novática

Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Plaza de España 6, 2ª planta, 28008 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn. 963740173 <novatica_prod@ati.es>
Administración y Redacción ATI Cataluña
Calle Àvila 48-50, 3a planta, local 9, 08005 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia <secregal@ati.es>
Suscripción y Ventas <novatica.subscripciones@atinet.es>
Publicidad Plaza de España 6, 2ª planta, 28008 Madrid
Tlnt.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: "Caleidoscopio temporal" - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

sumario

Nº 228, abril-junio 2014, año XL

Tema del próximo número: Gobierno corporativo de las TI

Presentación. Una introducción a MDE y su creciente adopción industrial

Jordi Cabot[1], Jesús García Molina[2, 4], Gustavo Rossi[3, 4]

[1]AtlanMod, Ecole des Mines de Nantes – INRIA – Nantes (Francia); [2]Facultad de Informática, Universidad de Murcia; [3]LIFIA, Universidad de La Plata (Argentina); [4]Coordinador de la sección técnica "Modelado del Software" de Novática

<jordi.cabot@inria.fr>,
<jmolina@um.es>,
<gustavo@lifia.info.unlp.edu.ar>

Editores invitados

Jordi Cabot es profesor titular en la *École des Mines* de Nantes (Francia) donde lidera el equipo AtlanMod afiliado a INRIA. Anteriormente, ha trabajado como postdoctorando en la Universidad de Toronto (Canadá), como profesor en la UOC (Universitat Oberta de Catalunya) y como investigador visitante en el *Politecnico di Milano* (Italia). Su área de investigación es la ingeniería del software, especialmente en relación al modelado de software, su verificación formal y los aspectos colaborativos y sociales que influyen en todas las actividad alrededor de ella. Aparte de sus publicaciones científicas, escribe acerca de todos estos temas en su blog Modeling Languages, <http://modeling-languages.com>.

Jesús J. García Molina es profesor de la Facultad de Informática de la Universidad de Murcia en la que ha sido decano. Ha sido pionero en nuestro país en la programación orientada a objetos y en la ingeniería de software dirigida por modelos, tanto en la enseñanza universitaria como en investigación. Dirige el grupo de investigación Modelum cuyas líneas de investigación abordan diversos aspectos de la ingeniería dirigida por modelos. Ha participado en varios proyectos de transferencia de MDE a las empresas. Desde el año 2002 colabora con ***Novática*** en la sección técnica de "*Tecnología de Objetos*" (ahora "*Modelado del Software*").

Gustavo Rossi es profesor titular de la Facultad de Informática en la Universidad Nacional de La Plata (Argentina) donde lidera el laboratorio LIFIA. Es investigador principal del CONICET (Consejo Nacional de Investigaciones Científicas y Tecnológicas) de Argentina. Su área principal de interés es la Ingeniería Web y también investiga aspectos de Ingeniería de Requisitos e Interacción Hombre-Maquina. Desde el año 2000 colabora con ***Novática*** en la Sección Técnica de "*Tecnología de Objetos*" (ahora "*Modelado del Software*").

1. Introducción

La Ingeniería del Software Dirigida por Modelos (*Model-Driven Software Engineering*, MDSE o simplemente MDE) es un área de la Ingeniería del Software que ha despertado un interés creciente a lo largo de los años transcurridos del siglo XXI.

Una prueba de ello es que la presente monografía es la tercera que le ha dedicado ***Novática*** en todo este tiempo. En marzo de 2004 se publicó la monografía "UML e Ingeniería de modelos" [1] centrada principalmente en el lenguaje de modelado UML pero que ya incluía algunos artículos, en particular "*En búsqueda de un principio básico para la Ingeniería Dirigida por Modelos*" de ***Jean Bézivin***, que presentaban el nuevo paradigma de desarrollo de software que entonces comenzaba a emerger.

Posteriormente, en marzo de 2008, la monografía *"Desarrollo de software dirigido por modelos"* [2] reunió una colección de artículos sobre los principales aspectos de MDE, escritos por algunos de los más destacados expertos del área.

