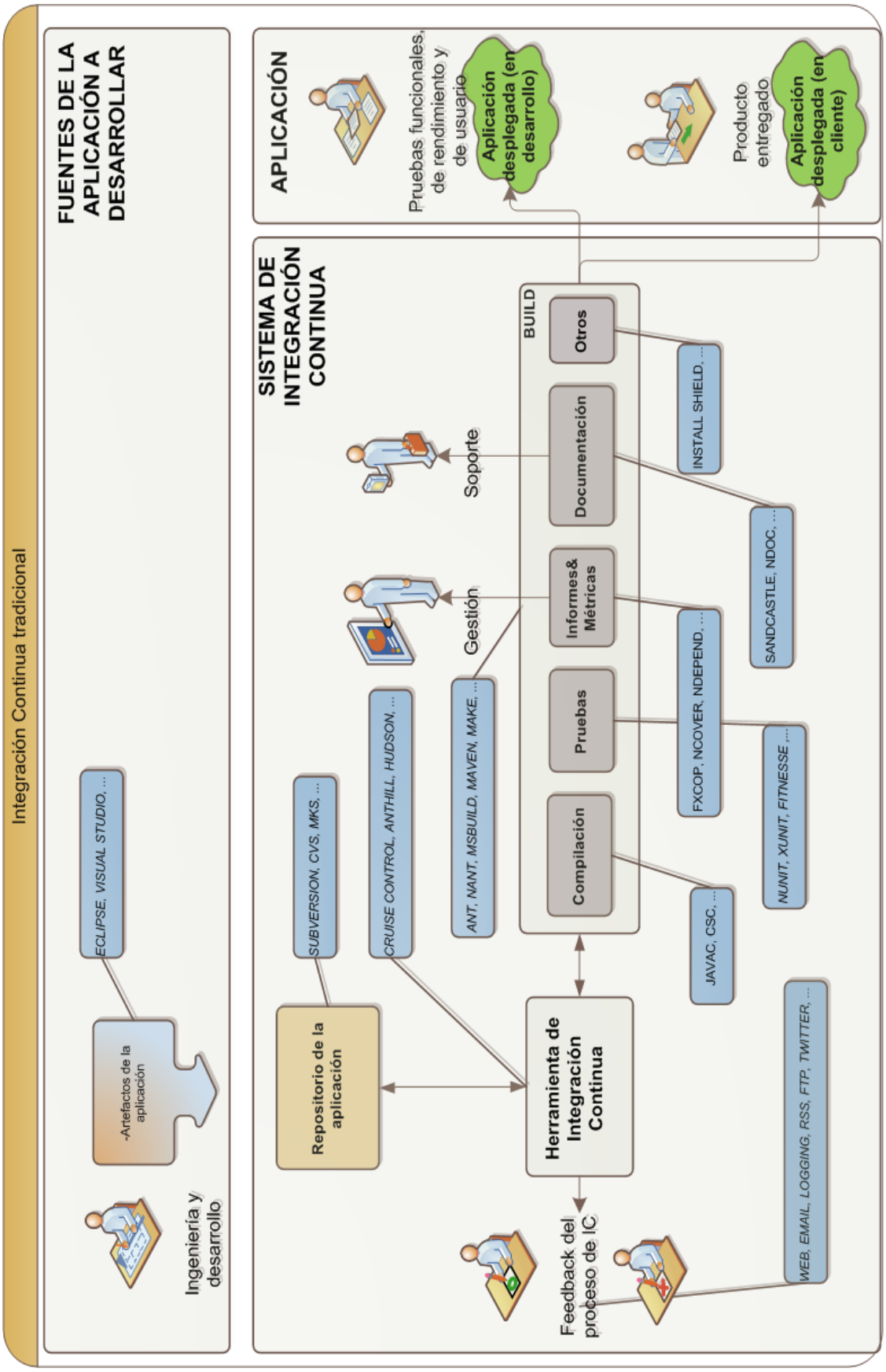


Figura 17.1: Integración Continua tradicional

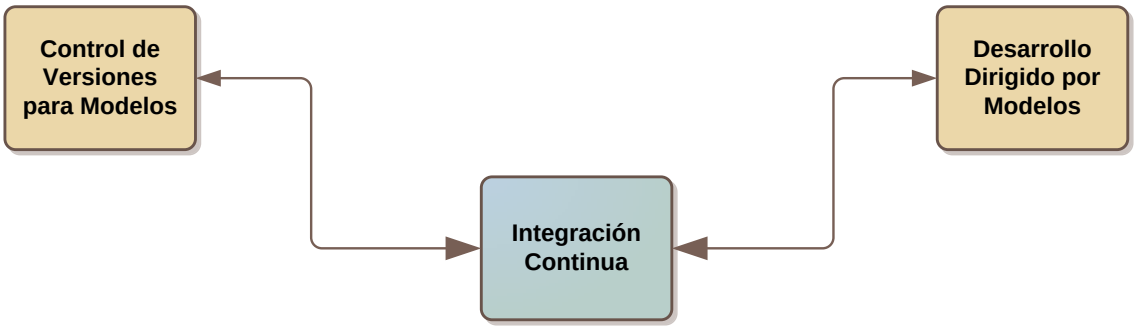


Figura 17.2: MDCI (Model-Driven Continuous Integration)

- Si la generación de artefactos funciona correctamente, el resultado de la etapa de informes y métricas podría enviarse al equipo de gestión, y el resultado de la etapa de documentación podría enviarse a un hipotético equipo de soporte.
- Suponiendo que la construcción ha tenido éxito, el software se despliega en un equipo en el que incluso se podrían hacer más pruebas (no todas las pruebas pueden ser automáticas), como por ejemplo ciertas pruebas de usabilidad.

17.2. Proceso de Integración Continua Dirigida por Modelos

Por un lado, las empresas cada día son más conscientes de la importancia que tiene realizar una buena integración de código debido, entre otros motivos, a la reducción de riesgos y de costes. Este es el motivo de que la práctica y las herramientas de integración continua sean cada día más populares [Duvall et al., 2007].

Por otro lado, las empresas quieren reducir costes, no reinventar la rueda en cada desarrollo y crear productos de calidad. Por ese motivo, la aproximación de desarrollo de software dirigida por modelos está ganando importancia tanto en el mundo académico como en el mundo empresarial [Völter and Stahl, 2006].

En este trabajo se ha realizado por primera vez una unión plena entre CI, MDE y MVCS y es por ello que se puede hablar de una relación directa entre los tres conceptos. Así, se ha elegido el nombre de MDCI (Model-Driven Continuous Integration o Integración Continua Dirigida por Modelos) (Fig. 17.2) como representativo para tales efectos.

Según el PMI (Project Management Institute) [PMI, 2005], un proceso es un conjunto de actividades interrelacionadas realizadas para obtener un conjunto específico de productos, resultados o servicios. Otras definiciones aceptadas son:

- Conjunto de actividades enlazadas entre sí que, partiendo de uno o más *inputs* (entradas), generan un *output* (resultado).
- Serie sistemática de acciones dirigidas al logro de un objetivo.
- Conjunto de actividades que realiza la organización, transformando entradas para crear, producir y entregar sus productos o servicios, de tal forma que satisfagan las necesidades de sus clientes.

Para definir completamente un proceso hay que responder a las preguntas planteadas en la Fig. 17.3[2].

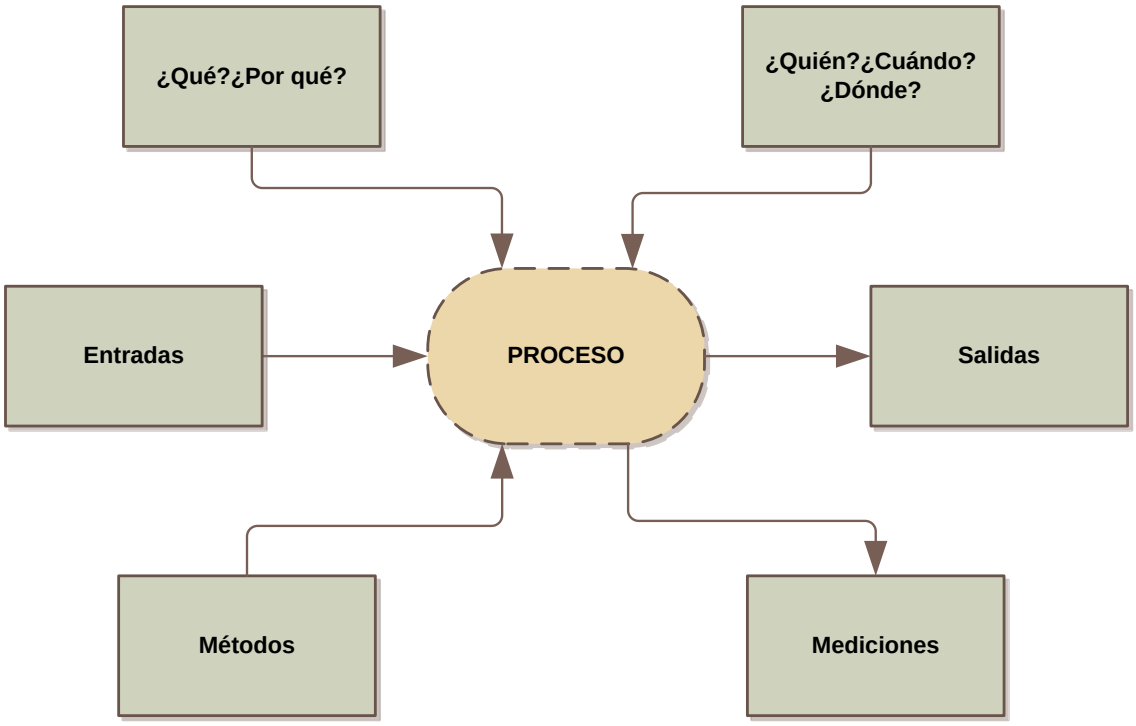


Figura 17.3: Componentes de un proceso

En los siguientes apartados se responderán a las preguntas planteadas para definir un proceso MDCI (Model-Driven Continuous Integration), es decir: ***un proceso en el que se aplique la práctica de la integración continua en un desarrollo de software dirigido por modelos de forma eficiente.***

17.2.1. Proceso MDCI. ¿Quién?¿Cuándo?¿Dónde?

Utilizando TMDE como iniciativa MDE, se apuesta por las líneas o familias de productos (véase capítulo 7). La principal diferencia entre el desarrollo tradicional

[2]http://spincolombia.blogspot.com/2007/08/boletn-no-1-mes-de-julio.html

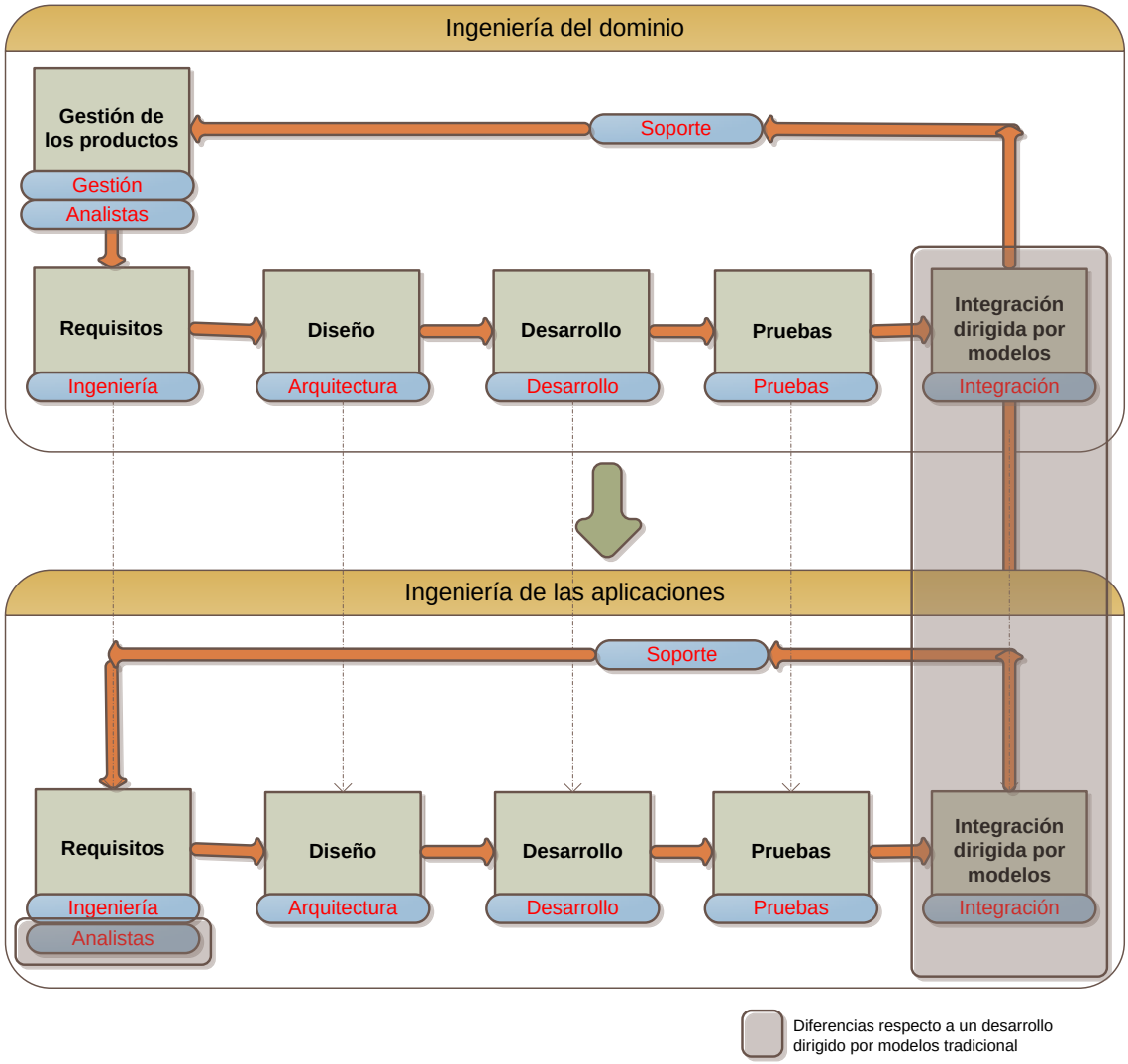


Figura 17.4: Ciclo MDCI

y las líneas de productos es que pasa de centrarse en un producto a centrarse en varios productos. Es entonces cuando se puede hablar del doble ciclo de vida [van der Linden et al., 2007]. La Fig. 17.4 se ha adaptado a partir de dicho ciclo de vida y da respuesta a las tres preguntas: *¿quién?¿cuándo?* y *¿dónde?*. Los recursos humanos que se necesitarán para llevar a cabo todas las tareas son:

- *Gestión*. Hace referencia al cargo que comúnmente se denomina *product manager*. Su misión es planificar y gestionar la evolución de toda la familia de productos. Ello incluye tanto aspectos económicos como funcionales. Sus decisiones serán muy importantes para los responsables del negocio, para el departamento de marketing y de ventas, y para Ingeniería.

- *Analistas.* Son los expertos en el dominio para el cual se realiza la familia de productos software. Ayudan a Gestión a decidir las características, los aspectos comunes y los aspectos variables de una familia de productos. Además, en MDCI (Model-Driven Continuous Integration) tienen la posibilidad de decidir los requisitos de una aplicación y modificarla o crearla de forma transparente para ellos, ya que no tienen o no se puede suponer que tienen conocimientos técnicos más allá que el del manejo de un ordenador a nivel usuario.
- *Ingeniería*
  - del Dominio. Su misión es estudiar los requisitos de la familia de productos como un todo. Ofrece retroalimentación a Gestión con información de la viabilidad (coste, posibilidad, etc.) de implementar las características deseadas.
  - de la Aplicación. Su misión es estudiar los requisitos de cada producto individual. Muchas veces algunas de las características que se introducen en los productos son estudiadas para ser introducidas en todos los productos de la familia.
- *Arquitectura.*
  - del Dominio. Se encarga de diseñar la arquitectura base para todos los productos de la familia. Incluye la selección de componentes reusables e interfaces para ser extendida.
  - de la Aplicación. Basándose en la arquitectura base, se crea una arquitectura para un producto concreto.
- *Desarrollo.*
  - del Dominio. Desarrollan el software para toda la familia de productos en función del diseño creado por Arquitectura.
  - de la Aplicación. Desarrollan el software para un producto en función del diseño creado por Arquitectura.
- *Pruebas.*
  - del Dominio. Tienen como misión crear y realizar pruebas para el conjunto de la familia de productos.
  - de la Aplicación. Tienen como misión crear y realizar pruebas para un determinado producto.

- *Integración.*
  - del Dominio. Preparan la forma en la que se ha de integrar el software. Al trabajar con modelos, deberán seleccionar las herramientas y utilidades necesarias de modo que la fase de integración continua comience únicamente cuando hay un cambio real en la versión de un modelo. Además, la generación de artefactos deberá ser incremental y ofrecer una retroalimentación uniforme respecto a la del resto de actividades que ocurren en la integración continua que no tengan relación con modelos.
  - de la Aplicación. En base a las directrices dadas por el equipo encargado de toda la familia de productos, permiten la integración de cada uno de los productos software. Para ello, necesitarán configurar las herramientas según sus necesidades como producto individual.
- *Soporte.*
  - del Dominio. Preparan documentación, manuales, ayudas, presentaciones y otros artefactos relacionados con la familia de productos.
  - de la Aplicación. Preparan documentación, manuales, ayudas, presentaciones y otros artefactos relacionados con un producto concreto.

17.2.2. Proceso MDCI. ¿Qué?¿Por qué?

Sin entrar en detalles en la forma de gestionar los productos, en la ingeniería de requisitos, en el diseño, en el desarrollo y en las pruebas, en este apartado se indican los recursos necesarios para llevar a cabo el proceso MDCI (Model-Driven Continuous Integration). Dichos recursos no serían necesarios en desarrollos tradicionales.

- *Gestor de integración continua.* Será necesario un sistema que se encargue de automatizar la ejecución de la integración continua de manera transparente a los desarrolladores. De ese modo, los expertos en el dominio podrán hacer comenzar (cuando cambien un modelo con una herramienta de modelado) ciclos de integración continua sin ser necesariamente conscientes de ello. El gestor de integración continua deberá ofrecer retroalimentación de todo lo acontecido durante la generación de artefactos a partir de modelos.
- *Sistema de control de versiones para modelos.* Será necesario un sistema de control de versiones que se encargue de controlar las versiones de los diferentes modelos. Dicho sistema deberá tratar adecuadamente las unidades de

versión de los modelos para evitar falsas detecciones de nuevas versiones y la no detección de nuevas versiones que sí se han producido.

- *Generador de artefactos dirigido por modelos incremental.* Será necesario un generador de artefactos que, a partir de modelos, genere únicamente los mínimos artefactos necesarios para que el impacto sobre la aplicación final sea el mínimo posible. Para lograr los objetivos, también será necesario un subsistema que se encargue de comparar modelos y obtener las diferencias entre ellos.

- *Mapeador de metamodelos personalizados a MOF/Ecore.* Será necesario un sistema que se encargue de mapear a un metamodelo MOF/Ecore los metamodelos que, por el motivo que sea, se hayan utilizado para definir los lenguajes de dominio específico. El objetivo es cumplir con los estándares de la industria software en cuanto al empleo de modelos para generar artefactos.

17.2.3. Proceso MDCI. Entradas y salidas

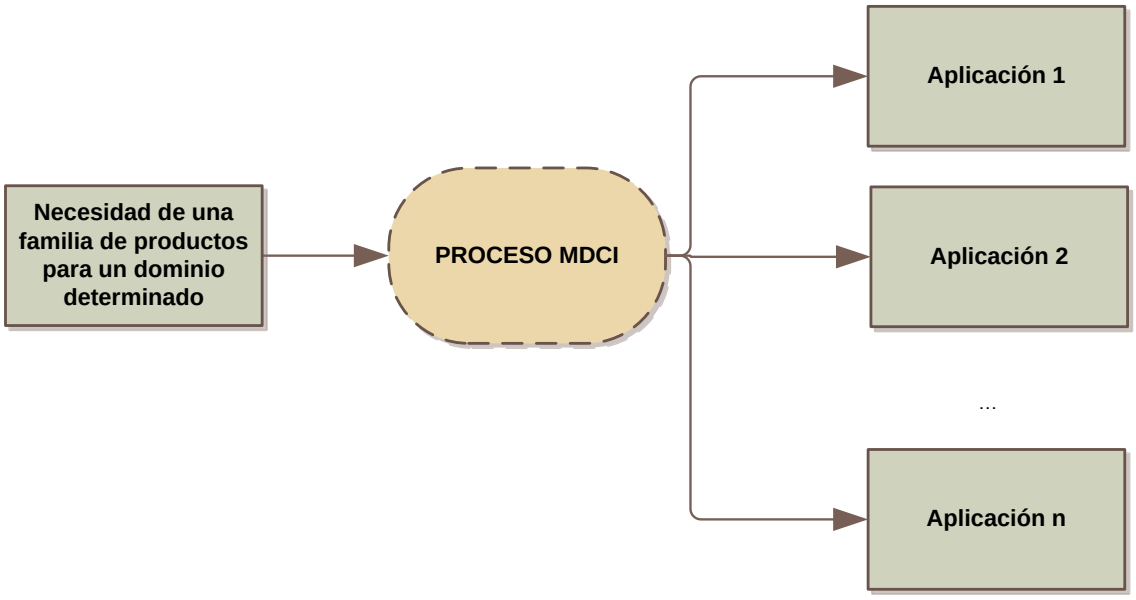


Figura 17.5: Entradas y salidas en MDCI

La entrada es *la necesidad de crear una familia de productos software para un dominio de conocimiento determinado.* La salida obtenida está formada por *las diferentes aplicaciones construidas para los diferentes clientes.* La Fig. 17.5 lo representa.

17.2.4. Proceso MDCI. Métodos

La Fig. 17.6 muestra un aspecto global de cómo es el proceso MDCI (Model-Driven Continuous Integration) propuesto. El origen suele ser el deseo de construir una familia de productos software para un determinado dominio de aplicación. Por lo tanto, se hace una distinción entre los artefactos genéricos, los artefactos específicos y los artefactos esquemáticos repetitivos [Kelly and Tolvanen, 2008]. Para trabajar con modelos, y como se recomienda en [Völter and Stahl, 2006], se utilizan inicialmente dos repositorios:

- *Repositorio del dominio de las aplicaciones.* Dado que podrían existir varias aplicaciones que utilicen una misma plataforma base, un mismo metamodelo para los modelos y unas mismas transformaciones definidas (al igual que ocurre en todas las líneas de productos software [Clements and Northrop, 2002]), sería aconsejable que todas las aplicaciones de una misma familia de productos compartan dicho repositorio, evitándose así problemas de duplicados e inconsistencias.

- *Repositorio específico de la aplicación.* Cada una de las aplicaciones podría tener características variables para las cuales no se emplean modelos y sí técnicas tradicionales como lenguajes orientados a objetos.

Sin embargo, para trabajar con modelos sería recomendable utilizar un tercer repositorio, el *repositorio específico de los modelos de la aplicación.* Dado que los repositorios tradicionales no sirven para trabajar con modelos, es necesario un tercer repositorio, específico de modelos [Altmanninger et al., 2008]. También es importante que antes de introducir el modelo en el repositorio, se realice un mapeo entre la tecnología utilizada y MOF/Ecore para poder trabajar a partir de ese momento con tecnologías estándar (véase capítulo 7).

Cuando la herramienta de CI detecta que ha cambiado el repositorio de los modelos, es cuando el generador MDE construye los artefactos a partir de los modelos. La herramienta de CI debería estar adaptada al generador para poder ofrecer la retroalimentación de su ejecución de manera coherente con el resto de información que pudiera ofrecer de otras ejecuciones.

Posteriormente, existiría un cuarto repositorio[3] en el que se combina la información procedente tanto de los repositorios tradicionales como del generador MDE. A partir de ese momento es en el que se empieza a realizar la etapa de construcción, de igual modo a como se realiza con la integración continua tradicional.

[3]la separación entre los repositorios tiene que ser al menos *conceptual*; no tiene por qué ser *física*

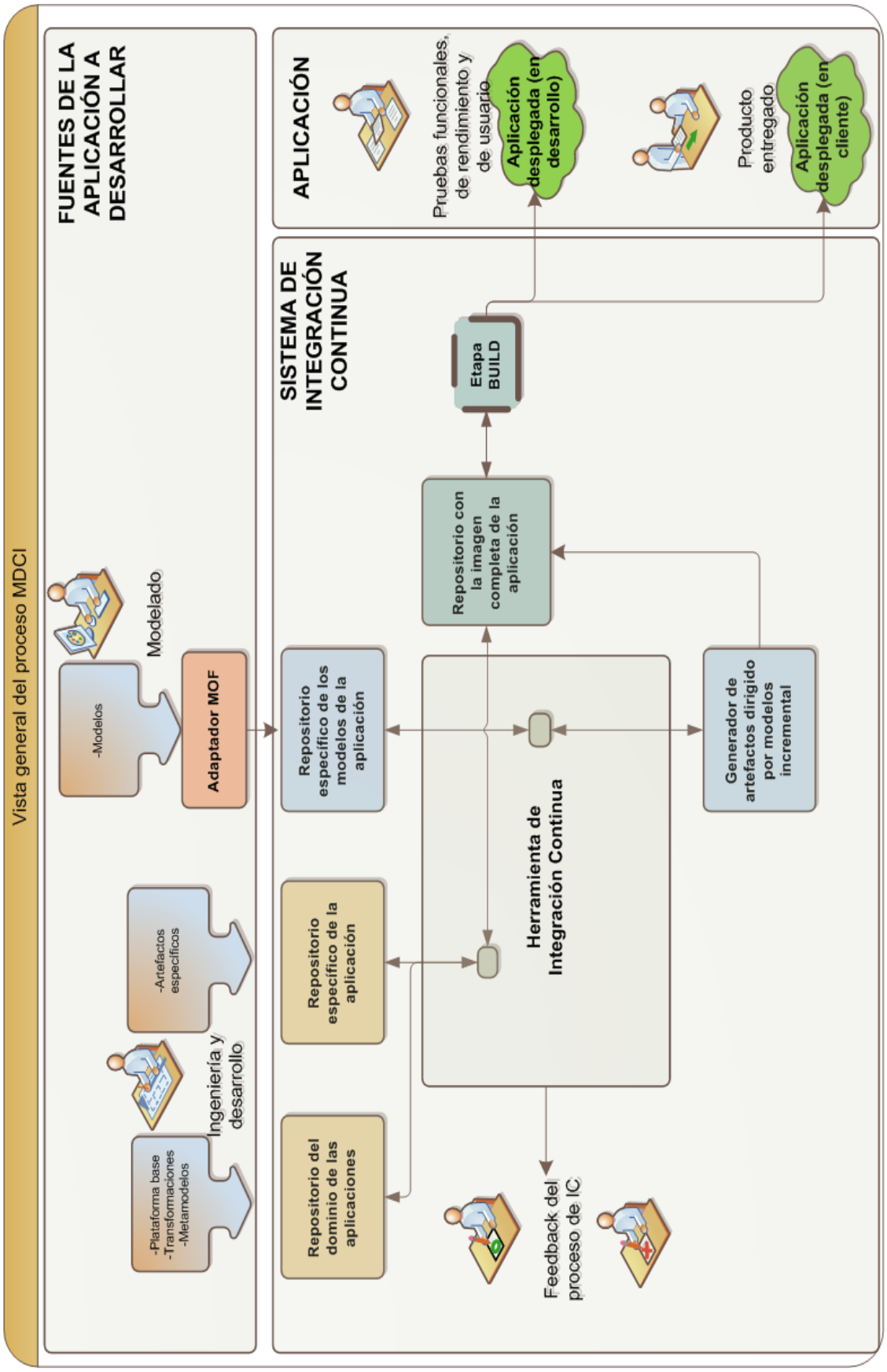


Figura 17.6: Proceso MDCI

Una vez finalizada la integración continua, realizadas pruebas adicionales y dada por concluida la aplicación por parte del cliente, se podría dar permiso a un experto en el dominio de los modelos (generalmente un empleado del cliente que ha encargado la aplicación) para que modificara los modelos que pertenecen a dicho dominio (p.e., un empleado de una fábrica de quesos podría modificar el modelo que se corresponde con el proceso de fabricación de un queso debido a que es él quien realmente conoce cómo se ha de fabricar el queso). El objetivo es que el experto o analista del dominio ni tenga ni pueda cambiar directamente el código generado pero sí pueda, mediante una herramienta de modelado, modificar un modelo y enviarlo al repositorio, desencadenando de nuevo el proceso de integración continua, que concluirá con la aplicación del cliente modificada de manera totalmente transparente para él.

Además, el empleo de herramientas relacionadas con CI posibilita que todo el proceso se realice de forma distribuida. En la Fig. 17.7 se puede ver un ejemplo de los posibles agentes o elementos que intervienen en la modificación por parte de dos expertos en un dominio de una aplicación que ya está funcionando en un cliente. Se observa que hay un total de ocho agentes, todos ellos en ubicaciones físicas diferentes. Esto es fundamental porque los desarrolladores, los gestores, los clientes, las aplicaciones y los equipos raramente estarán en la misma ubicación física.

Pese a que puede resultar confuso que un experto de un dominio (sin conocimientos informáticos) modifique la aplicación, es viable desde el momento en el que la aplicación ya está totalmente integrada, habiendo pasado por todas las pruebas y validaciones (así se garantiza que cambios en los modelos no puedan llevar la construcción a un estado defectuoso). Además, el uso de DSLs permite realizar validaciones y comprobaciones[4] que impiden a un usuario realizar operaciones que semánticamente no tengan sentido (p.e., un DSL de un proceso de fabricación alimentario no debería dejar guardar un modelo que represente un proceso de fabricación que no tenga un nodo final).

En el anexo A pueden verse ejemplos reales del diseño utilizado para permitir una comunicación entre el equipo de desarrollo de las aplicaciones de trazabilidad alimentaria realizadas durante este trabajo [García-Díaz et al., 2008, García-Díaz et al., 2009a] y los productores lácteos. Dicho diseño permitió que los productores (expertos en un dominio no informático) pudieran describir de manera fiable y pre-

[4]En dos niveles: sintaxis abstracta del metamodelo y sintaxis concreta

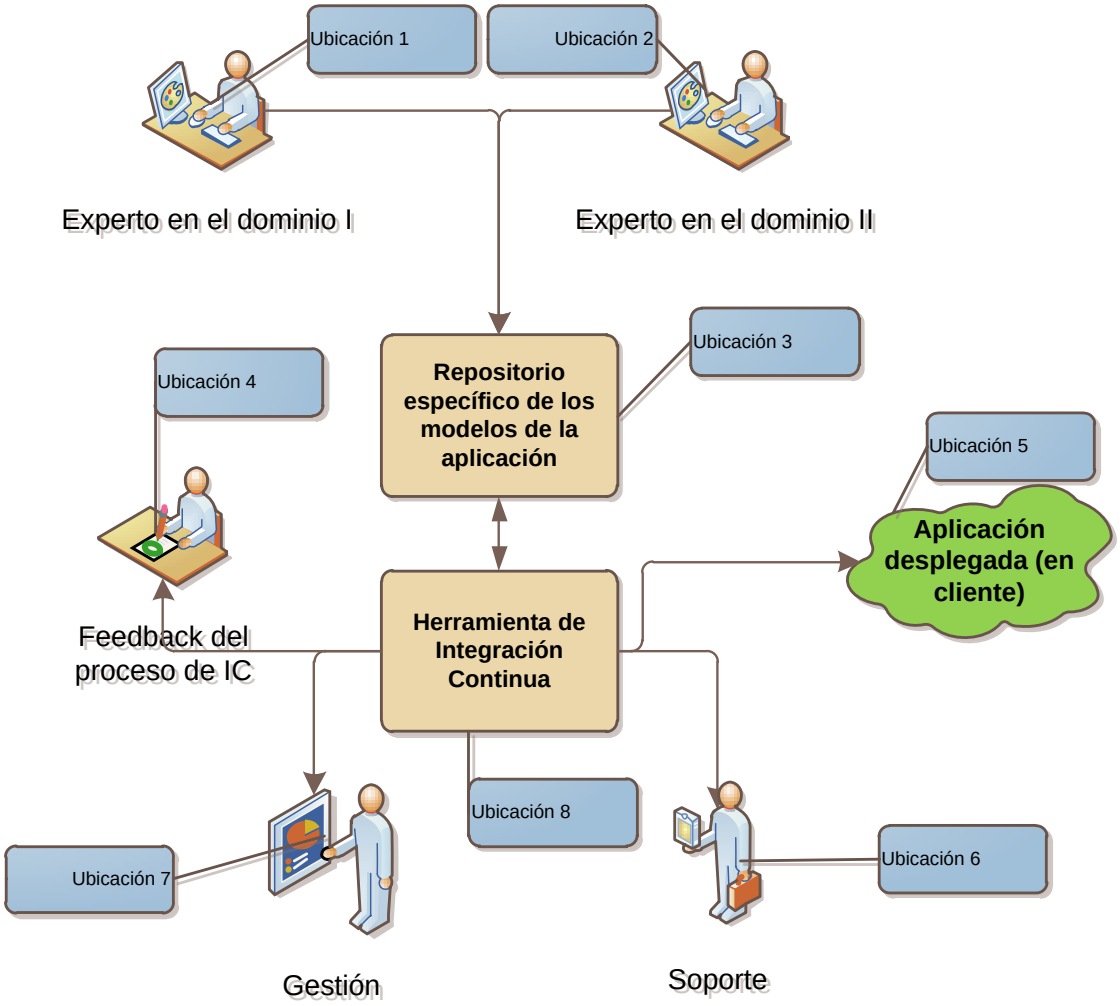


Figura 17.7: Posibles ubicaciones físicas en MDCI

cisa el modo de cómo fabrican sus quesos[5] al equipo de desarrollo de las aplicaciones, lo que quiere decir que con un DSL, de sintaxis y semántica equivalente, ellos mismos podrían modificar, transparentemente para ellos gracias a MDCI (Model-Driven Continuous Integration), su propio proceso de fabricación.

17.2.5. Proceso MDCI. Mediciones

En este apartado se tienen en cuenta las mediciones sobre los objetivos, calidad y resultados:

- *Número de ciclos de integración iniciados.* Debido a que los sistemas de control

[5]En Asturias hay más de 70 tipos de quesos. Además, cada productor de cada tipo de queso emplea matices que suelen hacer que cada proceso de fabricación sea único

de versiones para modelos que funcionan correctamente evitan la detección errónea de cambios de versiones, se evita que se comiencen ciclos de integración innecesariamente.

- *Número de artefactos*. El número de artefactos generados utilizando MDCI (Model-Driven Continuous Integration) deberá ser inferior (o igual en el caso peor) al número de artefactos sin Model-Driven Continuous Integration. Si no fuera así, sería muy improbable que con Model-Driven Continuous Integration se consiga una mejora respecto a los planteamientos tradicionales.

- *Tamaño de los artefactos*. El tamaño de la suma total de artefactos utilizando Model-Driven Continuous Integration deberá ser inferior (o igual en el caso peor) al tamaño de los artefactos sin Model-Driven Continuous Integration. Si no fuera así, el peso de la generación incremental seguiría siendo tan grande como el de la tradicional.

- *Tiempo de despliegue*. El tiempo de despliegue también deberá ser inferior con Model-Driven Continuous Integration. Esto es así porque habrá menos artefactos que desplegar y, por lo tanto, el impacto de cambio sobre aplicaciones ya desplegadas también será menor. Además, por ejemplo, en muchas ocasiones no hará falta eliminar y volver cargar registros en una base de datos, ya que actualizándola directamente se consigue un incremento de velocidad.

- *Tiempo de generación*. Al generar menos artefactos, el tiempo de generación de los artefactos es casi con total seguridad menor con Model-Driven Continuous Integration, aunque no tiene por qué ser así necesariamente debido al tiempo empleado en comparar los modelos. Sin embargo, el tiempo de generación no es un factor crítico, siendo mucho más importante el tiempo de despliegue, que es el que puede afectar directamente a las aplicaciones en producción.

17.3. Conclusiones

Dados los problemas que se han ido planteando a lo largo de esta Tesis y las ventajas que ha ofrecido el empleo de la herramienta MDCIP (Model-Driven Continuous Integration Prototype), el último salto natural es la definición de un proceso MDCI (Model-Driven Continuous Integration). La idea es tener claras las respuestas a las preguntas clave que facilitarán su empleo:

- ¿Quién?¿Cuándo?¿Dónde?

- ¿Qué?¿Por qué?
- ¿Entradas?¿Salidas?
- ¿Métodos?
- ¿Mediciones?

En los próximos capítulos se mostrarán los resultados finales y los anexos del documento.

Bloque VIII

Conclusiones y futuro trabajo

Índice del bloque

CAPÍTULO 18

Conclusiones finales

En esta Tesis, basada en el estudio de viabilidad realizado en [García-Díaz, 2009], se han realizado las siguientes tareas:

1. Se han expuesto los diversos problemas presentados para crear un prototipo MDCI (Model-Driven Continuous Integration) y sus correspondientes soluciones:
   - Elección de una iniciativa MDE (solución adoptada en el capítulo 7 - conclusiones en sección 7.4).
   - Detección de nuevas versiones en los VCSs (solución adoptada en el capítulo 9 - conclusiones en sección 9.4).
   - Integración de MDE con herramientas de CI (solución adoptada en el capítulo 11 - conclusiones en sección 11.3).
   - Generación incremental de artefactos (solución adoptada en el capítulo 13 - conclusiones en sección 13.7).
2. Se ha desarrollado una herramienta prototípica, que permite hacer de Model-Driven Continuous Integration una realidad (véase capítulo 14 - conclusiones en sección 14.6).
3. Se han estudiado diversos ámbitos de aplicación en los que podría aplicarse el prototipo desarrollado y, por tanto, Model-Driven Continuous Integration (véase capítulo 15 - conclusiones en sección 15.4).
4. Se ha realizado una evaluación, mediante el empleo de varios ámbitos de aplicación, que muestra las ventajas del uso de Model-Driven Continuous Integration frente a otros tipos de desarrollos (véase capítulo 16 - conclusiones en sección 16.8).

5. Gracias a todo ello, se ha podido *cumplir* con el objetivo global de la investigación, es decir:

> Llevar a cabo un proceso en el que se aplique la práctica de la integración continua en un desarrollo de software dirigido por modelos de forma eficiente. Así se conseguirán dos **sub-objetivos** principales:
>
> *a*) ***Permitir que los expertos tecnológicos apliquen integración continua durante el desarrollo***. Que los diferentes miembros del equipo de desarrollo puedan integrar de manera continua y distribuida sus desarrollos dirigidos por modelos (al igual que lo pueden hacer mediante la programación tradicional).
>
> *b*) ***Permitir que expertos en un dominio de conocimiento modifiquen una aplicación***. Que diferentes expertos en un dominio modifiquen de manera distribuida modelos de dicho dominio, provocando que las aplicaciones se modifiquen dinámica y transparentemente para ellos. Así, se permitiría que sean los expertos en el dominio de conocimiento los que creen, desplieguen o cambien las aplicaciones en función de las necesidades del negocio del cliente.

Con el trabajo realizado (mediante el prototipo desarrollado), el equipo de desarrollo podrá comenzar a trabajar en desarrollos dirigidos por modelos como si de desarrollos tradicionales se tratara. Podrá trabajar concurrentemente con repositorios de modelos en los que se detectarán correctamente los cambios entre diferentes versiones. Los artefactos generados irán evolucionando con los cambios de los modelos, evitando tener que regenerar todos los artefactos con cada modificación y adaptándose la generación al estado del sistema en cada momento. Por otra parte, el cliente o usuario final del sistema construido, si tiene los conocimientos suficientes del dominio de conocimiento, podrá hacer cambios en su aplicación de forma transparente para él y con el mínimo impacto posible. Además, la herramienta de integración continua se encargará de regenerar los artefactos involucrados en cada cambio, desplegándolos si así fuera necesario.

Resumiendo, algunas de las principales ventajas o posibilidades ofrecidas por MDCI (Model-Driven Continuous Integration)[1] frente a desarrollos MDE tradicionales son:

- Posibilidad de cambio concurrente por parte de varias personas de los modelos, los cuales son fuentes en los desarrollos MDE.