Seis años después presentamos una nueva monografía sobre MDE cuyo título *"Adopción industrial de la Ingeniería del Software Dirigida por Modelos"* claramente expone que en esta ocasión se ha pretendido cubrir el lado industrial de MDE, esto es, mostrar casos de éxito de aplicación en las empresas e industria del software y valorar cuál puede ser el futuro de esta visión del desarrollo de software que se centra en la utilización sistemática de modelos y en la automatización para aumentar la calidad y productividad.

Al igual que con las monografías anteriores hemos procurado contar con editores invitados que sean personas reconocidas internacionalmente en MDE. Esta vez hemos tenido la suerte de contar con Jordi Cabot, responsable del grupo AtlanMod (INRIA, *École de Mines* de Nantes), uno de los investigadores de mayor prestigio en la comunidad MDE en la actualidad como lo demuestran sus proyectos y publicaciones.

2. Modelado en Ingeniería del Software

Al igual que en otras disciplinas científicas y de ingeniería, los modelos han jugado un papel esencial en el área de la Ingeniería del Software, aunque menor del qué sería deseable. Un modelo es una representación de un aspecto de la realidad cuya finalidad es ayudar a comprender y razonar sobre ella, por lo que en su creación se ignoran los detalles no relevantes para ese fin (proceso de abstracción).

En los procesos de producción, los modelos permiten investigar, verificar, discutir y documentar las propiedades de los productos antes de que ellos sean fabricados, e incluso en algunos casos permiten automatizar el proceso de producción. De forma similar, en el caso de la ingeniería del software los modelos han sido utilizado desde los orígenes de la disciplina y han permitido modelar aspectos de los sistemas software como el comportamiento (por ejemplo, máquinas de estado o redes de Petri) o los datos de las aplicaciones (por ejemplo, modelos E/R). Los modelos pueden crearse de manera informal (por ejemplo, cuando se usan como medio de comunicación o de razonamiento sobre el problema) o formal (cuando deben ser procesados por herramientas software).

Un hito importante en la historia del modelado de software fue la aparición de UML (*Unified Modeling Language*) [3] en 1997, lo que supuso la disponibilidad de una notación estándar para representar los diferentes aspectos de las aplicaciones orientadas a objetos, y con ello poner fin a la gran confusión que reinaba en aquellos años en los que existían decenas de métodos cada uno con su propia notación.

UML ha sido una de las innovaciones con mayor éxito en la historia de la ingeniería del software como lo prueba la cantidad de libros y artículos publicados o que actualmente se enseñe en la mayoría de universidades del mundo.

Sin embargo, Grady Booch, uno de los creadores de UML, avisaba en 2002 [4] que a pesar de estar ampliamente extendido (estimó que un 7% de los 14 millones de profesionales del software lo usaba a diario, lo cual era un número importante) en la mayoría de ocasiones sólo se usaba para documentar, lo que no era su principal utilidad, ya que sus creadores habían pensado en él como un lenguaje que ayudase a razonar en la creación de código y sobre todo destinado a la generación automática de código a partir de modelos. Booch afirmó en esa conferencia que "ahora se inicia una nueva etapa de UML como lenguaje para MDE... MDE puede influir en la mejora de la calidad del

software al elevar los modelos a *ciudadanos de primera clase* como el código de los lenguajes de programación".

Doce años después de aquella conferencia, la difusión de UML ha crecido mucho en el mundo académico y muy poco en la industria donde todavía sigue siendo principalmente usado para documentar [5]. En cuanto al vaticinio sobre MDE, Booch acertó aunque se equivocó al considerar que su éxito estaría basado en el empleo de modelos UML ya que la comunidad MDE ha optado por crear lenguajes específicos del dominio (DSL), gráficos y textuales, en vez de usar UML debido a las dificultades para adaptarlo a dominios concretos con su mecanismo de perfiles [6][7].

3. Los modelos como código

A principios de la pasada década, el interés de la comunidad software se desplazó desde la tecnología de objetos al desarrollo basado en modelos. Las clases, componentes, *frameworks* y demás tecnología surgida alrededor de la orientación a objetos evidenciaron sus limitaciones para alcanzar los ansiados niveles de automatización necesarios para disponer de una verdadera industria del software. Entonces empezaron a surgir enfoques que planteaban la generación automática de código a partir de modelos [8][9][10].