[1] Basado en estándares e independiente de la herramienta utilizada

- Trazabilidad de la evolución de los modelos de igual forma que se puede hacer de la evolución del código de lenguajes de programación tradicionales (p.e., Java) utilizando sistemas de control de versiones como Subversion.
- Trazabilidad de la evolución de los artefactos generados con cada cambio de versión de los modelos, lo cual es una poderosa herramienta de control y de depuración.
- Generación de artefactos automática cuando se detecta que hay un cambio en los repositorios de los modelos.
- Generación de artefactos de forma incremental y adaptable, de tal forma que únicamente se generen los artefactos estrictamente necesarios para cada cambio en el modelo, provocando que se puedan generar diferentes artefactos para un mismo elemento del modelo en función de si éste se ha añadido, eliminado, cambiado o si permanece intacto.
- Realización automática de acciones personalizadas con los artefactos generados a partir de modelos (p.e., compilación, despliegue, generación de documentación).
- Obtención de retroalimentación a partir del sistema de control de versiones para modelos y del generador incremental de artefactos de forma consistente con la retroalimentación obtenida del resto de herramientas que se pueden utilizar en desarrollos tradicionales (p.e., NUnit, NCover).
- Integración total de las herramientas dirigidas por modelos con las herramientas de integración continua, lo que conlleva a la obtención de las ventajas ofrecidas por la integración continua dentro del ámbito de la ingeniería dirigida por modelos.
- Posibilidad de que expertos en un dominio, sin conocimientos técnicos informáticos, sean capaces de modificar una aplicación ya desplegada, únicamente por el hecho de modificar un modelo.

Las ventajas derivadas del cumplimiento con el objetivo son claras pero, pese a ello, todavía hay un gran camino que recorrer en cuanto a las líneas de investigación que ofrece la aproximación de desarrollo de la ingeniería dirigida por modelos y la práctica de la integración continua (véase capítulo 19). Así, Model-Driven Continuous Integration debería seguir madurando de forma paralela a MDE y CI.

CAPÍTULO 19

Líneas de investigación abiertas

En este capítulo se expondrán líneas de investigación que pueden ser explotadas en un futuro y que han surgido durante la realización de esta Tesis Doctoral:

19.1. Profundizar en el proceso MDCI

MDCI (Model-Driven Continuous Integration) es un término genérico utilizado para referirse a todo lo relacionado con la unión entre la integración continua y la ingeniería dirigida por modelos. Una de las metas de este trabajo ha sido definir un proceso Model-Driven Continuous Integration respondiendo a las principales preguntas que ha de responder todo proceso[1]:

- ¿Quién? ¿Cuándo? ¿Dónde?
- ¿Qué? ¿Por qué?
- Entradas y salidas.
- Métodos.
- Mediciones.

Sin embargo, podría ser interesante seguir profundizando en la definición del proceso Model-Driven Continuous Integration. Para ello se podría:

- Dar más detalles y profundizar más en la respuesta a las anteriores preguntas.
- Definir el proceso Model-Driven Continuous Integration en base a los atributos citados en otros trabajos como [Muller, 2008], los cuales podrían ser: propósito, estructura, base lógica, roles o secuencia. Estos atributos responden en gran

[1] http://spincolombia.blogspot.com/2007/08/boletn-no-1-mes-de-julio.html

parte a las mismas preguntas ya realizadas, pero podrían añadir información extra, interesante desde un punto de vista quizá algo más abstracto.

- Definir textualmente un procedimiento en base al proceso descrito[2].

19.2. Realizar más casos de prueba

Para la confección de este trabajo se han realizado los siguientes casos de prueba:

- Aplicación *TheBeerHouse* realizada con el framework ASP.NET MVC (véase sección 16.2.1).
- Aplicación *Simple Website Software* realizada con PHP (véase sección 16.2.2).
- Aplicación *Tweeterati* realizada con WPF (véase sección 16.2.3).

Aunque para los objetivos del trabajo se entiende que las pruebas realizadas son suficientes para mostrar las ventajas que proporciona Model-Driven Continuous Integration, sería interesante realizar más casos de prueba en diferentes ámbitos y dominios profesionales con el objetivo de realizar estudios cuantitativos y cualitativos sobre las diferencias entre utilizar técnicas tradicionales o ideas más avanzadas como las de CI, MDE o MDCI (Model-Driven Continuous Integration).

19.3. Profundizar en la iniciativa TMDE

TMDE [García-Díaz et al., 2010a], es una iniciativa para acometer desarrollos MDE tratando de aunar en un único desarrollo los puntos fuertes del estándar MDA [Miller et al., 2003] y de las Software Factories [Greenfield and Short, 2003]. En [García-Díaz et al., 2010a] se explican los aspectos generales de la nueva iniciativa y se muestra un caso real de aplicación basado en el trabajo realizado para desarrollar sistemas de trazabilidad alimentaria mediante la aproximación de desarrollo MDE. En dicho trabajo, se sugieren las ventajas obtenidas utilizando la nueva iniciativa. Sin embargo, existen muchos aspectos que han quedado sin tratar o han sido tratados muy levemente:

- Profundizar en el mapeo entre metamodelos no estándares y metamodelos MOF, ya que existen investigaciones para tratar de lograrlo [Bézivin et al., 2005b,Bézivin et al., 2005a], pero aún es un tema poco tratado por la literatura científica.

[2]La relación entre un procedimiento y un proceso es que el procedimiento engloba al proceso puesto que su misión es básicamente responder a la pregunta *¿cómo se lleva a cabo el proceso?*

- Estudiar el gasto de recursos que supone crear el mapeo entre metamodelos, en relación al tamaño total de un proyecto, en los casos en los que no se hubiera realizado ya dicho mapeo.

- Realizar más casos de prueba de aplicación simultánea de MDA, Software Factories y TMDE con el objetivo de estudiar cuantitativa y cualitativamente tiempos de desarrollo, dificultad y las características en las que se basan las principales iniciativas MDE (interoperabilidad, portabilidad, reusabilidad, productividad, nivel de abstracción, etc.).

19.4. Mejorar la herramienta MCTest

MCTest es una herramienta que permite evaluar algoritmos de comparación de modelos. Sirve para mejorar la precisión de las herramientas en determinar cuándo los elementos de diferentes modelos se corresponden entre ellos y cuándo no. Ello ofrece una gran cantidad de beneficios entre los que destacan el aumento del porcentaje de acierto para especificar si se produce una nueva versión de un modelo, muy útil en los sistemas de control de versiones para modelos, y la posibilidad de generar artefactos de manera incremental de forma mucho más eficiente.

En cuanto a los algoritmos, el siguiente paso será implementar la interfaz *IMatchEngine* de EMF Compare con el objetivo de obtener un algoritmo que trate de solventar los 7 casos estudiados en los que el algoritmo genético no se ha comportado correctamente (véase sección 9.3.1), sin que por ello el algoritmo pierda precisión con otros casos genéricos o eficiencia respecto al rendimiento ofrecido por defecto por EMF Compare. Además, se dejará abierta la posibilidad de probar algoritmos creados con otras herramientas diferentes a EMF Compare (p.e., SiDiff [Schmidt and Gloetzner, 2008]), ya que la herramienta es totalmente independiente de la tecnología subyacente.

En lo que respecta al alcance de la herramienta MCTest, el siguiente salto consistirá en ofrecer información del operador *diff* y no sólo del operador *match*, como en el caso de prueba estudiado. De ese modo, se ofrecerá mucha más información a los diseñadores de algoritmos de comparación de modelos.

Respecto a MCTest, un futuro trabajo crucial consistirá en proporcionar un *plug-in* que permita integrar la herramienta dentro de la plataforma Eclipse, facilitando así su uso mediante una interfaz gráfica en lugar de programática. Así, una futura línea de investigación consistirá en facilitar a los desarrolladores de algoritmos, la especificación de la solución óptima para cada caso de estudio a través de un entorno

gráfico, siendo un paso previo a que la herramienta sugiera al usuario un algoritmo para cada caso concreto en base al empleo de inteligencia artificial.

Por otra parte, también se mostrará en la salida (por defecto una hoja Excel) información sobre los recursos computaciones utilizados por cada algoritmo, aspecto que puede ser importante a la hora de valorar una implementación. De cara a la salida obtenida, también pretendemos ofrecer una comparativa más profunda entre los diferentes algoritmos mediante el empleo de gráficas.

Las denominadas 3-Way differences [Ohst et al., 2003] no se han tenido en cuenta en este trabajo porque la comparación utilizando un tercer documento, predecesor de los otros 2, está más orientada a la aparición de diferentes ramificaciones en el historial de un modelo mantenido por un sistema de gestión de versiones. Sin embargo, podría ser interesante como ampliación de MCTest en un futuro trabajo.

19.5. Adaptar los disparadores de las herramientas de CI

En [García-Díaz et al., 2010b] se ha mostrado un caso de estudio en el que se observa que los algoritmos de los disparadores actuales que se comportan bien en relación al número de construcciones iniciadas, se comportan mal en cuanto al tiempo que dejan a las fuentes sin integrar. Ello es debido a que los algoritmos no pueden saber a priori cuándo los desarrolladores van a realizar modificaciones en los repositorios. Los propios desarrolladores podrían ser los encargados de comenzar el proceso de integración pero ello requeriría de un trabajo manual o de una adaptación *ad-hoc* de cada sistema de control de versiones, lo cual no parece una solución óptima.

En este aspecto, el trabajo que se podría realizar se centra en, mediante técnicas de aprendizaje automático [Carbonell et al., 1984], proponer un algoritmo que evolucione según los hábitos de cada equipo de desarrollo, buscando una optimización de los parámetros citados.

19.6. Desarrollar herramienta MDCI

Para la realización del prototipo presentado en este trabajo ha sido necesaria la interacción simultánea de aproximadamente 30 aplicaciones informáticas, habiendo sido construidas algunas de ellas expresamente para este trabajo. Sin embargo, las herramientas previamente existentes tienen una naturaleza muy diversa y están concebidas para fines muy heterogéneos. Configurar todas las herramientas para que

funcionen es una tarea difícil que requiere ser realizada por personas con conocimientos en tecnologías muy diversas.

Así, sería muy interesante evitar o reducir el esfuerzo de los usuarios mediante la construcción de una herramienta con una arquitectura modular y extensible que permita ser utilizada, configurada y extendida de manera uniforme y sencilla. Obviamente, este trabajo presentará dificultades como la incontable cantidad de combinaciones de software que puede ser utilizada en cada desarrollo llevado a cabo por una organización. Sin embargo, no dejaría de ser interesante una homogeneización de la interacción entre herramientas. Quizá un aspecto interesante sería un esfuerzo por estandarizar o hacer interoperables las sintaxis de los lenguajes de *workflows* de actividades como Ant, NAnt, MSBuild o MWE.

19.7. Emplear modelos ejecutables con MDCI

Pese a que prácticamente todavía no se utiliza en el mundo empresarial, en los últimos años y de la mano de MDA, ha ido adquiriendo importancia xUML (Executable UML) [Mellor and Balcer, 2002]. Se trata de un perfil UML que permite especificar gráficamente un sistema. El modelado del sistema se hace de tal forma que, mediante el empleo de un lenguaje denominado *action language*, se permite que los propios modelos sean ejecutables directamente en una máquina.

Sería interesante estudiar la forma en la que se puede aplicar xUML en el contexto de MDCI (Model-Driven Continuous Integration). Así, se podrían cambiar dinámicamente aspectos de las aplicaciones sin necesidad de realizar nuevas generaciones de artefactos (posiblemente con un mejor rendimiento que cuando el modelo se ha de transformar en código Java o C#, por ejemplo). También, para lograr trabajar con aplicaciones de cierta envergadura, será necesario estudiar la forma en la que xUML podrá interactuar con lenguajes tradicionales.

19.8. Desarrollar plataforma para sintaxis concretas multimodales

Existen muchos trabajos que centran sus objetivos en estudiar los lenguajes de dominio específico. Muchos de ellos como [Langlois et al., 2007], se centran en la clasificación de los DSLs. Atendiendo a la sintaxis concreta, que es lo que el usuario final entiende y utiliza, existen varias posibilidades como utilizar sintaxis gráficas [Cook et al., 2007], sintaxis textuales [Efftinge and Völter, 2006] o incluso otros tipos de sintaxis menos utilizadas como tablas, árboles, *wizards* o combinaciones de las

anteriores [Tolvanen, 2008]. Lo cierto es que trabajando con un meta-metamodelo estándar como MOF [OMG, 2005a] se consigue tener una base común para todos los DSLs en términos de su sintaxis abstracta. A partir de ella, se podrían definir diferentes sintaxis concretas que mantuvieran la misma sintaxis abstracta y misma semántica asociada [Völter and Stahl, 2006].

Por lo tanto, la idea de esta línea de investigación sería definir un mecanismo mediante el cual se estudie la forma de automatizar el mapeo entre diferentes tipos de sintaxis concretas. De esa forma, a partir de una representación concreta de un lenguaje se podrían crear otras, de forma directa o semi-directa. Si se lleva a cabo esta línea de investigación podrían obtenerse beneficios como:

- Reaprovechar definiciones de sintaxis concretas ya realizadas.
- Permitir que los desarrolladores de los DSLs utilicen las sintaxis concretas que ellos prefieran.
- Realizar cambios de forma rápida ante peticiones de los clientes.
- Permitir hacer pruebas entre los potenciales usuarios del DSL, de tal forma que se pueda elegir, para cada caso y sin coste adicional, el tipo de sintaxis más adecuada para cada dominio y para cada grupo de usuarios.

19.9. Generar automáticamente DSLs a partir de ontologías

El diseño de los DSLs no ha sido el centro de muchas investigaciones, debido a que generalmente éstas se centran en cómo implementar el lenguaje y no en cómo diseñarlo. Para diseñar un DSL, primero hay que analizar el dominio y sobre todo entenderlo. Sin embargo, trabajos como [Mernik et al., 2005] indican la falta de técnicas establecidas para realizar análisis de dominios en el proceso de definición de un DSL (únicamente existe alguna como FAST [Pohl et al., 2005] o FODA [Kang et al., 1990]). Al mismo tiempo, se observa que hay muchas similitudes entre las necesidades del diseño de una ontología y de un metamodelo de un DSL[3]:

- Delimitar claramente el alcance del dominio.
- Determinar los elementos del dominio y sus relaciones.

[3]Históricamente las ontologías han sido utilizadas en el ámbito de la inteligencia artificial mientras que los metamodelos de los DSLs han sido utilizados más frecuentemente en el ámbito de la ingeniería del software

- Definir los conceptos que representan los elementos.
- Crear modelos de características describiendo las partes comunes y variables de los distintos modelos que puedan realizarse dentro del dominio.

Según [Chandrasekaran et al., 1999], una ontología puede servir para determinar los elementos, las relaciones y los conceptos de un dominio y, por tanto, es una buena base para crear un DSL. En [Tairas et al., 2008] se estudia, mediante un caso real, la forma en la que una ontología sobre un determinado dominio puede ayudar a identificar los elementos que formarán parte de un lenguaje específico para dicho dominio. El problema existente en la actualidad es la falta de automatización en las transformaciones entre una ontología y un DSL, ya que se realizan manualmente siguiendo unos patrones determinados.

Sería interesante investigar en procesos no dependientes de la ontología, que realicen transformaciones de manera automática o semi-automática. El desarrollo de esta línea de investigación permitiría realizar DSLs más fácilmente gracias a la reutilización del conocimiento contenido en las ontologías ya definidas durante los últimos años (y otras que se desarrollarán en el futuro). De ese modo, se reutilizarían eficientemente modelos como punto inicial del proceso de integración continua en MDE.

Bloque IX

Anexos

Índice del bloque

APÉNDICE A

Procesos de fabricación

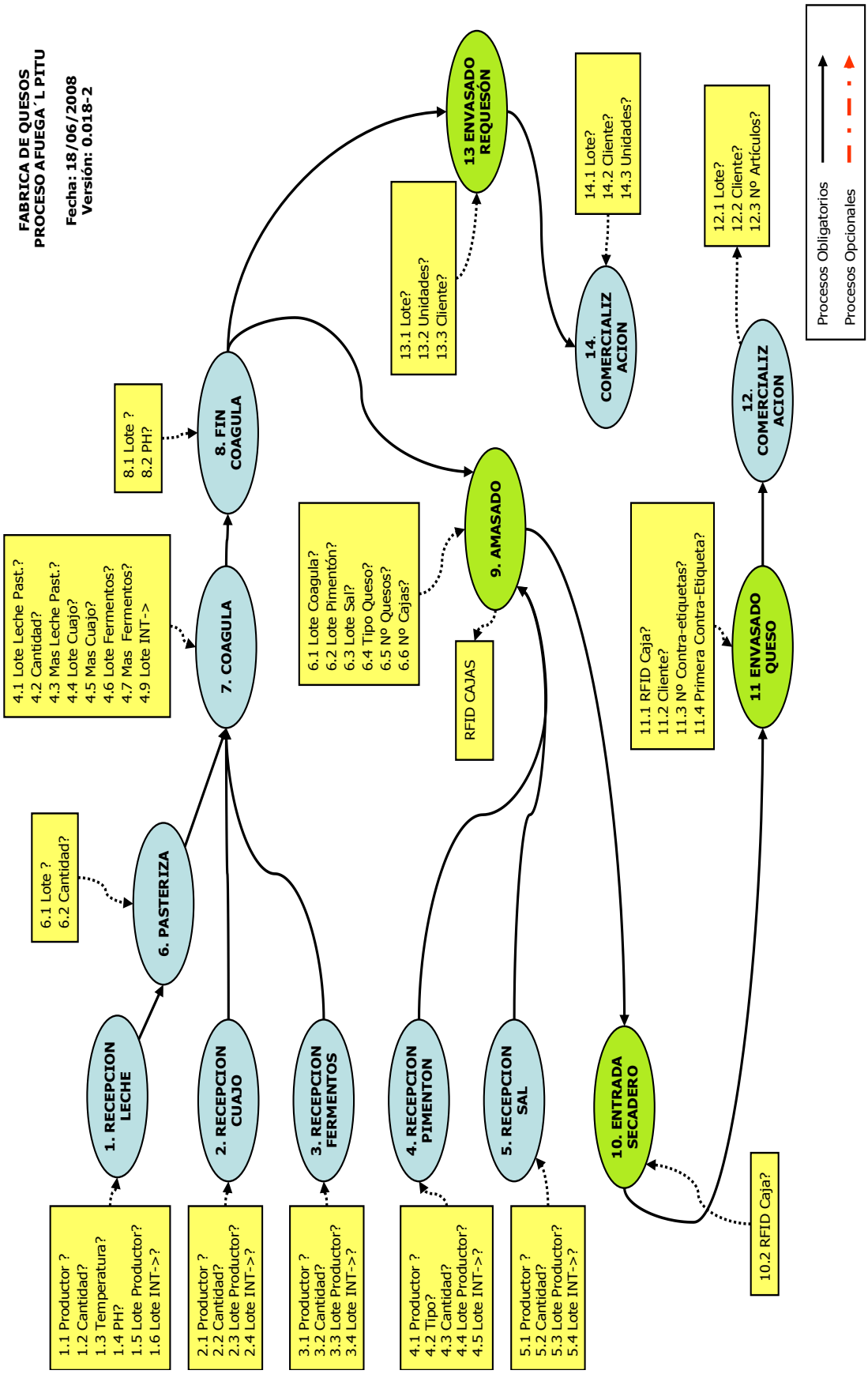


Figura A.1: Queso Afuelga'l pitu

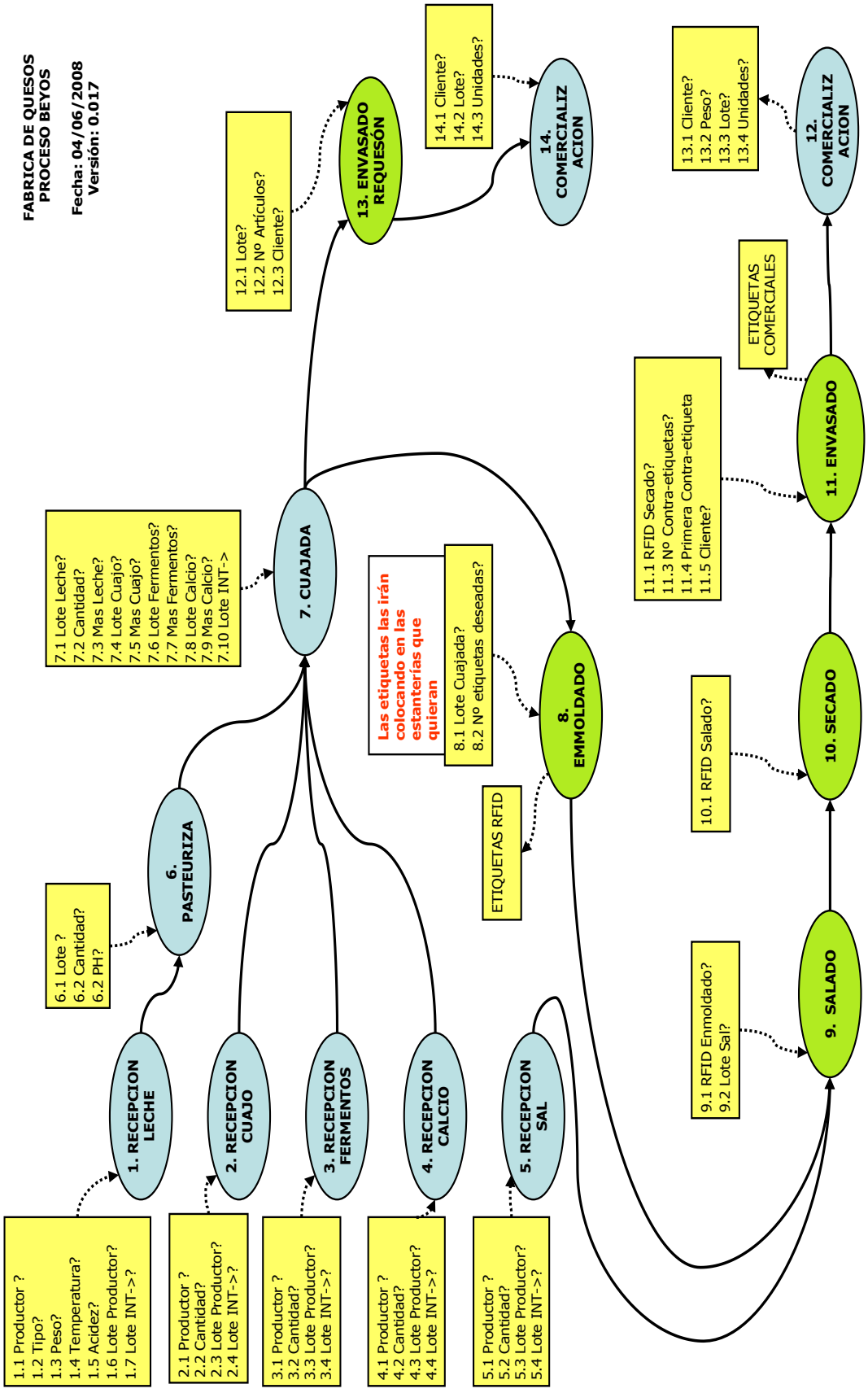


Figura A.2: Queso Beyos

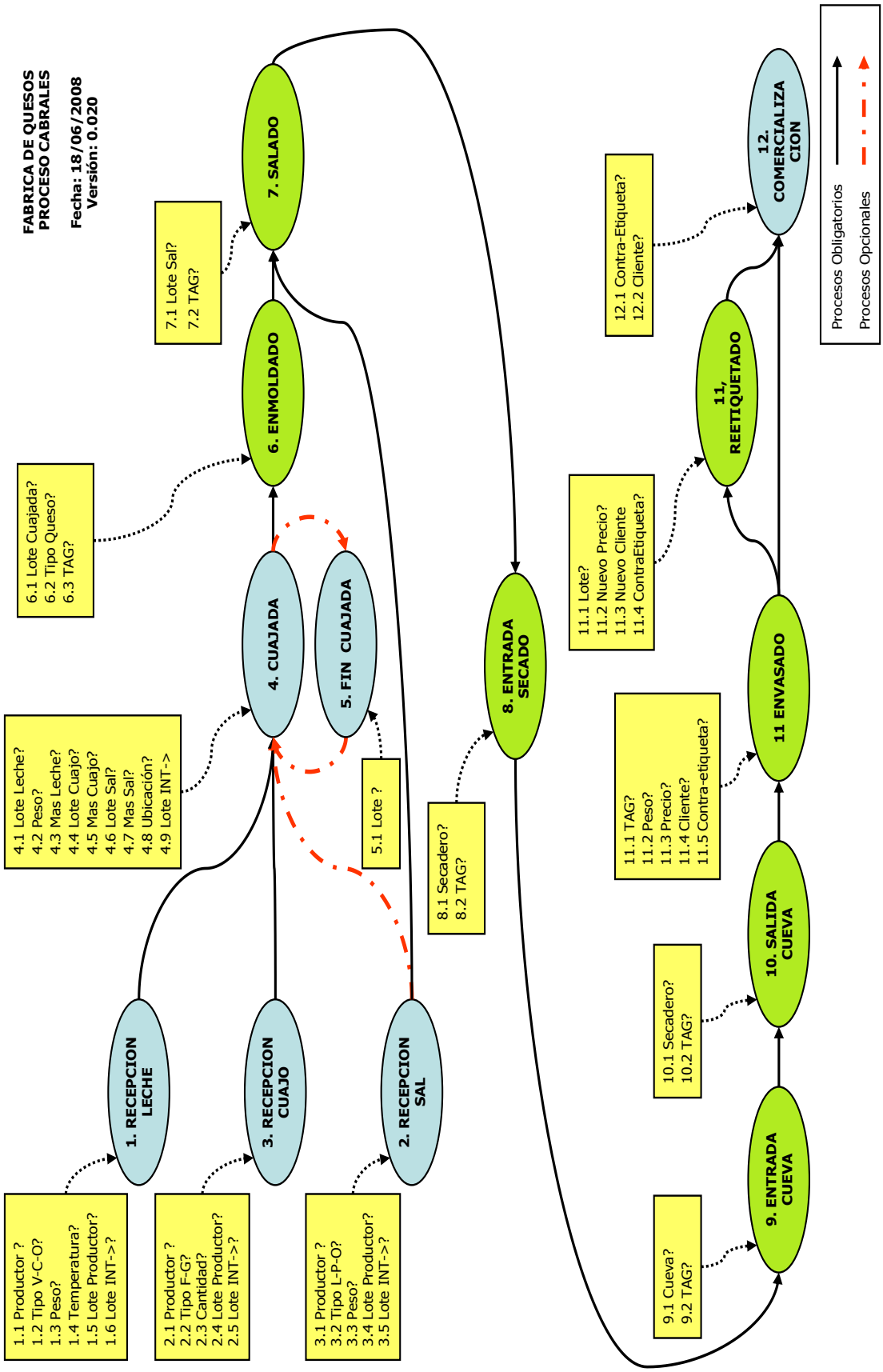


Figura A.3: Queso Cabrales

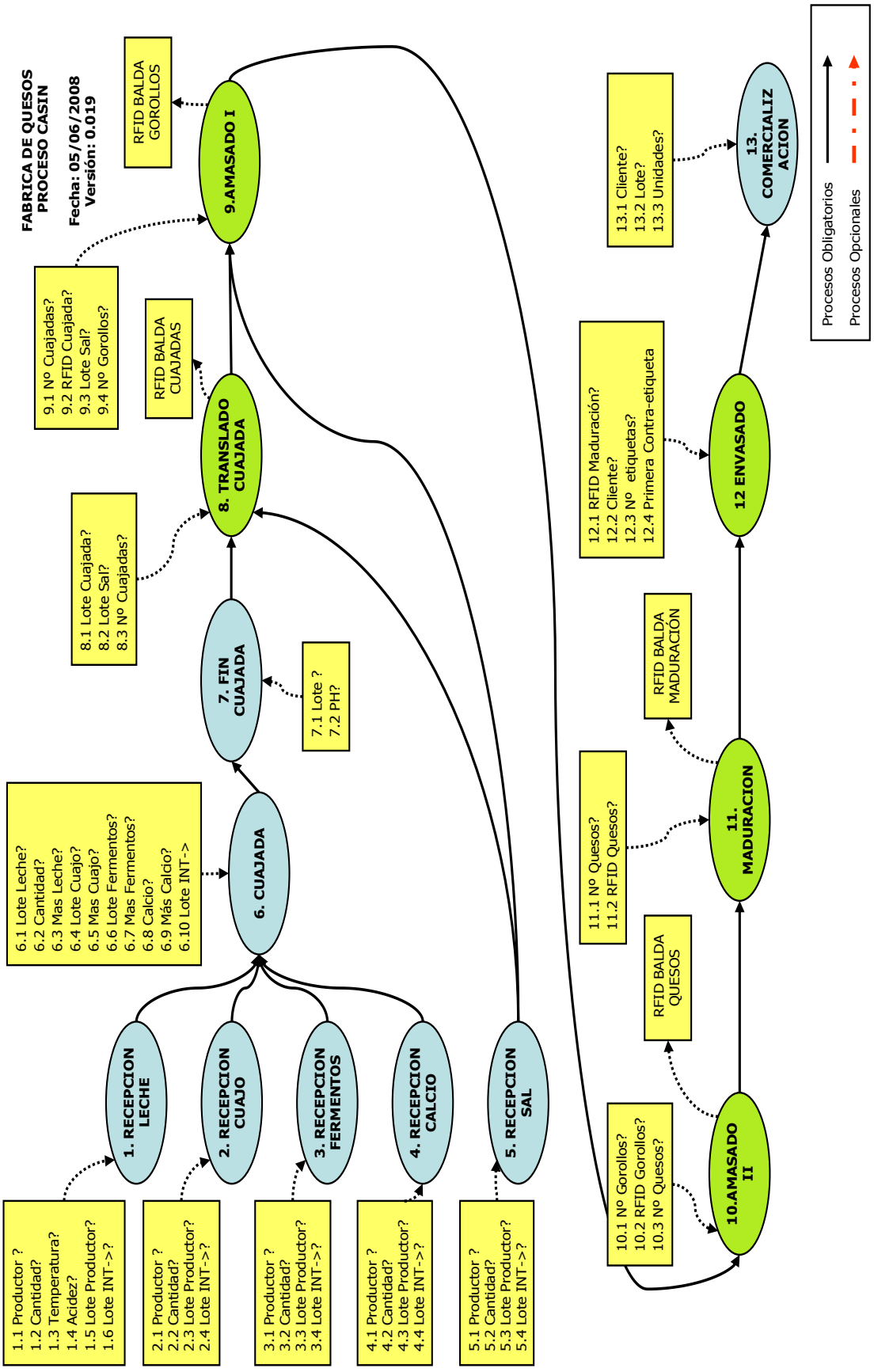


Figura A.4: Queso Casín

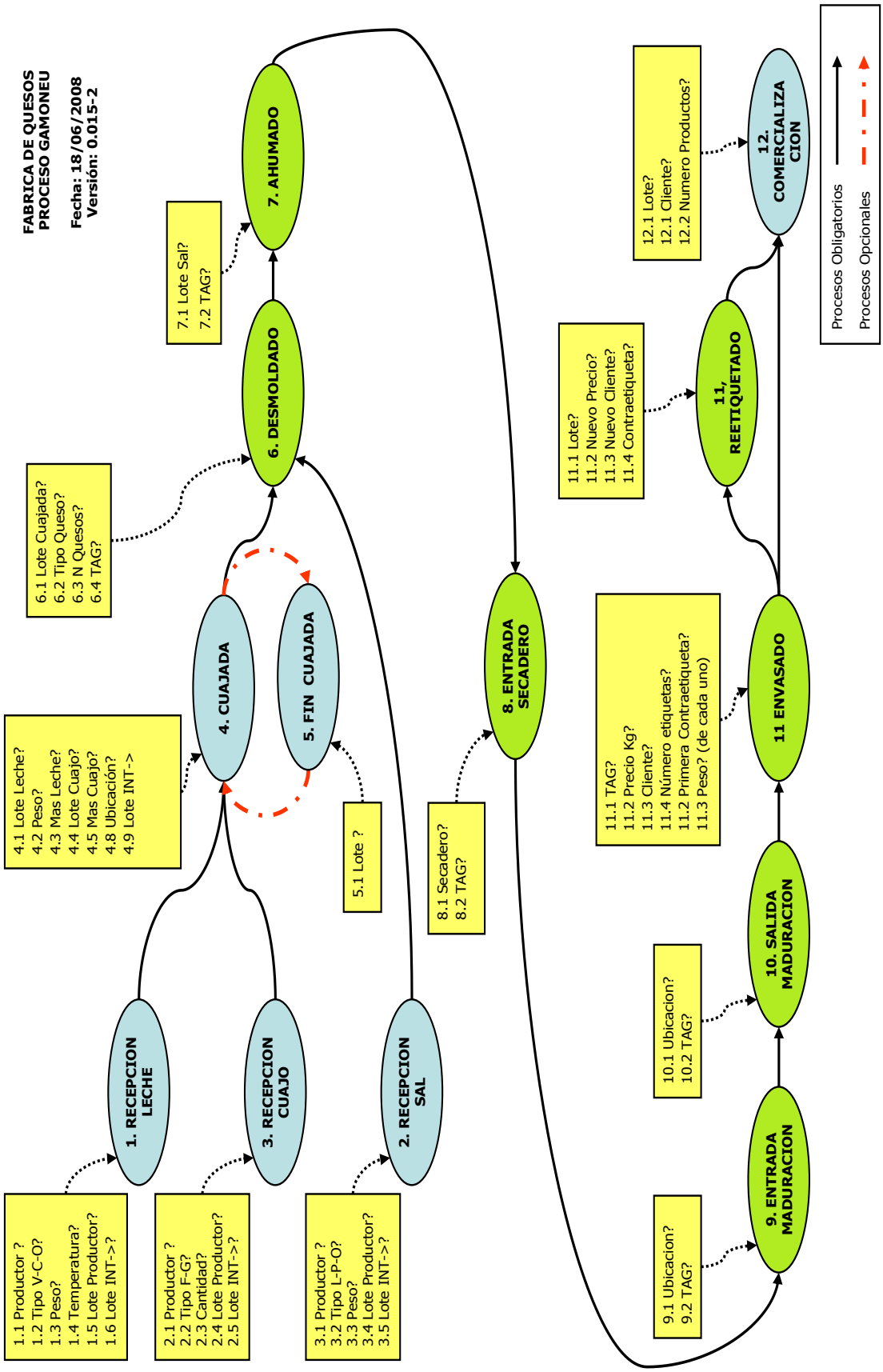


Figura A.5: Queso Gamoneu

APÉNDICE B

Software utilizado para construir la herramienta MDCIP

Para completar la herramienta MDCIP (Model-Driven Continuous Integration Prototype), se ha utilizado simultáneamente diverso software que se enumerará a continuación[1]:

- AnkhSvn v2.1.6941.125. *Plug-in* para permitir utilizar Subversion dentro de Visual Studio.
  http://ankhsvn.open.collab.net/
- Appcmd. Herramienta de línea de comandos que sirve para trabajar con el servidor Web IIS (Internet Information Server) 7.0.
  http://technet.microsoft.com/en-us/library/cc772200(WS.10).aspx
- ***CCNet.mde.plugin***. *Plug-in* para la herramienta CCNET (Cruise Control .NET):
  1. ***ModelVersionControl***. Permite a CCNET trabajar con el prototipo de sistema de control de versiones para modelos denominado *ModelVersionControlSystem*.
  2. ***SynchronizeRepositories***. Permite a CCNET realizar la tarea de sincronizar los cambios que ocurren en los artefactos de los tres repositorios primarios (repositorio del dominio de las aplicaciones, repositorio específico de la aplicación y repositorio específico de los modelos de la aplicación) en el repositorio de la imagen completa de la aplicación, de modo que la operación de construcción de artefactos se realice una vez que todos los artefactos estén sincronizados.

[1]En negrita está el software desarrollado expresamente para la realización de este trabajo

- Core FTP LE v2.1. Cliente FTP gratuito.
  http://www.coreftp.com/

- Cruise Control .NET v1.4.3.4023. Herramienta de integración continua.
  http://ccnet.thoughtworks.com/

- Dotnet Framework v3.5. Framework de entorno o de desarrollo creado por Microsoft.
  http://msdn.microsoft.com/en-us/netframework/default.aspx

- FxCop v1.35. Herramienta que analiza el código de los ensamblados realizados para la plataforma .NET, permitiendo comprobar si se ha desarrollado siguiendo las guías de mejores prácticas de Microsoft.
  http://msdn.microsoft.com/en-us/library/bb429476(VS.80).aspx

- Internet Information Server v7.0. Servidor Web que permite trabajar con la plataforma .NET.
  http://www.iis.net/

- ***MCTest***. Herramienta que permite probar, comparar o medir algoritmos de comparación de modelos de manera automática y con una interfaz de usuario amigable.

- ***ModelVersionControlSystem***. Prototipo de sistema de control de versiones para modelos que permite definir cuando los cambios en un modelo del repositorio conducen a una nueva versión de dicho modelo dentro del repositorio.

- MSBuild v3.5. Plataforma de construcción utilizada por Visual Studio que también puede ser utilizada aisladamente de manera muy similar a otras herramientas como *Ant* o *NAnt*.
  http://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx

- ***MsBuild.Extends***. *Plug-in* para la herramienta MSBuild:

  1. ***AutoSvnCommit***. Tarea para MSBuild que permite introducir las fuentes en un repositorio de *Subversion*. En el prototipo se utiliza como punto de inicio para realizar la integración de los tres repositorios principales o básicos.

2. ***OpenArchitectureWare***. Tarea realizada para MSBuild que permite ejecutar una generación de artefactos utilizando las herramientas incluidas en el Eclipse Modeling Project derivadas del proyecto openArchitectureWare.
3. ***UniqueSqlScriptsExecutor***. Esta tarea permite introducir un conjunto de *scripts* SQL, ubicados en diferentes archivos, en la base de datos en un orden preestablecido para evitar conflictos e inconsistencias con los datos. Además, permite introducir únicamente los *scripts* que han sido modificados desde la última vez que se haya ejecutado la tarea, permitiendo una evolución iterativa de la base de datos.

- MsTest v9.0. Herramienta de línea de comandos que se utiliza para ejecutar pruebas en la plataforma .NET.
  http://msdn.microsoft.com/en-us/library/ms182489(VS.80).aspx
- ***MWE.Extends***. *Plug-in* para la herramienta MWE (Modeling Workflow Engine):
  1. ***MetamodelExt***. Herramienta software que posibilita, a partir de un metamodelo Ecore, crear su metamodelo extendido, en el que para cada metaclase original se añaden cuatro nuevas metaclases (añadido, borrado, cambiado, no cambiado), que heredan de la original. Así, se permite representar la diferencia de dos modelos en base al metamodelo extendido.
  2. ***Repository***. Se encarga de simular el comportamiento de un repositorio. Permite realizar pruebas de forma fluida y podría ser sustituido en cualquier momento por un sistema de control de versiones para modelos que ofrezca mayores características.
  3. ***UnifiedDiff***. Permite calcular y representar la diferencia entre dos modelos que conforman a un mismo metamodelo, generando así otro modelo que conforma a otro metamodelo derivado del primero de ellos. Además, incluye todos los elementos que permanecen intactos en la última versión del modelo para permitir generar artefactos en función de ellos si así fuera necesario. Los modelos surgidos de las diferencias serán artefactos de primera clase en el proceso de generación, permitiendo captar en un único archivo a todos los elementos que se han actualizado, borrado, añadido o que permanecen intactos.
- NCover Community v1.5.8. Herramienta para determinar la cobertura de código en desarrollos realizados para la plataforma .NET

http://www.ncover.com/

- NDepend v3.0. Herramienta que sirve para analizar la estructura del código, realizar *refactorings*, revisar código o comparar diferentes versiones del mismo. http://www.ndepend.com/
- NDoc v1.3.1. Herramienta que sirve para generar archivos de documentación a partir de los comentarios insertados en el código fuente de una aplicación. http://ndoc.sourceforge.net/
- ***OAW.xsl***. Hoja de estilo utilizada por CCNET para procesar la salida del generador de código y mostrarlo de forma entendible por personas.
- Robocopy. Herramienta muy robusta que permite copiar archivos de una ruta a otra de forma eficiente.
  http://en.wikipedia.org/wiki/Robocopy/
- Simple WebSite Software v1.0. Software cuyo objetivo principal es proporcionar lo necesario para generar aplicaciones Web con una pequeña cantidad de esfuerzo. Posee únicamente una serie de componentes básicos como *banner* superior, menús en la barra izquierda y derecha, cuerpo y pie de páginas, etc.
  http://phpsws.sourceforge.net/
- Sql Server 2008 Express. Sistema de gestión de bases de datos.
  http://www.microsoft.com/express/sql/default.aspx
- Sql Server 2008 Management Studio. Herramienta gráfica que facilita el trabajo con Sql Server.
  http://msdn.microsoft.com/en-us/library/ms365247.aspx
- Subversion v1.6.3. Sistema de control de versiones.
  http://subversion.tigris.org/
- SvnRevisionLabeller v2.0. Permite etiquetar la retroalimentación obtenida al realizar una integración a partir de un repositorio Subversion según el número de la revisión que tiene almacenado el repositorio, de modo que permanecerán sincronizadas ambas versiones.
  http://code.google.com/p/svnrevisionlabeller/
- The Beer House v2.0. *Starter kit* para crear aplicaciones CMS (Content Management System) y de comercio electrónico de forma sencilla y completa.
  http://www.codeplex.com/TheBeerHouse/