La idea que subyacía en ellos era el uso de lenguajes de modelado específicos del dominio (DMSL o simplemente DSL) para reducir la cantidad de código escrito con lenguajes de programación. Esto supondría un aumento de productividad del mismo orden de magnitud o incluso mayor que el que supuso la aparición de los lenguajes de programación y el consiguiente abandono de los lenguajes ensambladores.

Además, fue surgiendo un consenso al considerar el *metamodelado* como el formalismo más apropiado para definir los DSLs, de hecho UML fue formalmente definido por medio del lenguaje de metamodelado MOF. Cabe destacar que la idea de un desarrollo basado en modelos y DSL creados a partir de metamodelos ya había sido plasmada en dos herramientas de finales de los noventa como son MetaEdit[1] y GME[2].

OMG presentó en noviembre de 2001 un enfoque de desarrollo de software basado en modelos que denominó MDA (*Model-Driven Architecture*)[3]. Sin duda ha sido la propuesta más conocida y ha jugado un papel clave en sentar las bases de una nueva disciplina de la ingeniería del software: la *Ingeniería del Software Dirigida por Modelos* (MDE).

MDA planteaba el diseño e implementación de cadenas de transformación de modelos como medio de automatizar la producción de código. Los modelos conformaban con metamodelos y eran creados con lenguajes de modelado. Realmente MDA, como especificación surgida de OMG, propugnaba el uso de sus estándares: MOF para crear metamodelos, QVT para escribir transformaciones modelo-a-modelo, Mof2Text para escribir transformaciones modelo-a-texto y perfiles UML para crear los DSML. Sin embargo, el enfoque es independiente de los lenguajes y herramientas utilizados en las soluciones.

En el mundo académico, MDA motivó un gran interés por el desarrollo basado en modelos y conforme maduraron las ideas se evidenció que MDA era sólo una visión, entre las muchas que son posibles, en el ámbito de MDE, como veremos más adelante cuando comentemos los posibles escenarios de uso.

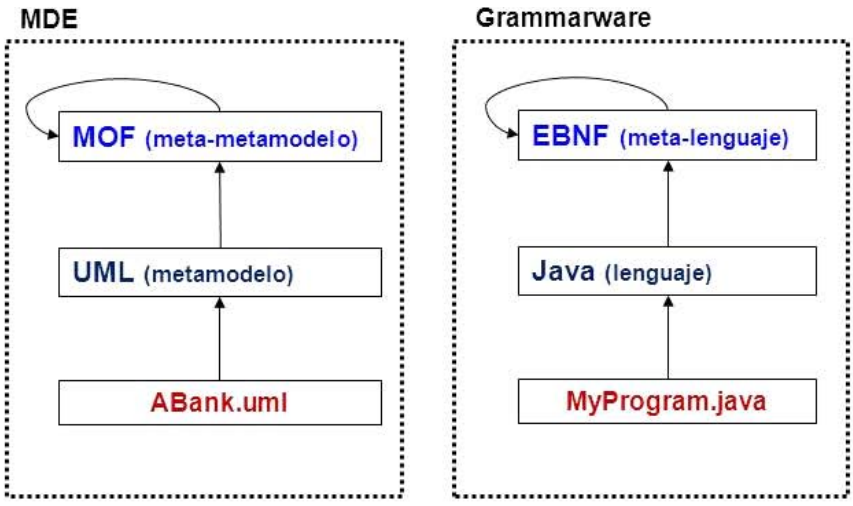

Figura 1. Comparación entre metamodelos y gramáticas.

4. Los principios de MDE

Como hemos indicado, MDE es el área de la ingeniería del software que se ocupa de la utilización sistemática de los modelos en las diferentes etapas de los procesos de producción de software con el fin de mejorar la productividad a través de un incremento en los niveles de abstracción y automatización.