- Tortoise SVN v1.6.0. Interfaz gráfica de usuario para trabajar con Subversion en Windows.
  http://tortoisesvn.tigris.org/
- Tweeterati v1.0B3. Cliente Windows de Twitter que permite trabajar del mismo modo de cómo se haría en la Web.
  http://tweeterati.codeplex.com/
- Visual Studio 2008. Entorno de desarrollo para la plataforma .NET.
  http://www.microsoft.com/VisualStudio/
- WampServer v2.0. Herramienta que integra al servidor Web Apache, al lenguaje de *script* PHP y al sistema de gestión de bases de datos MySql.
  http://www.wampserver.com/
- Windows Vista Ultimate. Sistema operativo.
  http://www.microsoft.com/windows/windows-vista/

APÉNDICE C

Metamodelos

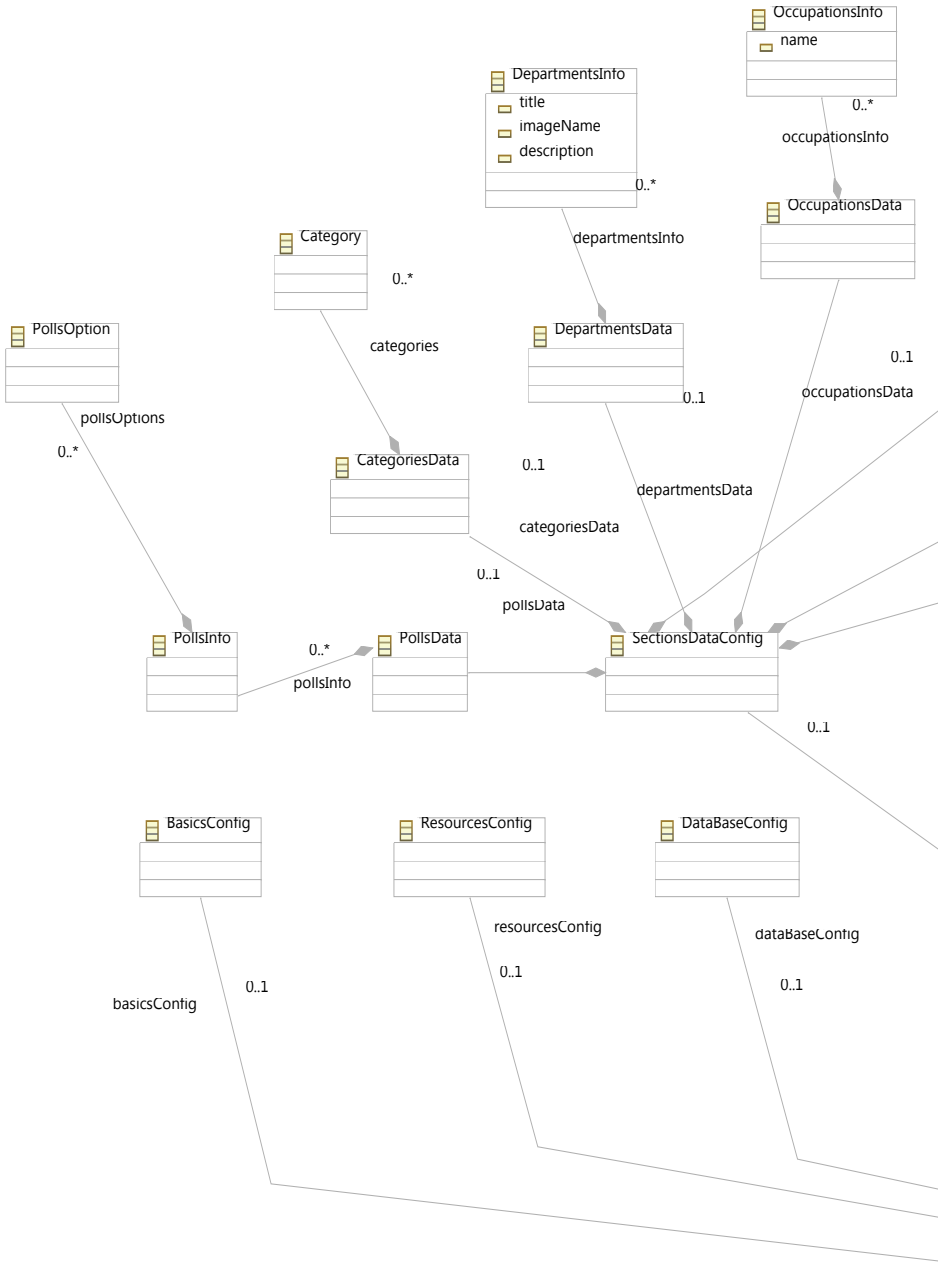


Figura C.1: Metamodelo del lenguaje CMSLanguage (1/4)

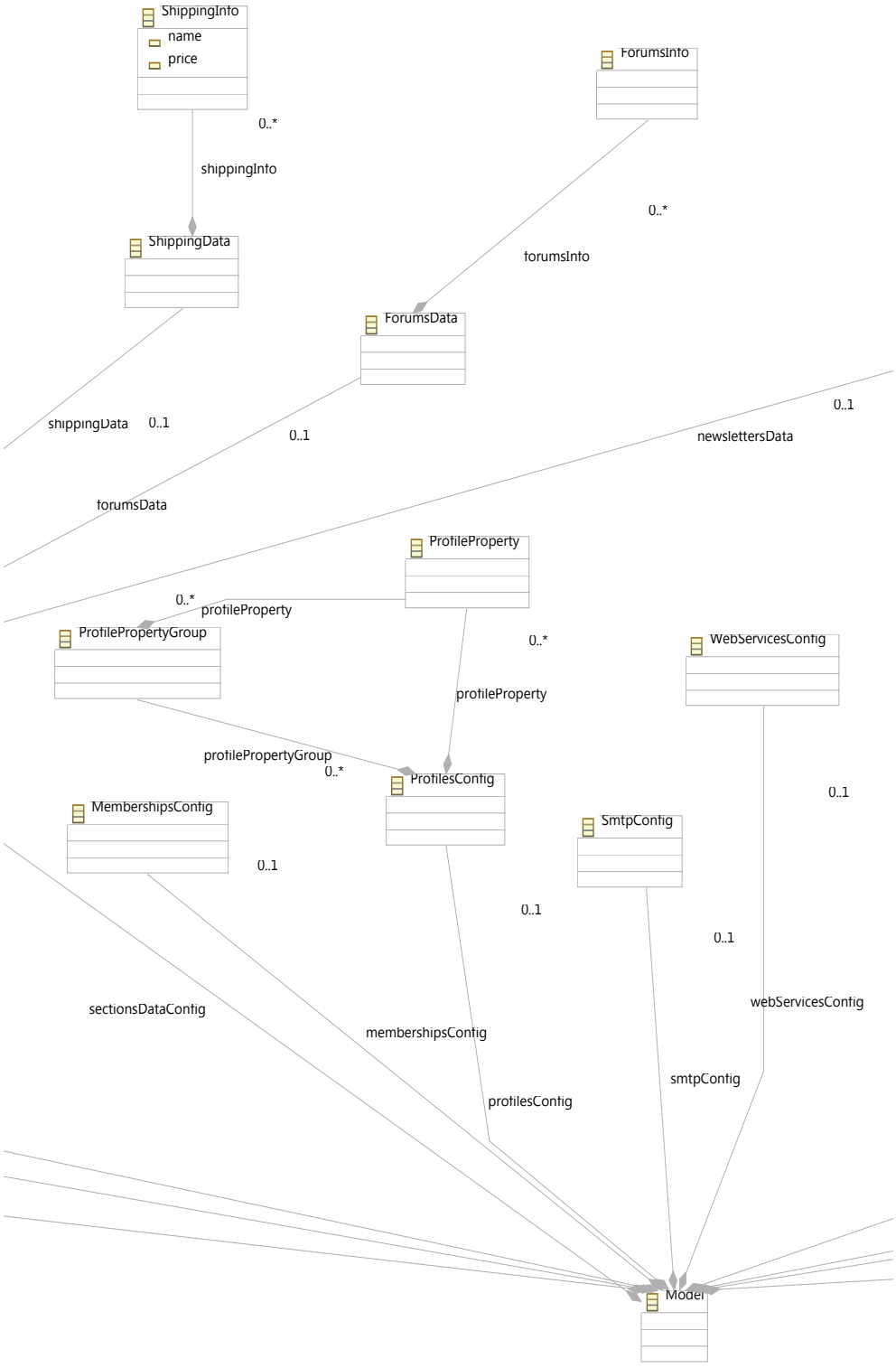


Figura C.2: Metamodelo del lenguaje CMSLanguage (2/4)

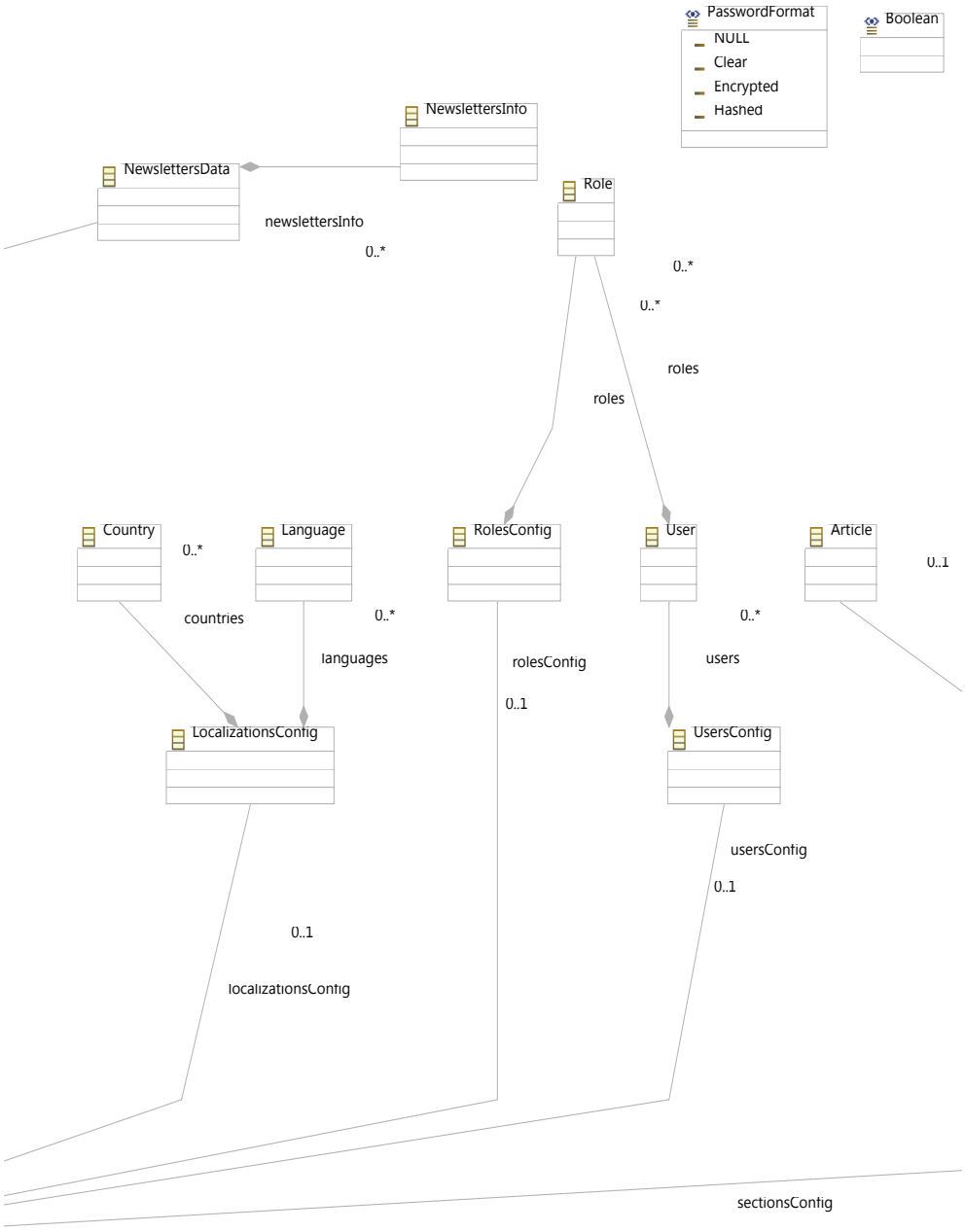


Figura C.3: Metamodelo del lenguaje CMSLanguage (3/4)

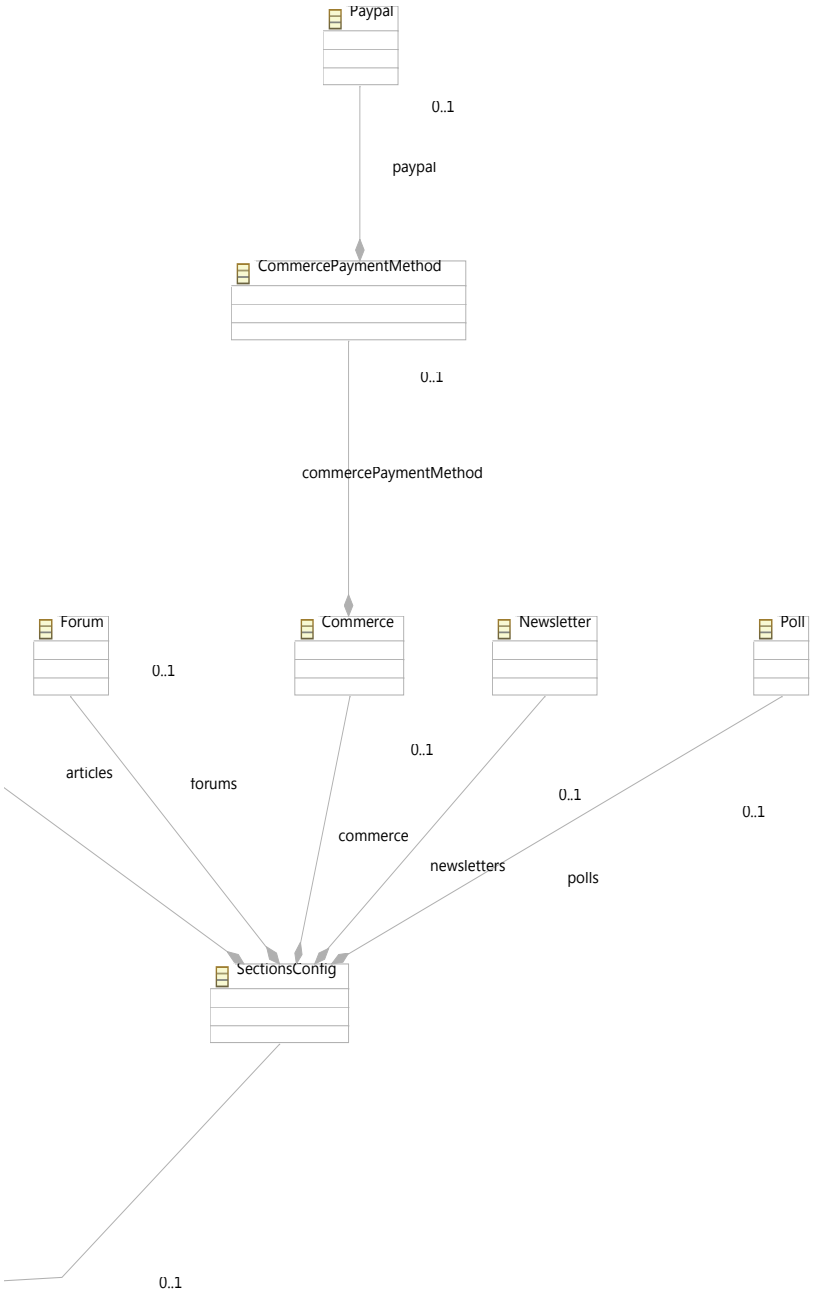


Figura C.4: Metamodelo del lenguaje CMSLanguage (4/4)

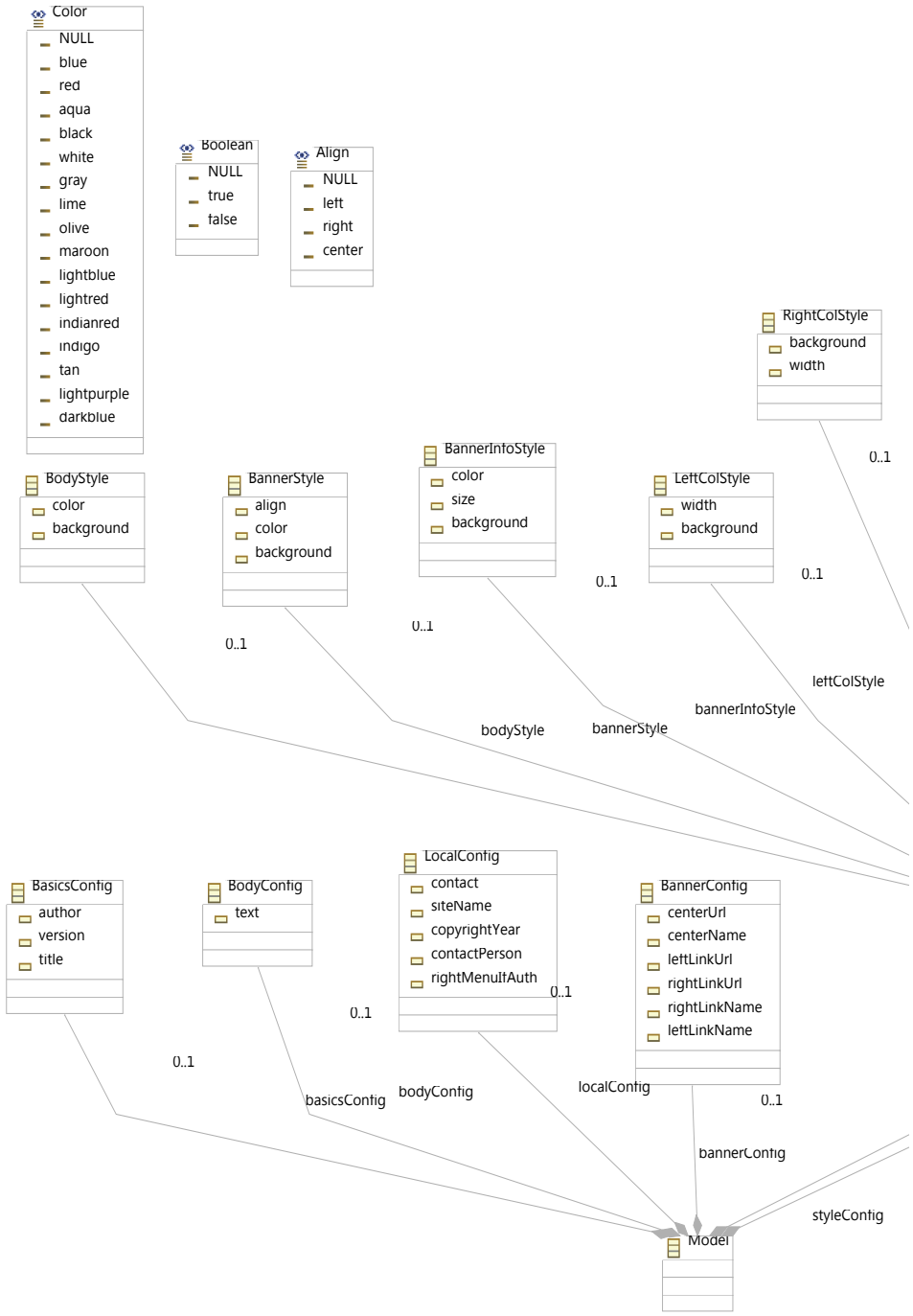


Figura C.5: Metamodelo del lenguaje SWSLanguage (1/2)