Una solución MDE se basa en la aplicación de transformaciones sobre modelos. Estos modelos son expresados por un lenguaje de modelado que se define por medio de un metamodelo. Por lo tanto, podemos considerar que los cuatro elementos clave de MDE son: *modelo*, *metamodelo*, *transformación de modelos* y *lenguaje de modelado*.

La **figura 1** ilustra la relación entre modelo y metamodelo a través de un paralelismo entre los conceptos de programa y gramática. De la misma forma que un programa es conforme a una gramática, un modelo es conforme a un metamodelo que determina su estructura.

Además, del mismo modo que una gramática es expresada por una notación especial, como es el caso de EBNF (*Extended Backus-Naur Form*), también son necesarios lenguajes de metamodelado para expresar los metamodelos, MOF y Ecore son los más conocidos.

La **figura 1** expresa, por lo tanto, que un modelo UML llamado "Abank.uml" es conforme a un metamodelo de UML que ha sido definido con MOF y se establece el paralelismo con un programa Java "MyProgram.java" que es conforme a la gramática de Java que se ha expresado con el lenguaje EBNF.

Cada metamodelo define un lenguaje de modelado asociado, de la misma forma que un lenguaje de programación es definido por una gramática. En el caso de los lenguajes de modelado se diferencia entre la sintaxis abstracta (metamodelo que expresa los conceptos del lenguaje y relaciones entre ellos) y la sintaxis concreta que expresa la notación del lenguaje.

Todos los modelos especificados con un lenguaje de modelado deberán ser una instanciación correcta de su metamodelo. Se suele utilizar el término "conforma con" para expresar estas relaciones de instanciación.

Los lenguajes de modelado pueden ser gráficos, textuales o híbridos según la naturaleza de la representación que permiten crear. Por ejemplo, UML es un lenguaje híbrido en el que predomina la parte gráfica, aunque también existe una notación para expresar los modelos de forma textual. Se puede

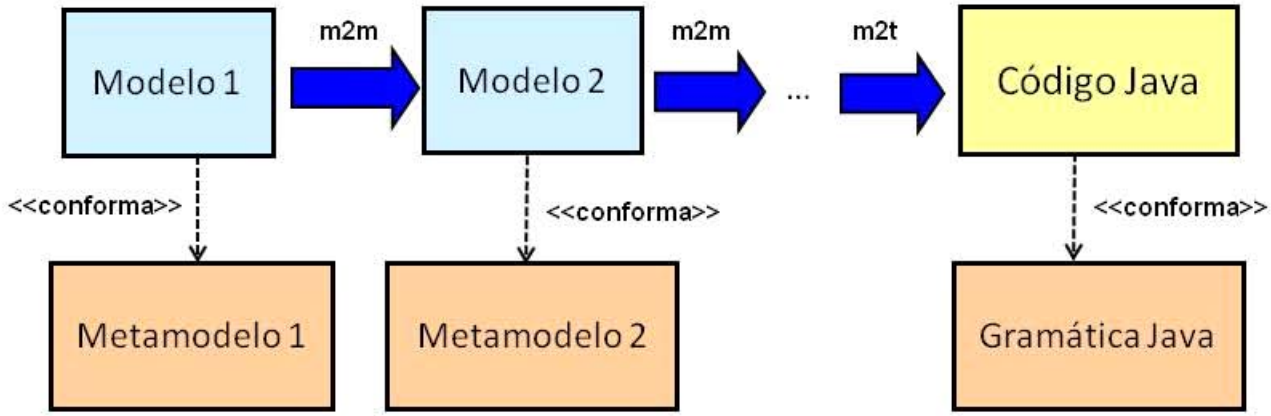

Figura 2. Cadena de transformación de modelos.

llegar a dar el caso de que un mismo lenguaje incluya diferentes notaciones alternativas (ejemplo una gráfica y una textual) aunque todas deberán ser conformes al metamodelo del lenguaje.

Existen dos grandes categorías de lenguajes de modelado:

- Lenguajes específicos del dominio (DSL) que son lenguajes diseñados específicamente para expresar modelos para un dominio, contexto o empresa. Un ejemplo sería un DSL diseñado por un fabricante de *smartphones* para expresar modelos con los que generar código para una plataforma de creación de aplicaciones móviles.
- Lenguajes de modelado de propósito general que son lenguajes diseñados para expresar modelos para cualquier dominio y contexto. Un ejemplo típico sería UML.