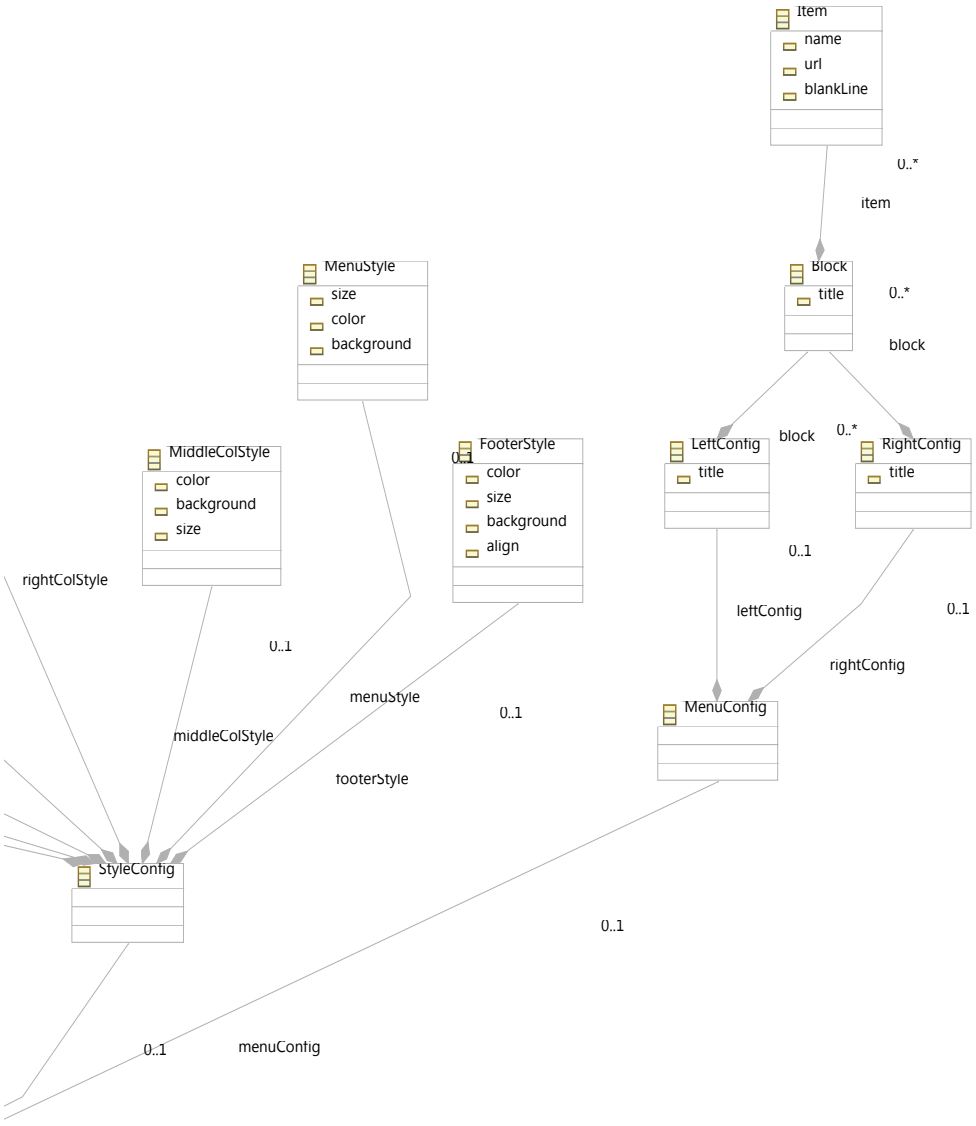


Figura C.6: Metamodelo del lenguaje SWSLanguage (2/2)

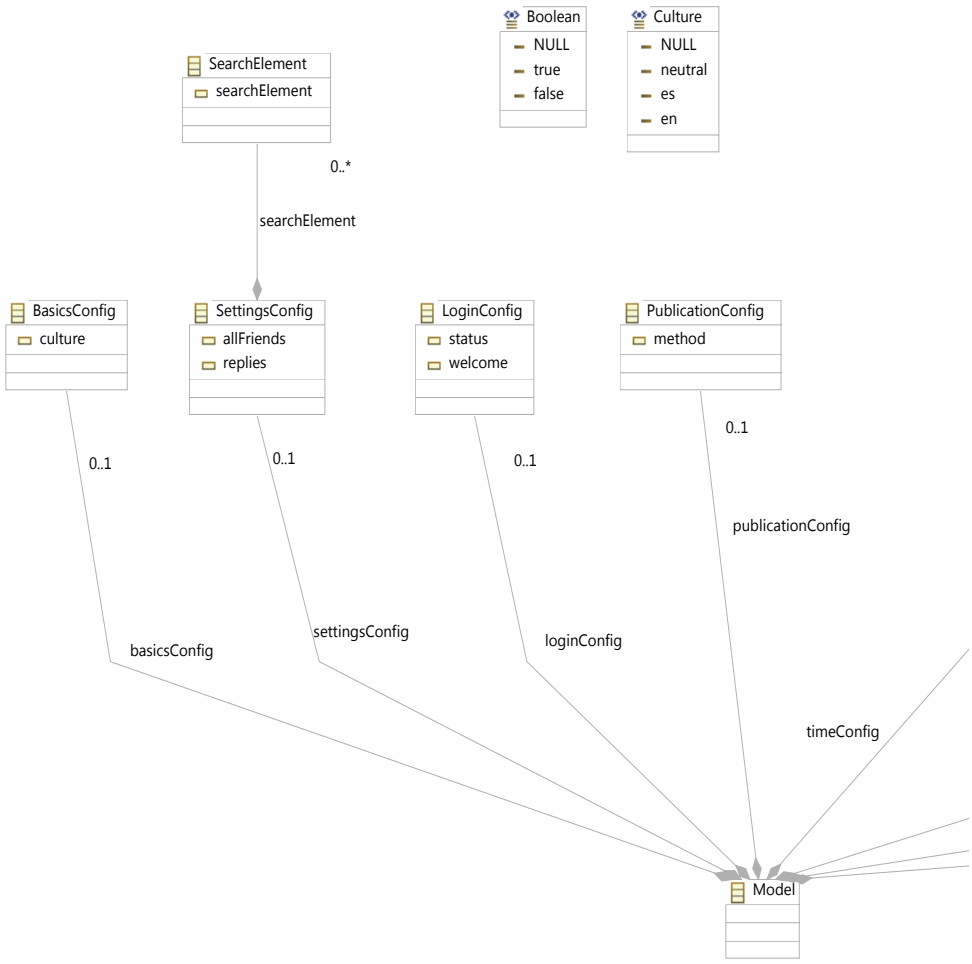


Figura C.7: Metamodelo del lenguaje TWILanguage (1/2)

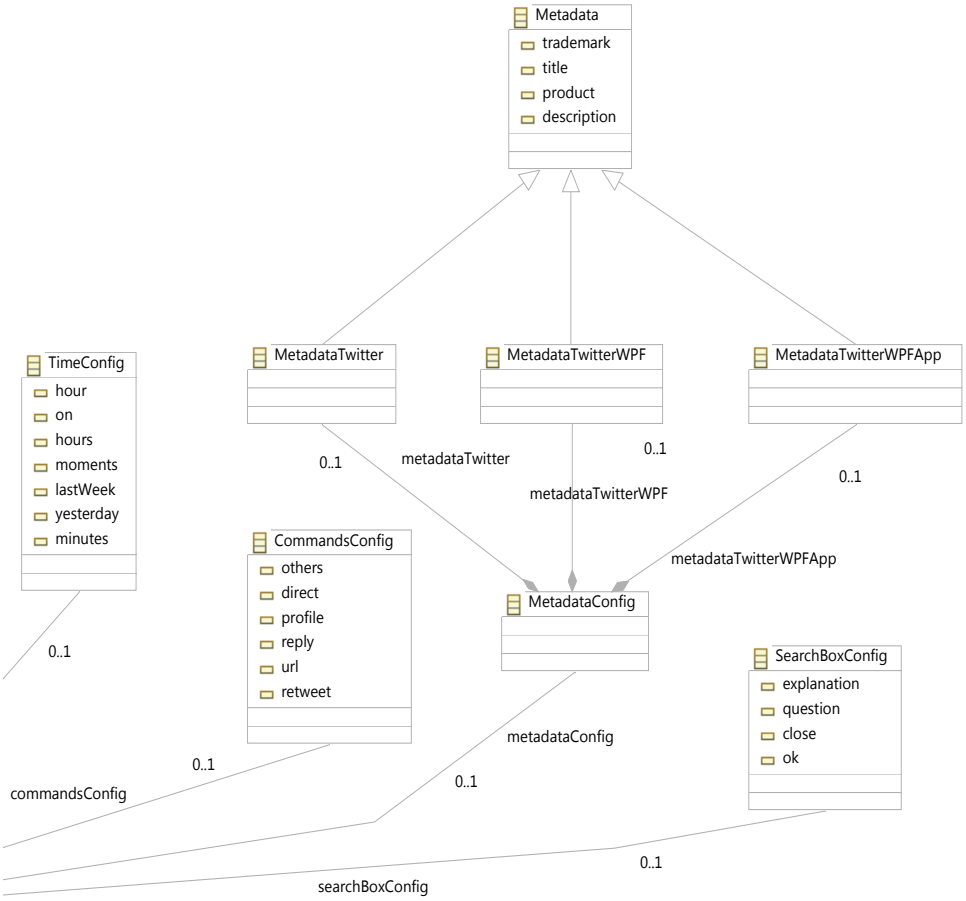


Figura C.8: Metamodelo del lenguaje TWILanguage (2/2)

Referencias

[Alanen and Porres, 2003] Alanen, M. and Porres, I. (2003). Difference and union of models. In *Proceedings of UML 2003*, pages 2–17.

[Alonso, 2008] Alonso, J. M. (2008). Towards egovernment 2.0. Technical report, W3C.

[Altmanninger, 2007] Altmanninger, K. (2007). Models in conflict - towards a semantically enhanced version control system for models. In Giese, H., editor, *MoDELS Workshops*, volume 5002 of *Lecture Notes in Computer Science*, pages 293–304. Springer.

[Altmanninger et al., 2009] Altmanninger, K., Brosch, P., Kappel, G., Langer, P., Seidl, M., Wieland, K., and Wimmer, M. (2009). Why model versioning research is needed!? an experience report. In *Proceedings of the Joint MoDSE-MCCM 2009 Workshop*. Vortrag: Joint MoDSE-MCCM 2009 Workshop - Models and Evolution, Denver, USA; 2009-10-04.

[Altmanninger et al., 2008] Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Seidl, M., Schwinger, W., and Wimmer, M. (2008). Amor - towards adaptable model versioning. In *1st Int. Workshop on Model Co-Evolution and Consistency Management, in conjunction with Models'08*.

[Arabnia et al., 2009] Arabnia, H. R., de la Fuente, D., and Olivas, J. A., editors (2009). *Proceedings of the 2009 International Conference on Artificial Intelligence, ICAI 2009, July 13-16, 2009, Las Vegas Nevada, USA, 2 Volumes*. CSREA Press.

[Arabnia and Mun, 2008] Arabnia, H. R. and Mun, Y., editors (2008). *Proceedings of the 2008 International Conference on Artificial Intelligence, ICAI 2008, July 14-17, 2008, Las Vegas, Nevada, USA, 2 Volumes (includes the 2008 International Conference on Machine Learning; Models, Technologies and Applications)*. CSREA Press.

[Atkinson and Kühne, 2003] Atkinson, C. and Kühne, T. (2003). Model-driven development: A metamodeling foundation. *IEEE Software*, 20:36–41.

[Bassett, 1996] Bassett, P. G. (1996). *Framing software reuse: lessons from the real world*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Beck, 1994] Beck, K. (1994). Simple smalltalk testing: With patterns. *The Smalltalk Report*, 4(2):16–18.

[Beck, 1999] Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

[Beck, 2003] Beck, K. (2003). *Test-Driven Development by Example*. The Addison-Wesley Signature Series. Addison-Wesley.

[Bellinaso, 2006] Bellinaso, M. (2006). *ASP.NET 2.0 Website Programming: Problem - Design - Solution*. Wrox.

[Bendix and Emanuelsson, 2008] Bendix, L. and Emanuelsson, P. (2008). Diff and merge support for model based development. In *CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models*, pages 31–34, New York, NY, USA. ACM.

[Berardi et al., 2009] Berardi, N., Katawazi, A., and Bellinaso, M. (2009). *ASP.NET MVC 1.0 Website Programming: Problem - Design - Solution*. Wrox.

[Boehm, 2002] Boehm, B. W. (2002). *Software engineering economics*. Springer-Verlag New York, Inc., New York, NY, USA.

[Booch et al., 1999] Booch, G., Rumbaugh, J., and Rumbaugh, J. (1999). *The Unified Modeling Language User Guide*. James Rumbaugh.

[Brun and Pierantonio, 2008] Brun, C. and Pierantonio, A. (2008). Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, 9(2):29–34.

[Budinsky et al., 2003] Budinsky, F., Brodsky, S. A., and Merks, E. (2003). *Eclipse Modeling Framework*. Pearson Education.

[Bunse et al., 2009] Bunse, C., Gross, H.-G., and Peper, C. (2009). Models in software engineering. In Chaudron, M. R., editor, *Embedded System Construction — Evaluation of Model-Driven and Component-Based Development Approaches*, pages 66–77. Springer-Verlag, Berlin, Heidelberg.

[Bézivin, 2005] Bézivin, J. (2005). On the unification power of models. In *Software and System Modeling*, volume 4.

[Bézivin et al., 2005a] Bézivin, J., Brunette, C., Chevrel, R., Jouault, F., and Kurtev, I. (2005a). Bridging the generic modeling environment and the eclipse modeling framework. In *OOPSLA'05: Proceedings of the International Workshop on Software Factories*.

[Bézivin et al., 2003] Bézivin, J., Dupé, G., Jouault, F., Pitette, G., and Rougui, J. E. (2003). First experiments with the atl model transformation language: Transforming xslt into xquery. In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*.

[Bézivin et al., 2005b] Bézivin, J., Hillairet, G., Jouault, F., Kurtev, I., and Piers, W. (2005b). Bridging dsl tools and the eclipse modeling framework. In *OOPSLA'05: Proceedings of the International Workshop on Software Factories*.

[Caldiera and Basili, 1991] Caldiera, G. and Basili, V. R. (1991). Identifying and qualifying reusable software components. *Computer*, 24(2):61–70.

[Carbonell et al., 1984] Carbonell, J. G., Michalski, R. S., and Mitchell, T. M. (1984). An overview of machine learning. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 3–23. Springer, Berlin, Heidelberg.

[Chandrasekaran et al., 1999] Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26.

[Chapin et al., 2001] Chapin, N., Hale, J. E., Fernandez-Ramil, J., and Tan, W.-G. (2001). Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(1):3–30.

[Chen, 2004] Chen, X. (2004). *Developing Application Frameworks in .NET*. Apress.

[Cicchetti et al., 2007a] Cicchetti, A., Ruscio, D. D., and Pierantonio, A. (2007a). Applying a difference representation approach to the petrinet metamodel. Technical report, Dipartimento di Informatica, Università degli Studi dell'Aquila.

[Cicchetti et al., 2007b] Cicchetti, A., Ruscio, D. D., and Pierantonio, A. (2007b). An atl based implementation to support a metamodel independent approach to

difference representation. Technical report, Dipartimento di Informatica, Università degli Studi dell'Aquila.

[Cicchetti et al., 2007c] Cicchetti, A., Ruscio, D. D., and Pierantonio, A. (2007c). A metamodel independent approach to difference representation. *Journal of Object Technology*, 6(9):165–185.

[Cleaveland and Cleaveland, 2001] Cleaveland, C. C. and Cleaveland, J. C. (2001). *Program Generators with XML and Java with CD-ROM.* Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Clements and Northrop, 2002] Clements, P. and Northrop, L. (2002). *Software Product Lines - Practice and Patterns.* Addison-Wesley.

[Cobena et al., 2001] Cobena, G., Abiteboul, S., and Marian, A. (2001). Detecting changes in xml documents. In *In ICDE*, pages 41–52.

[Compuware, 2005] Compuware (2005). Compuware optimalj: How model-driven development enhances productivity. Technical report, Compuware Corporation.

[Cook, 2004] Cook, S. (2004). *The MDA Journal: Model Driven Architecture Straight From The Masters*, chapter Domain-Specific Modeling and Model Driven Architecture, page ch. 5. Meghan Kiffer Pr.

[Cook et al., 2007] Cook, S., Jones, G., Kent, S., and Wills, A. C. (2007). *Domain-Specific Development with Visual Studio DSL Tools.* Addison-Wesley.

[Cook and Kent, 2008] Cook, S. and Kent, S. (2008). The domain-specific ide. *The European Journal for the Informatics Professional (UPGRADE)*, 9(2):17–21.

[Cusumano and Selby, 1997] Cusumano, M. A. and Selby, R. W. (1997). How microsoft builds software. *Communications of the ACM*, 40(6):53–61.

[Czarnecki and Eisenecker, 2000] Czarnecki, K. and Eisenecker, U. (2000). *Generative Programming.* Addison-Wesley.

[Czarnecki and Helsen, 2003] Czarnecki, K. and Helsen, S. (2003). Classification of model transformation approaches. In *OOPSLA '03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.*

[de Sousa Saraiva and da Silva, 2009] de Sousa Saraiva, J. and da Silva, A. R. (2009). Development of cms-based web-applications using a model-driven approach. *Software Engineering Advances, International Conference on*, 0:500–505.

[Dijkstra, 1972] Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, 15(10):859–866.

[Djaouti et al., 2008] Djaouti, D., Alvarez, J., Jessel, J.-P., Methel, G., and Molinier, P. (2008). A gameplay definition through videogame classification. *Int. J. Comput. Games Technol.*, 2008:1–7.

[Djurić, 2006] Djurić, D. (2006). The tao of modeling spaces. *Journal of Object Technology*, 5:125–147.

[Dollard, 2004] Dollard, K. (2004). *Code Generation in Microsoft .NET*. Apress.

[Downing, 1998] Downing, T. B. (1998). *Java RMI: Remote Method Invocation*. IDG Books Worldwide, Inc., Foster City, CA, USA.

[Duvall et al., 2007] Duvall, P., Matyas, S., and Glover, A. (2007). *Continuous integration: improving software quality and reducing risk*. Addison-Wesley Professional.

[Ebert et al., 2001] Ebert, C., Parro, C. H., Suttels, R., and Kolarczyk, H. (2001). Improving validation activities in a global software development. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 545–554, Washington, DC, USA. IEEE Computer Society.

[Efftinge and Kadura, 2006] Efftinge, S. and Kadura, C. (2006). *OpenArchitectureWare 4.1 Xpand Language Reference*. openArchitectureWare.

[Efftinge and Völter, 2006] Efftinge, S. and Völter, M. (2006). oaw xtext: A framework for textual dsls. Technical report, openArchitectureWare.

[Elrad et al., 2001] Elrad, T., Filman, R. E., and Bader, A. (2001). Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32.

[ESA, 2009] ESA (2009). Essential facts about the computer and video game industry 2009 sales, demographic and usage data. Technical report, Entertainment Software Association.

[Essalmi and Ayed, 2006] Essalmi, F. and Ayed, L. J. B. (2006). Graphical uml view from extended backus-naur form grammars. In *ICALT '06: Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies*, pages 544–546, Washington, DC, USA. IEEE Computer Society.

[Farail et al., 2006] Farail, P., Gaufillet, P., Canals, A., Le Camus, C., Sciamma, D., Michel, P., Crégut, X., and Pantel, M. (2006). the topcased project: a toolkit in open source for critical aeronautic systems design. In *Embedded Real Time Software (ERTS)*, Toulouse.

[Fellbaum, 1998] Fellbaum, C., editor (1998). *WordNet. An Electronic Lexical Database*. The MIT Press, Cambridge, MA ; London.

[Fenster, 2006] Fenster, L. (2006). *Effective Use of Microsoft Enterprise Library: Building Blocks for Creating Enterprise Applications and Services (Microsoft .NET Development Series)*. Addison-Wesley Professional.

[Fernández-Fernández et al., 2008] Fernández-Fernández, H., Palacios-González, E., García-Díaz, V., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2008). Design of intelligent business applications based in bpm and mde. In [Arabnia and Mun, 2008], pages 591–597.

[Fernández-Fernández et al., 2010] Fernández-Fernández, H., Palacios-González, E., García-Díaz, V., García-Bustelo, B. C. P., Martínez, O. S., and Lovelle, J. M. C. (2010). Sbpmn - an easier business process modeling notation for business users. *Computer Standards & Interfaces*, 32(1-2):18–28.

[Fernández-Fernández et al., 2009] Fernández-Fernández, H., Palacios-González, E., García-Díaz, V., G-Bustelo, B. C. P., Sanjuan-Martínez, O., and Lovelle, J. M. C. (2009). Developing a business application with bpm and mde. *International Journal of Artificial Intelligence and Interactive Multimedia*, 1(2):26–32.

[Figueiredo et al., 2004] Figueiredo, A. M. C. M., de Jesus Mendes, M., Kamada, A., Borelli, J. R., Rodrigues, M. A., Damasceno, L. L., Souza, J. G., and Tizzo, N. P. (2004). Applying mda concepts in an e-government platform. In Traunmüller, R., editor, *EGOV*, volume 3183 of *Lecture Notes in Computer Science*, pages 260–265. Springer.

[Fitzgerald, 2003] Fitzgerald, M. (2003). *Learning XSLT*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

[Fleischer, 2009] Fleischer, G. (2009). Continuous integration. what componies expect and solutions provide. Technical report, Fontys University of Applied Sciencies.

[Forgy, 1990] Forgy, C. L. (1990). Expert systems. chapter Rete: a fast algorithm for the many pattern/many object pattern match problem, pages 324–341. IEEE Computer Society Press, Los Alamitos, CA, USA.

[Fowler, 2004] Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Pearson Education, 3 edition.

[Fowler, 2009a] Fowler, M. (2009a). Continuous integration. Web. `http://martinfowler.com/articles/continuousIntegration.html`.

[Fowler, 2009b] Fowler, M. (2009b). Inversion of control containers and the dependency injection pattern. Web. `http://www.martinfowler.com/articles/injection.html`.

[Frame, 2002] Frame, J. D. (2002). *The new project management: tools for an age of rapid change, complexity and other business realities; electronic version*. Jossey-Bass Business & Management Series. Jossey-Bass, Newark.

[Frankel, 2003] Frankel, D. S. (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons.

[Furtado et al., 2007] Furtado, A., Santos, A., and Ramalho, G. (2007). Computer games software factory and edutainment platform for microsoft .net. *Software, IET*, 1(6):280–293.

[Gaedke et al., 2009] Gaedke, M., Grossniklaus, M., and Díaz, O., editors (2009). *Web Engineering, 9th International Conference, ICWE 2009, San Sebastián, Spain, June 24-26, 2009, Proceedings*, volume 5648 of *Lecture Notes in Computer Science*. Springer.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley.

[García-Díaz, 2009] García-Díaz, V. (2009). Aplicabilidad de la ingeniería dirigida por modelos. Technical report, Departamento de Informática, Universidad de Oviedo.

[García-Díaz et al., 2008] García-Díaz, V., Fernández-Fernández, H., Palacios-González, E., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2008). Intelligent traceability system of cabrales cheese using mda talisman. In [Arabnia and Mun, 2008], pages 578–584.

[García-Díaz et al., 2009a] García-Díaz, V., Fernández-Fernández, H., Palacios-González, E., G-Bustelo, B. C. P., Sanjuan-Martínez, O., and Lovelle, J. M. C. (2009a). Automated code generation support for bi with mda talisman. *International Journal of Artificial Intelligence and Interactive Multimedia*, 1(2):87–93.

[García-Díaz et al., 2010a] García-Díaz, V., Fernández-Fernández, H., Palacios-González, E., G-Bustelo, B. C. P., Sanjuan-Martínez, O., and Lovelle, J. M. C. (2010a). Talisman mde. mixing mde principles. *Journal of Systems and Software*, 83(7):1179–1191.

[García-Díaz et al., 2010b] García-Díaz, V., G-Bustelo, B. C. P., Sanjuán-Martínez, O., and Lovelle, J. M. C. (2010b). Towards an adaptive integration trigger. In *International Symposium on Distributed Computing and Artificial Intelligence*, pages 457–460.

[García-Díaz et al., 2011a] García-Díaz, V., G-Bustelo, B. C. P., Sanjuán-Martínez, O., and Lovelle, J. M. C. (2011a). Bridging the gap between model-driven engineering and continuous integration. *Science of Computer Programming*, **No publicado. Pendiente de aceptación**.

[García-Díaz et al., 2011b] García-Díaz, V., G-Bustelo, B. C. P., Sanjuán-Martínez, O., Valdéz, E. R. N., and Lovelle, J. M. C. (2011b). Mctest: Towards an improvement of match algorithms for models. *IET Software*, **No publicado. Pendiente de aceptación**.

[García-Díaz et al., 2009b] García-Díaz, V., Palacios-González, E., G-Bustelo, B. C. P., Palacio, D. Z., and Lovelle, J. M. C. (2009b). Version control for an intelligent traceability system focusing on underlying source code changes control. In [Arabnia et al., 2009], pages 816–822.

[García-Díaz et al., 2011c] García-Díaz, V., Sanjuán-Martínez, O., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2011c). Incremental evolution in model-driven engineering. *Journal of Software Maintenance and Evolution: Research and Practice*, **No publicado. Pendiente de aceptación**.

[García-Díaz et al., 2009c] García-Díaz, V., Tolosa, J. B., G-Bustelo, B. C. P., Palacios-González, E., Martínez, O. S., and Crespo, R. G. (2009c). Talisman mde framework: An architecture for intelligent model-driven engineering. In [Omatu et al., 2009], pages 299–306.

[García-Díaz et al., 2011] García-Díaz, V., Bustelo, B. C. P. G., Martínez, O. S., Lovelle, J. M. C., and de Pablos, P. O. (2011). *Regional Innovation Systems and Sustainable Development. Emerging Technologies*, chapter Traceability Systems for Sustainable Development in Rural Areas, pages 1–8. IGI-Global.

[Garcés et al., 2009] Garcés, K., Jouault, F., Cointe, P., and Bézivin, J. (2009). Managing model adaptation by precise detection of metamodel changes. In Paige, R. F., Hartman, A., and Rensink, A., editors, *Model Driven Architecture - Foundations and Applications, 5th European Conference, ECMDA-FA 2009, Enschede, The Netherlands, June 23-26, 2009. Proceedings*, volume 5562 of *Lecture Notes in Computer Science*, pages 34–49. Springer.

[Gasevic et al., 2006] Gasevic, D., Djuric, D., Devedzic, V., and Selic, B. (2006). *Model Driven Architecture and Ontology Development*. Springer, Secaucus, NJ, USA.

[Golan et al., 2004] Golan, E., Krissoff, B., Kuchler, F., Calvin, L., Nelson, K., and Price, G. (2004). Traceability in the u.s. food supply: Economic theory and industry studies. Agricultural Economics Reports 33939, United States Department of Agriculture, Economic Research Service.

[Grady, 1999] Grady, R. B. (1999). An economic release decision model: Insights into software project management. In *In Proceedings of the Applications of Software Measurement Conference*, pages 227–239. Orange Park, FL: Software Quality Engineering.

[Greenfield and Short, 2003] Greenfield, J. and Short, K. (2003). Software factories: Assembling applications with patterns, models, frameworks, and tools. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27, New York, NY, USA. ACM.

[Greenfield and Short, 2004] Greenfield, J. and Short, K. (2004). Moving to software factories. *Software Development Magazine*, 12(8).

[Greenfield et al., 2004] Greenfield, J., Short, K., Cook, S., and Kent, S. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons.

[Gronback, 2006] Gronback, R. C. (2006). Eclipse modeling project and omg standard. In *Eclipse Modeling Symposium*.

[Gronback, 2009] Gronback, R. C. (2009). *Eclipse Modeling Project: A Domain-specific Language Toolkit: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Educational Publishers Inc, 1 edition.

[Groth, 2004] Groth, R. (2004). Is the software industry's productivity declining? *IEEE Software*, 21(6):92–94.

[Gruschko et al., 2007] Gruschko, B., Kolovos, D. S., and Paige, R. F. (2007). Towards synchronizing models with evolving metamodels. In *In Proc. Int. Workshop on Model-Driven Software Evolution held with the ECSMR*.

[Guttman, 2004] Guttman, M. (2004). *The MDA Journal: Model Driven Architecture Straight From The Masters*, chapter Microsoft Should Note Compete With MDA, page ch. 6. Meghan Kiffer Pr.

[Guttman and Parodi, 2006] Guttman, M. and Parodi, J. (2006). *Real-Life MDA: Solving Business Problems with Model Driven Architecture*. Morgan Kaufmann.

[Haase et al., 2007] Haase, A., Völter, M., Efftinge, S., and Kolb, B. (2007). Introduction to openarchitectureware 4.1.2. In *Proceedings of TOOLS EUROPE 2007 - Objects, Models, Components, Patterns,*.

[Harel and Rumpe, 2004] Harel, D. and Rumpe, B. (2004). Meaningful modeling: What's the semantics of "semantics¿ *Computer*, 37(10):64–72.

[Hashimi and Hashimi, 2006] Hashimi, S. and Hashimi, S. I. (2006). *Deploying .NET Applications*. Apress.

[Hearnden et al., 2006] Hearnden, D., Lawley, M., and Raymond, K. (2006). Incremental Model Transformation for the Evolution of Model-Driven Systems. In Nierstrasz, O., Whittle, J., Harel, D., and Reggio, G., editors, *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, chapter 23, pages 321–335. Springer Berlin / Heidelberg, Berlin, Heidelberg.

[Henriksson and Larsson, 2003] Henriksson, A. and Larsson, H. (2003). A definition of round-trip engineering. Technical report, Department of Computer and Information Science, Linkopings University, Sweden.

[Herbsleb and Grinter, 1999] Herbsleb, J. D. and Grinter, R. E. (1999). Splitting the organization and integrating the code: Conway's law revisited. In *ICSE '99:*

Proceedings of the 21st international conference on Software engineering, pages 85–95, New York, NY, USA. ACM.

[Hirschberg, 1977] Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675.

[Hnětynka and Plášil, 2004] Hnětynka, P. and Plášil, F. (2004). Distributed versioning model for mof. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin.

[Holck and Jorgensen, 2004] Holck, J. and Jorgensen, N. (2004). Continuous integration and quality assurance: A case study of two open source projects. *Australian Journal of Information Systems*, 11(1):40–53.

[Hsu and Lockwood, 1993] Hsu, J. and Lockwood, T. (1993). Collaborative computing. *BYTE*, 18(3):113–120.

[Hubert, 2001] Hubert, R. (2001). *Convergent Architecture: Building Model-Driven J2EE Systems with UML (OMG Press)*. John Wiley & Sons, Inc., New York, NY, USA. Foreword By-Taylor, David A.

[Hugues et al., 2007] Hugues, J., Pautet, L., and Zalila, B. (2007). From mdd to full industrial process: building distributed real-time embedded systems for the high-integrity domain. In *Proceedings of the 13th Monterey conference on Composition of embedded systems*, pages 35–52, Berlin, Heidelberg. Springer-Verlag.

[Hunt and Thomas, 1999] Hunt, A. and Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional.

[Hunt and Mcllroy, 1977] Hunt, J. and Mcllroy, M. (1977). An algorithm for differential file comparison. Technical Report Computing Science Technical Report No.41, Bell Telephone Laboratories.

[IBM, 2004] IBM (2004). Ibm rational rose xde developer. Technical report, Rational Software.

[IEEE, 2000] IEEE (2000). Ieee recommended practice for architectural description of software-intensive systems. Technical report, IEEE-SA Standards Board.

[Iskander, 2007] Iskander, M. (2007). *Innovations in E-learning, Instruction Technology, Assessment and Engineering Education*. Springer Publishing Company, Incorporated, 1st edition.

[Jacobson, 2007] Jacobson, I. (2007). Enough of re processes- let's do practices. In *RE*. IEEE.

[Jacobson et al., 1999] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley, Reading, Mass.

[Javed et al., 2007] Javed, A. Z., Strooper, P. A., and Watson, G. N. (2007). Automated generation of test cases using model-driven architecture. In *AST '07: Proceedings of the Second International Workshop on Automation of Software Test*, page 3, Washington, DC, USA. IEEE Computer Society.

[Johnson and Johnson, 1975] Johnson, S. C. and Johnson, S. C. (1975). Yacc: Yet another compiler-compiler. Technical report, Yacc.

[Joreskog, 1971] Joreskog, K. G. (1971). A general method for analysis of covariance structure. *Biometrika*, 57(2):239–251.

[Joreskog and Goldberger, 1975] Joreskog, K. G. and Goldberger, A. (1975). Estimation of a model with multiple indicators and multiple causes of a single latent variable. *Journal of the American Statistical Association*, 10:631–639.

[Jouault et al., 2006] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., and Valduriez, P. (2006). Atl: a qvt-like transformation language. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720, New York, NY, USA. ACM.

[Jézéquel, 2005] Jézéquel, J.-M. (2005). Model transformation techniques. Technical report, ModelWare, INRIA.

[Kang et al., 1990] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute.

[Karakoidas et al., 2004] Karakoidas, V., Androutsellis-Theotokis, S., Spinellis, D., and Charalabidis, Y. (2004). Applying mda in enterprise application interoperability: The praxis project. In Kühn, H., editor, *Workshop on Ontology and Enterprise Modelling: Intgredients for Interoperability*, pages 76–84.

[Karlesky and Williams, 2007] Karlesky, M. and Williams, G. (2007). Mocking the embedded world: Test-driven development, continuous integration, and design patterns. In *Embedded Systems Conference Silicon Valley*.

[Kelly and Tolvanen, 2008] Kelly, S. and Tolvanen, J.-P. (2008). *Domain-Specific Modeling: Enabling full code generation.* John Wiley & Sons.

[Kent, 2002] Kent, S. (2002). Model driven engineering. In *IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods*, pages 286–298, London, UK. Springer-Verlag.

[Khuller and Raghavachari, 1996] Khuller, S. and Raghavachari, B. (1996). Graph and network algorithms. *ACM Comput. Surv.*, 28(1):43–45.

[Kiczales, 1996] Kiczales, G. (1996). Aspect-oriented programming. *Computing Surveys*, 28:154.

[Kiczales et al., 2001] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. (2001). Getting started with aspectj. *Communications of the ACM*, 44(10):59–65.

[Kleppe et al., 2003] Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Knuth, 1964] Knuth, D. E. (1964). Backus normal form vs. backus naur form. *Communications of the ACM*, 7(12):735–736.

[Kolovos et al., 2009] Kolovos, D. S., Di Ruscio, D., Pierantonio, A., and Paige, R. F. (2009). Different models for model matching: An analysis of approaches to support model differencing. In *CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, pages 1–6, Washington, DC, USA. IEEE Computer Society.

[Kolovos et al., 2006] Kolovos, D. S., Paige, R. F., Kelly, T., and Polack, F. A. C. (2006). Requirements for domain-specific languages. In *In Proceedings of the First ECOOP Workshop on Domain-Specific Program Development.*

[Konemann, 2009] Konemann, P. (2009). Model-independent differences. In *CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, pages 37–42, Washington, DC, USA. IEEE Computer Society.

[Korel et al., 1991] Korel, B., Wedde, H., Magaraj, S., Nawaz, K., and Dayana, V. (1991). Version management in distributed network environment. In *Proceedings of the 3rd international workshop on Software configuration management*, pages 161–166, New York, NY, USA. ACM.

[Kotteman and Konsynski, 1984] Kotteman, J. and Konsynski, B. (1984). Dynamic metasystems for information systems development. In *Conf. Information Systems*, pages 187–204.

[Krasner and Pope, 1988] Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program*, 1(3):26–49.

[Kung et al., 1996] Kung, D. C., Gao, J., Toyoshima, P. H., and Chen, C. (1996). On regression testing of object-oriented programs. *Journal of Sys*, 32(1):21–40.

[Kurtev et al., 2002] Kurtev, I., Bézivin, J., and Aksit, M. (2002). Technological spaces: An initial appraisal. In *Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE*.

[Langlois et al., 2007] Langlois, B., Jitia, C. E., and Jouenne, E. (2007). Dsl classification. In *DSM '07: Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling*, pages 28–38.

[Lawley and Steel, 2006] Lawley, M. and Steel, J. (2006). Practical Declarative Model Transformation with Tefkat. pages 139–150.

[Ledeczi et al., 2001] Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., and Volgyesi, P. (2001). The generic modeling environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary*, volume 17.

[Lenz and Wienands, 2006] Lenz, G. and Wienands, C. (2006). *Practical Software Factories in .NET*. Apress.

[Lin et al., 2007] Lin, Y., Gray, J., and Jouault, F. (2007). Dsmdiff: A differentiation tool for domain-specific models. *European Journal of Information Systems,*, 16:349–361.

[Lin et al., 2004] Lin, Y., Zhang, J., and Gray, J. (2004). Model comparison: A key challenge for transformation testing and version control in model driven software development. In *Control in Model Driven Software Development. OOPSLA/GPCE: Best Practices for Model-Driven Software Development*.

[Love, 2009] Love, C. (2009). *ASP.NET 3.5 Website Programming: Problem - Design - Solution*. Wrox.

[Maiden and Sutcliffe, 1992] Maiden, N. and Sutcliffe, A. (1992). Exploiting reusable specifications through analogy. *Commun. ACM*, 35(4):55–64.

[Mandelin et al., 2006] Mandelin, D., Kimelman, D., and Yellin, D. (2006). A bayesian approach to diagram matching with application to architectural models. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 222–231, New York, NY, USA. ACM.

[Martínez et al., 2011a] Martínez, J. S., García-Díaz, V., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2011a). Bpmn musim: Notación bpmn muy simplificada. In *Conferencia Ibérica de Sistemas y Tecnologías de Información*.

[Martínez et al., 2011b] Martínez, J. S., García-Díaz, V., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2011b). Isastur modeler: A tool for bpmn musim. In *Conferencia Ibérica de Sistemas y Tecnologías de Información*.

[McConnell, 2004] McConnell, S. (2004). *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA.

[Mcilroy, 1968] Mcilroy, D. (1968). Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen*, pages 88–98, Germany.

[Mellor and Balcer, 2002] Mellor, S. J. and Balcer, M. J. (2002). *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley Professional.

[Mellor et al., 2003] Mellor, S. J., Clark, A. N., and Futagami, T. (2003). Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5):14–18.

[Melnik et al., 2002] Melnik, S., Garcia-molina, H., and Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm. In *18th International Conference on Data Engineering*, pages 117–128.

[Menezes et al., 1996] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA.

[Mens et al., 2005] Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., and Jazayeri, M. (2005). Challenges in software evolution. In *Proc. 8th International Workshop on Principles of Software Evolution*, pages 13–22. IEEE.

[Mernik et al., 2005] Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344.

[Metz, 1999] Metz, C. (1999). Aaa protocols: Authentication, authorization, and accounting for the internet. *IEEE Internet Computing*, 3(6):75–79.

[Microsoft, 2008] Microsoft (2008). *Microsoft Enterprise Library 4.1*. Microsoft patterns & practices.

[Microtool, 2009] Microtool (2009). objectif, the tool for model-driven software development with the uml for .net, java and c++. Technical report, Microtool.

[Miller et al., 2003] Miller, J., Mukerji, J., Belaunde, M., Burt, C., Cummins, F., Dsouza, D., Duddy, K., Kaim, W. E., Oya, M., Soley, R., and Watson, A. (2003). Mda guide, v1.0.1. Technical report, Object Management Group. `http://www.omg.org/docs/omg/03-06-01.pdf`.

[Mittal et al., 2004] Mittal, P. A., Kumar, M., Mohania, M. K., Nair, M., Batra, N., Roy, P., Saronwala, A., and Yagnik, L. (2004). A framework for egovernance solutions. *IBM J. Res. Dev.*, 48(5/6):717–733.

[Montenegro-Marín et al., 2011] Montenegro-Marín, C. E., Cueva-Lovelle, J. M., Sanjuán-Martínez, O., and García-Díaz, V. (2011). Metamodel, domain specific language (dsl) tool, and transformations for learning management systems (lms). *Journal of Systems and Software*, **No publicado. Pendiente de aceptación**.

[Montero-Reyno and Cubel, 2009] Montero-Reyno, E. and Cubel, J. . C. (2009). A platform-independent model for videogame gameplay specification. In Barry, A., Helen, K., and Tanya, K., editors, *Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference*, London. Brunel University.

[Montgomery et al., 2007] Montgomery, D. C., Peck, E. A., and Vining, G. G. (2007). *Introduction to Linear Regression Analysis, Solutions Manual (Wiley Series in Probability and Statistics)*. Wiley-Interscience.

[Mora et al., 2007] Mora, B., Ruiz, F., García, F., and Piattini, M. (2007). Experiencia en transformación de modelos de procesos de negocios desde bpmn a xpdl. In *IDEAS'07: Workshop Iberoamericano de Requisitos y Ambiestos de Software*.

[Muller, 2008] Muller, G. (2008). What is a process? Technical report, Embedded Systems Institute.

[Murta et al., 2008] Murta, L., Correa, C., Prudencio, J. G., and Werner, C. (2008). Towards odyssey-vcs 2: improvements over a uml-based version control system. In *CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models*, pages 25–30, New York, NY, USA. ACM.