Los DSLs como alternativa a los lenguajes de programación de propósito general (GPL) han existido desde los primeros días de la programación. Ejemplos de DSLs utilizados desde hace muchos años son SQL, make, Excel y VHDL. Sin embargo, con la aparición de MDE ha crecido de forma considerable el interés por ellos. Mientras que tradicionalmente los DSLs se han construido con la tecnología (*Grammarware*) usada para crear GPLs, en el contexto de MDE se están creando con la propia tecnología MDE (*Modelware*): modelos, metamodelos y transformaciones de modelos.

Una solución MDE para automatizar una tarea de desarrollo de software incluye una cadena de transformaciones en la que dado un modelo fuente se generan artefactos software de una aplicación (por ejemplo, código fuente o archivos XML). Esta cadena está formada por un conjunto de transformaciones modelo-a-modelo (M2M) más una transformación modelo-a-texto (M2T) final,como muestra la figura 2.

El objetivo de las transformaciones M2M es reducir el salto semántico desde el metamodelo al que conforma el modelo fuente al código mediante el uso de modelos intermedios. Por lo tanto, cuando este salto semántico es pequeño la cadena puede reducirse a una única transformación m2t. Un tercer tipo de transformaciones de modelos, que son usadas con bastante menos frecuencia que las dos anteriores, son las transformaciones texto-a-modelo (T2M) que generan un modelo a partir de un documento textual (por ejemplo, código fuente), las cuales son necesarias, por ejemplo, cuando se usa MDE en migración de aplicaciones.

Las transformaciones se pueden implementar en un lenguaje de programación como Java haciendo uso de un API de manejo de modelos como EMF (*Eclipse Modeling Framework*), pero se suelen escribir con alguno de los muchos DSLs que se han definido en los últimos años para tal fin, entre los que destacan: ATL[4] y QVT[5] para transformaciones m2m; Acceleo[6], y Mofscript[7] para transformaciones m2t, y Gra2MoL [12] para transformaciones t2m.

Los lenguajes de transformaciones modelo-a-modelo proporcionan construcciones para definir correspondencias entre elementos de un metamodelo fuente y otro destino, de modo que la ejecución de la transformación pueda generar un modelo destino a partir de uno de entrada.

Los lenguajes de transformación modelo-a-texto proporcionan construcciones que permiten recorren un modelo de acuerdo a la estructura definida en el metamodelo y expresar qué texto se debe generar de acuerdo a los elementos del metamodelo accedidos. Finalmente, el lenguaje Gra2MoL permite expresar una correspondencia entre una gramática y un metamodelo de modo que la ejecución de la transformación pueda generar elementos del modelo destino a partir del texto de entrada.

Debe notarse que aunque distintos, todos los elementos del *Modelware* (espacio tecnológico MDE) son modelos: un meta-modelo es un modelo con un objetivo específico y una transformación también es un modelo cuando se escribe con un lenguaje definido con un metamodelo. Esta unificación de conceptos permite tratar de una forma homogénea todos los elementos involucrados en un proyecto y es una de las principales propiedades de MDE.

El lector puede ampliar detalles sobre MDE en [13] que ofrece una visión completa de todos los conceptos, incluso de algunos temas avanzados. Una introducción a MDE en español y más centrada en las herramientas que en los conceptos puede encontrarse en [14].

5. Casos de uso de MDE

Los cuatro principales escenarios en los que se usa MDE son: automatizar el desarrollo de software, modernización dirigida por modelos, interoperabilidad de sistemas y uso de modelos en tiempo de ejecución (conocido como Models@runtime) [13].

El uso más conocido y extendido de MDE consiste en automatizar alguna tarea del ciclo de vida de desarrollo de una aplicación, lo que se suele denominar "desarrollo de software dirigido por modelos" (en inglés *Model-Driven Sofware Development*, MDSD o simplemente MDD).