[Myers, 1986] Myers, E. W. (1986). An o(nd) difference algorithm and its variations. *Algorithmica*, 1:251–266.

[Nathan, 2006] Nathan, A. (2006). *Windows Presentation Foundation Unleashed (WPF) (Unleashed)*. Sams, Indianapolis, IN, USA.

[Nejati et al., 2007] Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., and Zave, P. (2007). Matching and merging of statecharts specifications. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 54–64, Washington, DC, USA. IEEE Computer Society.

[Néron et al., 2009] Néron, B., Ménager, H., Maufrais, C., Joly, N., Maupetit, J., Letort, S., Carrère, S., Tufféry, P., and Letondal, C. (2009). Mobyle: a new full web bioinformatics framework. *Bioinformatics*, 25(22):3005–3011.

[Newton, 2007] Newton, K. (2007). *The Definitive Guide to the Microsoft Enterprise Library*. Apress, Berkely, CA, USA.

[Nguer and Spyratos, 2008] Nguer, E. M. and Spyratos, N. (2008). A user-friendly interface for evaluating preference queries over tabular data. In *SIGDOC '08: Proceedings of the 26th annual ACM international conference on Design of communication*, pages 33–42, New York, NY, USA. ACM.

[Nodenot et al., 2008] Nodenot, T., Caron, P.-A., Le Pallec, X., and Laforcade, P. (2008). Applying model driven engineering techniques and tools to the planets game learning scenario. *Journal of Interactive Media in Education*.

[Ohst et al., 2003] Ohst, D., Welle, M., and Kelter, U. (2003). Differences between versions of uml diagrams. In *ESEC/FSE*, volume 28, pages 227–236, New York, NY, USA. ACM.

[Oliveira et al., 2005] Oliveira, H. L. R., Murta, L. G. P., and Werner, C. (2005). Odyssey-vcs: a flexible version control system for uml model elements. In *SCM*, pages 1–16. ACM.

[Oliver, 2003] Oliver, I. (2003). Model driven embedded systems. In *ACSD '03: Proceedings of the Third International Conference on Application of Concurrency to System Design*, page 5, Washington, DC, USA. IEEE Computer Society.

[Olsson, 1999] Olsson, K. (1999). Daily build. the best of both worlds: Rapid development and control. Technical report, Swedish Eng. Industries.

[Omatu et al., 2009] Omatu, S., Rocha, M., Bravo, J., Riverola, F. F., Corchado, E., Bustillo, A., and Corchado, J. M., editors (2009). *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, 10th International Work-Conference on Artificial Neural Networks, IWANN 2009 Workshops, Salamanca, Spain, June 10-12, 2009. Proceedings, Part II*, volume 5518 of *Lecture Notes in Computer Science*. Springer.

[OMG, 1991] OMG (1991). Interface definition language. Technical report, OMG. `http://www.iso.org/cate/d25486.html`.

[OMG, 2003a] OMG (2003a). Uml 2.0 diagram interchange. Technical report, OMG. `http://www.omg.org/cgi-bin/doc?ptc/2003-09-01`.

[OMG, 2003b] OMG (2003b). Uml 2.0 ocl specification. Technical report, OMG. `www.omg.org/docs/ptc/03-10-14.pdf`.

[OMG, 2005a] OMG (2005a). Meta object facility 2.0. Technical report, OMG. `http://www.omg.org/mof/`.

[OMG, 2005b] OMG (2005b). Meta object facility 2.0 xmi mapping specification, v2.1. Technical report, Object Management Group. `http://www.omg.org/docs/formal/05-09-01.pdf`.

[OMG, 2005c] OMG (2005c). Meta object facitity 2.0 query view transformation. Technical report, OMG. `http://www.omg.org/docs/ptc/07-07-07.pdf`.

[OMG, 2005d] OMG (2005d). Ontology definition metamodel. Technical report, Object Management Group. `http://www.omg.org/docs/ad/05-08-01.pdf`.

[OMG, 2007a] OMG (2007a). Meta object facility versioning and development lifecycle specification 2.0. Technical report, Object Management Group. `http://www.omg.org/docs/formal/07-05-01.pdf`.

[OMG, 2007b] OMG (2007b). Uml 2.0 infrastructure specification. Technical report, Object Management Group. `http://www.omg.org/docs/formal/07-11-04.pdf`.

[OMG, 2007c] OMG (2007c). Uml 2.0 superstructure specification. Technical report, OMG. `www.omg.org/docs/formal/07-11-02.pdf`.

[OReilly et al., 2003] OReilly, C., Morrow, P., and Bustard, D. (2003). *Software Configuration Management*, volume 2649/2003 of *Lecture Notes in Computer Science*, chapter Improving Conflict Detection in Optimistic Concurrency Control Models, pages 61–69. Springer.

[Palacios-González et al., 2008a] Palacios-González, E., Fernández-Fernández, H., García-Díaz, V., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2008a). A review of intelligent software development tools. In [Arabnia and Mun, 2008], pages 585–590.

[Palacios-González et al., 2008b] Palacios-González, E., Fernández-Fernández, H., García-Díaz, V., G-Bustelo, B. C. P., Lovelle, J. M. C., and Martínez, O. S. (2008b). General purpose mde tools. *International Journal of Artificial Intelligence and Interactive Multimedia*, 1(1):72–75.

[Palacios-González et al., 2009] Palacios-González, E., García-Bustelo, B. C. P., García-Díaz, V., Fernández-Fernández, H., and Lovelle, J. M. C. (2009). Intelligent generation of web applications for egovernment. In [Arabnia et al., 2009], pages 811–815.

[Parnas, 1976] Parnas, D. L. (1976). On the design and development of program families. *IEEE Trans. Softw. Eng.*, 2(1):1–9.

[Pavlo et al., 2006] Pavlo, A., Couvares, P., Gietzel, R., Karp, A., Alderman, I. D., Livny, M., and Bacon, C. (2006). The nmi build & test laboratory: continuous integration framework for distributed computing software. In *LISA '06: Proceedings of the 20th conference on Large Installation System Administration*, pages 21–21, Berkeley, CA, USA. USENIX Association.

[Peterson, 1981] Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Pham et al., 2007] Pham, H. N., Mahmoud, Q. H., Ferworn, A., and Sadeghian, A. (2007). Applying model-driven development to pervasive system engineering. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 7, Washington, DC, USA. IEEE Computer Society.

[PMI, 2005] PMI (2005). *A Guide to the Project Management Body of Knowledge*. Project Management Institute, third edition.

[Pohl et al., 2005] Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software Product Line Engineering : Foundations, Principles and Techniques*. Springer.

[Pontico et al., 2007] Pontico, F., Farenc, C., and Winckler, M. (2007). Model-based support for specifying eservice egovernment applications. In *TAMODIA'06: Proceedings of the 5th international conference on Task models and diagrams for users interface design*, pages 54–67, Berlin, Heidelberg. Springer-Verlag.

[Pope, 1998] Pope, A. L. (1998). *The CORBA reference guide: understanding the Common Object Request Broker Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Pressman, 1986] Pressman, R. S. (1986). *Software engineering: a practitioner's approach (2nd ed.)*. McGraw-Hill, Inc., New York, NY, USA.

[quan QIAO et al., 2007] quan QIAO, X., feng LI, X., and LI, Y. (2007). Mda-based 3g service creation approach and telecom service domain meta-model. *The Journal of China Universities of Posts and Telecommunications*.

[Ráth et al., 2008] Ráth, I., Bergmann, G., Ökrös, A., and Varró, D. (2008). Live model transformations driven by incremental pattern matching. In *Proceedings of the 1st international conference on Theory and Practice of Model Transformations*, ICMT '08, pages 107–121, Berlin, Heidelberg. Springer-Verlag.

[Ratzan, 1998] Ratzan, S. (1998). *The mad cow crisis : health and the public good*. NYU Press.

[Read and Cornell, 1977] Read, R. C. and Cornell, D. G. (1977). The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363.

[Reddy and France, 2005] Reddy, R. and France, R. (2005). Model composition - a signature-based approach. In *in "Aspect Oriented Modeling (AOM) Workshop, Montego*.

[Redmond, 1997] Redmond, F. E. (1997). *Dcom: Microsoft Distributed Component Object Model with Cdrom*. IDG Books Worldwide, Inc., Foster City, CA, USA.

[Reiter et al., 2007] Reiter, T., Altmanninger, K., Bergmayr, A., and Kotsis, G. (2007). Models in conflict - detection of semantic conflicts in model-based development. In *Proceedings of 3rd International Workshop on Model-Driven Enterprise Information Systems (MDEIS-2007), in conjuction with 9th International Conference on Enterprise Information Systems*.

[Richard E and Clark, 2007] Richard E, M. and Clark, R. C. (2007). *e-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning*. Pfeiffer.

[Richardson and Gwaltney, 2005] Richardson, J. and Gwaltney, W. (2005). *Ship it! A Practical Guide to Successful Software Projects*. Pragmatic Bookshelf.

[Rivera and Vallecillo, 2008] Rivera, J. E. and Vallecillo, A. (2008). Representing and operating with model differences. In *TOOLS EUROPE 2008*, volume 11 of *LNBIP*, pages 141–160. Springer.

[Robinson, 1965] Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *J. ACM*, 12:23–41.

[Schmidt and Gloetzner, 2008] Schmidt, M. and Gloetzner, T. (2008). Constructing difference tools for models using the sidiff framework. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, pages 947–948, New York, NY, USA. ACM.

[Scribner et al., 2000] Scribner, K., Scribner, K., and Stiver, M. C. (2000). *Understanding Soap: Simple Object Access Protocol*. Sams, Indianapolis, IN, USA.

[Seidewitz, 2003] Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5):26–32.

[Selic, 2003] Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25.

[Selic, 2008] Selic, B. (2008). Mda manifestations. *The European Journal for the Informatics Professional (UPGRADE)*, 9(2):12–16.

[Selonen, 2007] Selonen, P. (2007). A review of uml model comparison approaches. In *Proceedings of Nordic Workshop on Model Driven Engineering*.

[Selonen and Kettunen, 2007] Selonen, P. and Kettunen, M. (2007). Metamodel-based inference of inter-model correspondence. In *CSMR '07: Proceedings of the*

11th European Conference on Software Maintenance and Reengineering, pages 71–80, Washington, DC, USA. IEEE Computer Society.

[Sharpe, 2003] Sharpe, R. (2003). Building a better bug-trap. *The Economist. Science Technology Quarterly*.

[Shneiderman, 1987] Shneiderman, B. (1987). Designing the user interface strategies for effective human-computer interaction. *SIGBIO News.*, 9(1):6.

[Shukla and Schmidt, 2006] Shukla, D. and Schmidt, B. (2006). *Essential Windows Workflow Foundation (Microsoft .Net Development Series)*. Addison-Wesley Professional.

[Smith, 1982] Smith, A. J. (1982). Cache memories. *ACM Comput. Surv.*, 14(3):473–530.

[Souer et al., 2009] Souer, J., Kupers, T., Helms, R., and Brinkkemper, S. (2009). Model-driven web engineering for the automated configuration of web content management systems. In Gaedke, M., Grossniklaus, M., and Díaz, O., editors, *ICWE*, volume 5648 of *Lecture Notes in Computer Science*, pages 121–135. Springer.

[Spinellis, 2002] Spinellis, D. D. (2002). The information furnace: User-friendly home control. In *In Proceedings of the 3rd International System Administration and Networking Conference SANE 2002*, pages 145–174.

[StandishGroup, 2008] StandishGroup (2008). www.standishgroup.com.

[Steinberg et al., 2009] Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional.

[Sztipanovits and Karsai, 1997] Sztipanovits, J. and Karsai, G. (1997). Model-integrated computing. *Computer*, 30(4):110–111.

[Tai, 1979] Tai, K.-C. (1979). The tree-to-tree correction problem. *J. ACM*, 26(3):422–433.

[Tairas et al., 2008] Tairas, R., Mernik, M., and Gray, J. (2008). Using ontologies in the domain analysis of domain-specific languages. In *TWOMDE '08: Proceedings of the 1st International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering*.

[Tariq and Akhter, 2005] Tariq, N. and Akhter, N. (2005). Comparison of model driven architecture (mda) based tools. Technical report, Institute of Technology y Karolinska University Hospital. Stockholm.

[Tassey, 2002] Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology.

[Thomas, 2004] Thomas, D. (2004). Mda: Revenge of the modelers or uml utopia? *IEEE Software*, 21(3):15–17.

[Tichy, 1985] Tichy, W. F. (1985). Rcs - a system for version control. *Softw., Pract. Exper.*, 15(7):637–654.

[Tolosa, 2009] Tolosa, J. B. (2009). A new approach for meta-model interoperability through transformation models. Master's thesis, Master's Thesis. University of Oviedo.

[Tolosa et al., 2009] Tolosa, J. B., García-Díaz, V., Martínez, O. S., Fernández-Fernández, H., and Fernández, G. G. (2009). Towards meta-model interoperability of models through intelligent transformations. In [Omatu et al., 2009], pages 315–322.

[Tolosa et al., 2010] Tolosa, J. B., Sanjuán-Martínez, O., García-Díaz, V., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2010). Towards the systematic measurement of atl transformation models. *Software: Practice and Experience*, 41:789–815.

[Tolvanen, 2008] Tolvanen, J.-P. (2008). Domain-specific modeling in practice. Technical report, Metacase.

[Toulmé, 2007] Toulmé, A. (2007). Presentation of emf compare utility. In *EclipseCon*.

[Treude et al., 2007] Treude, C., Berlik, S., Wenzel, S., and Kelter, U. (2007). Difference computation of large models. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 295–304, New York, NY, USA. ACM.

[Vahid and Givargis, 2001] Vahid, F. and Givargis, T. (2001). *Embedded System Design: A Unified Hardware/Software Introduction.* John Wiley & Sons, Inc., New York, NY, USA.

[van der Linden et al., 2007] van der Linden, F. J., Schmid, K., and Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer.

[van Deursen, 1997] van Deursen, A. (1997). Domain-specific languages versus object-oriented frameworks: A financial engineering case study. In *Proceedings of Smalltalk and Java in Industry and Academia (STJA'97)*, pages 35–39.

[van Deursen et al., 2000] van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36.

[Varró and Balogh, 2007] Varró, D. and Balogh, A. (2007). The model transformation language of the viatra2 framework. *Sci. Comput. Program.*, 68:187–207.

[Vernadat et al., 2006] Vernadat, F., Percebois, C., Farail, P., Vingerhoeds, R., Rossignol, A., Talpin, J. P., and Chemouil, D. (2006). The topcased project - a toolkit in open-source for critical applications and system development. In *Data Systems In Aerospace (DASIA), Berlin, Germany, 22/05/2006-25/05/2006*. European Space Agency (ESA Publications).

[Völter, 2003a] Völter, M. (2003a). A catalog of patterns for program generation. In *EuroPloP2003, Eighth European Conference on Pattern Languages of Programs*.

[Völter, 2003b] Völter, M. (2003b). A generative component infrastructure for embedded systems. Technical report, Independent Consultant.

[Völter and Stahl, 2006] Völter, M. and Stahl, T. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

[Wade, 1984] Wade, J. (1984). Practical guidelines for a user-friendly interface. In *APL '84: Proceedings of the international conference on APL*, pages 365–371, New York, NY, USA. ACM.

[Wang, 2005] Wang, W. (2005). Evaluation of uml model transformation tools. Technical report, Business Informatics Group, Vienna University of Technology.

[Ward, 1994] Ward, M. (1994). Language oriented programming. *Software-Concepts and Tools*, 15:147–161.

[Watson, 2008] Watson, A. (2008). A brief history of mda. *The European Journal for the Informatics Professional (UPGRADE)*, 9(2):7–11.

[Wright, 1932] Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the 6th International Congress of Genetics. Ithaca, New York*, pages 356–366.

[Xing and Stroulia, 2005] Xing, Z. and Stroulia, E. (2005). Umldiff: an algorithm for object-oriented design differencing. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 54–65, New York, NY, USA. ACM.

[Zamora and Lemus, 2008] Zamora, C. S. and Lemus, I. S. (2008). Modelos de ecuaciones estructurales ¿qué es eso? / structural equations models: what's that? *Ciencia y Trabajo*, 10(29):106–110.

[Zarras, 2006] Zarras, A. (2006). Applying model-driven architecture to achieve distribution transparencies. *Information and Software Technology*, 48(7):498–516.

[Zhang and Shasha, 1989] Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262.

Acrónimos

AC-MDSD *Architecture-Centric Model-Driven Software Development*

AOP *Aspect Oriented Programming*

API *Application Programming Interface*

ARE *Automatic Round-trip Engineering*

ATL *ATLAS Transformation Language*

BDS *Borland Developer Studio*

BNF *Backus-Naur formalism*

BPEL *Business Process Execution Language*

BPM *Business Process Modeling*

BPMN *Business Process Modeling Notation*

CASE *Computer-Aided Software Engineering*

CC *Cruise Control*

CI *Continuous Integration*

CIM *Computation-Independent Model*

CLR *Common-Language Runtime*

CMMI *Capability Maturity Model Integration*

CMOF *Complete Meta-Object Facility*

CMS *Content Management System*

CORBA *Common Object Request Broker Architecture*

CRUD *Create, read, update and delete*

CSS *Cascading Style Sheets*

DAO *Data Access Objects*

DCOM *Distributed Component Object Model*

DSL *Domain-Specific Language*

EBNF *Extended Backus-Naur Form*

EIF *Enterprise Instrumentation Framework*

EMF *Eclipse Modeling Framework*

EMOF *Essential Meta-Object Facility*

EMP *Eclipse Modeling Project*

EPL *Eclipse Public Licence*

ER *Entity Relationship*

GEF *Graphical Editing Framework*

GMF *Graphical Modeling Framework*

GP *Generative Programming*

GPL *General Purpose Language*

GXPDML *Graphical eXtensible Process Definition Markup Language*

HTML *HyperText Markup Language*

IDE *Integrated Development Environment*

IDL *Interface Definition Language*

IIS *Internet Information Server*

INI *Windows Initialization file*

ISM *Implementation-Specific Model*

IVA *Impuesto sobre el Valor Añadido*

KM3 *Kernel Meta Meta Model*

M2M *Model-to-Model*

M2T *Model-to-Text*

MCSF *Mobile Client Software Factory*

MCTest *Model Comparison Testing Engine*

MDA *Model-Driven Architecture*

MDCI *Model-Driven Continuous Integration*

MDCIP *Model-Driven Continuous Integration Prototype*

MDD *Model-Driven Development*

MDE *Model-Driven Engineering*

MDSD *Model-Driven Software Development*

MIC *Model-Integration Computing*

MM *Model Management*

MOF *Meta-Object Facility*

MPS *Meta Programming Systen*

MS *Modeling Space*

MVCS *Model Version Control System*

MWE *Modeling Workflow Engine*

NIST *National Institute of Standards and Technology*

oAW *openArchitectureWare*

OCL *Object Constraint Language*

ODM *Ontology Definition Metamodel*

OMA *Object Management Architecture*

OMG *Object Management Group*

OWL *Web Ontology Language*

PHP *Hypertext Preprocessor*

PIM *Platform-Independent Model*

PMI *Project Management Institute*

PSM *Platform-Specific Model*

QVT *Query-View-Transformation*

RAE *Real Academia Española*

RE *Roundtrip Engineering*

RMI *Remote Method Invocation*

SAN *Storage Area Networks*

SBPMN *Simple Business Process Modeling Notation*

SCSF *Smart Client Software Factory*

SLD *Selection, Linear, Definitive*

SOAP *Simple Object Access Protocol*

SQL *Structured Query Language*

SVG *Scalable Vector Graphics*

SWS *Simple WebSite Software*

TDD *Test-Driven Development*

TMDE *TALISMAN Model-Driven Engineering*

TMF *Textual Modeling Framework*

TS *Technical Space*

UC *Unit of Comparisong*

UML *Unified Modeling Language*

UP *Unified Process*

URL *Uniform Resource Locator*

UUID *Universally Unique Identifier*

UV *Unit of Versioning*

VCS *Version Control System*

W3C *World Wide Web Consortium*

WCSF *Web Client Software Factory*

WPF *Windows Presentation Foundation*

WSSF *Web Service Software Factory*

WWF *Windows Workflow Foundation*

XMI *XML Metadata Interchange*

XML *Extensible Markup Language*

XP *Extreme Programming*

XPDML *eXtensible Process Definition Markup Language*

XSLT *Extensible Stylesheet Language Transformations*

xUML *Executable Unified Modeling Language*

Índice alfabético