Se suele partir de un modelo que representa algún aspecto del sistema software (con frecuencia un modelo del dominio con las entidades del negocio, sus atributos y las relaciones entre ellas) y a partir de este modelo se generan artefactos software de la aplicación final a través de una cadena de transformaciones de modelos como

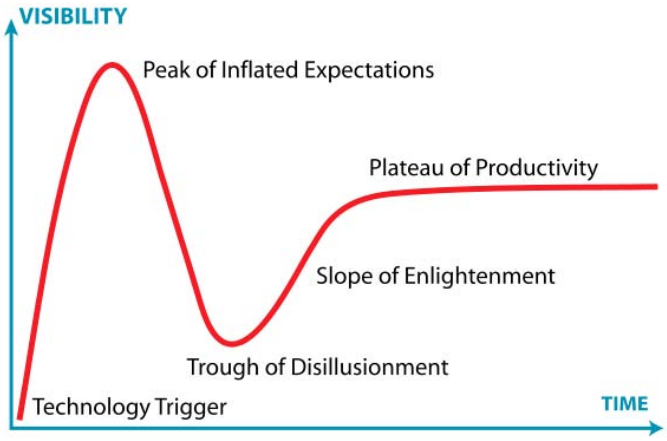

Figura 3. Ciclo "*hype*" de la tecnología.

hemos visto anteriormente e ilustrado en la **figura 2**.

Las técnicas MDE también pueden usarse para automatizar tareas de evolución de software como son una ingeniería inversa o una obtención de métricas. En estos escenarios se parte de artefactos software y es preciso obtener modelos que los representen para poder aplicar transformaciones de modelos que automaticen las tareas de evolución.

En 2003, OMG lanzó la iniciativa ADM (*Architecture-Driven Modernization*)[8] cuyo objetivo es la definición de un conjunto de siete metamodelos para tareas comunes en modernización o reingeniería de software.

Entre estos metamodelos, KDM (*Knowledge Discovery Metamodel*) juega un papel esencial en la propuesta ya que está destinado a representar el código de una aplicación a diferentes niveles de abstracción que van desde la representación de instrucciones de un GPL hasta la representación de reglas de negocio. Se trata, pues, de un metamodelo muy grande organizado en cuatro capas (*Infrastructure*, *Program Elements*, *Resource* and *Abstraction*).

ASTM (*Abstract Syntax Tree Metamodel*) es un metamodelo que complementa KDM y está destinado a representar el código en forma de árbol de sintaxis abstracta (AST). Una explicación detallada sobre cómo usar KDM y ASTM para el modelado de código PL-SQL puede encontrarse en [15], dónde se presenta un caso de estudio de obtención de métricas.

Otro metamodelo disponible de ADM es SMM (*Software Metrics Metamodel*) para representar métricas y también se ha definido AFP (*Automated Function Point*) un método para automatizar la obtención de los puntos de función.

Los modelos también facilitan la integración de sistemas, por ejemplo herramientas, al poder ser usados como una "lingua franca" o representación intermedia [13] que simplifique la transformación e importación/exportación de los datos de un sistema a otro.

Models@runtime es, sin duda, el uso menos extendido de los modelos y se refiere al uso de modelos en tiempo de ejecución como mecanismo de control de sistemas software. Estos modelos incluso pueden ser cambiados dinámicamente como sucede en el caso de sistemas adaptativos (*self-adaptive systems*) que son capaces de adaptar su comportamiento (y con ello el del sistema que controlan) a los cambios en su entorno.

6. Adopción en la industria

Cada vez aumentan las voces de investigadores, profesionales y consultores que propugnan el uso de MDE. Sin embargo, su adopción en la industria de MDE es muy baja, de hecho el modelado con UML o cualquier otra notación (sin considerar la automatización) es una actividad adoptada en pocas empresas [5][13][16] y que cuando se realiza no es una actividad central del proceso de desarrollo [17].

No obstante, se pueden percibir señales de que estamos en el camino de una adopción exitosa de MDE en la industria, como lo es el creciente interés por los DSL, como lo prueba el gran número de libros publicados en los últimos años, (entre los que podemos destacar [18] y [19]), la consolidación de MDE como una disciplina centrada en escenarios como los cuatro comentados arriba (no centrada exclusivamente en la generación de código a partir de modelos) y la disponibilidad de un buen número de herramientas en torno al *framework* EMF (Eclipse está jugando un papel fundamental implementando los estándares de OMG).

Bran Selic analizó la adopción industrial de MDE en [16] y señaló que los factores que la dificultaban eran de tres tipos: técnicos (necesidad de mejorar la base teórica y de disponer de herramientas usables y robustas), culturales (la mentalidad conservadora de las empresas, la inercia al cambio y la falta de concienciación de las capacidades de MDE) y económicos (las empresas se centran en el ROI en el corto plazo y hay dificultades para encontrar profesionales bien preparados en MDE).

Frente a esto, señalaba que era preciso un mayor esfuerzo en investigación, una labor de transferencia de tecnología, que las universidades ofrezcan una buena formación en MDE y la aparición de más estándares. Cabe destacar que Selic, al igual que muchos expertos, considera que con diferencia los factores culturales y económicos son más importantes que los técnicos.

MDE puede estar siguiendo el típico ciclo de sobre-expectación (*hype cycle*, en inglés) que caracteriza la adopción industrial de las tecnologías y que muestra la **figura 3**, en el que después de un crecimiento debido a las expectativas infladas (en este caso probablemente debido a las expectativas generadas en torno a UML), se cae debido a que se entra en una etapa de desilusión, para luego crecer de nuevo hasta llegar a una estabilización cuando la tecnología se extiende y adopta.

Quizá ahora podamos estar al inicio de esa segunda etapa de crecimiento en la que se conoce bien cómo MDE puede ser usada para que las empresas obtengan beneficios.

7. Resumen de la monografía

Los artículos que conforman esta monografía intentan profundizar en todos los aspectos aquí mencionados sobre todo desde un punto de vista industrial.

Creemos que la adopción de MDE ofrece ahora un montón de oportunidades y una ventaja competitiva para las empresas que se decidan a cambiar su manera de ver la ingeniería de software con respecto a las organizaciones más conservadoras que prefieran esperar a que sea evidente el éxito de MDE. Esperamos que los artículos de esta monografía ayuden a tomar esta decisión.

En *Ingeniería del Software con Modelos: Estado actual y futuros retos*, **Richard Paige** da el contexto necesario para entender la situación actual de MDE, explicando las partes de MDE que han funcionado bien (y también las que no tanto) y defendiendo la idea de que hay que ir más allá de MDE y focalizarse en lo que él llama Ingeniería de Modelos.

En *ARTIST: Una solución global para la modernización de software hacia el cloud*, **Clara Pezuela** describe cómo el uso de técnicas de ingeniería inversa basada en

modelos permite automatizar una gran parte del proceso de migración de aplicaciones a un entorno *cloud*, donde el proceso de migración no se limita sólo a la parte funcional sino que cubre también el análisis de viabilidad, el *testing*, la certificación final, etc.

SmartEA : una herramienta de arquitectura empresarial basada en MDE por ***Stéphane Drapeau***, ***Frédéric Madiot***, ***Jean-François Brazeau*** y ***Pierre-Laurent Dugré*** nos descubre como MDE puede ayudar a la empresa más allá del ámbito estrictamente del software. SmartEA muestra cómo utilizar técnicas MDE para especificar y analizar la arquitectura empresarial (*Enterprise Architecture*, en inglés) de una organización, proporcionando a cada miembro la visión que necesita.

En *Optimización del Rendimiento de Aplicaciones ABAP*, ***Orlando Ávila-García*** nos explica como han aplicado MDE en OpenCanarias para optimizar el rendimiento de aplicaciones ya existentes. Aunque a simple vista podría parecer que esta tarea se debería realizar directamente operando sobre el código fuente, Orlando nos explica como el uso de modelos nos permite razonar a más alto nivel de abstracción obteniendo así mejores resultados.

Quince años de MDE en Software Front-ends: de WebML a WebRatio e IFML de ***Marco Brambilla*** y ***Stefano Butti*** nos explica la historia del estándar IFML (*Interaction Flow Modeling Language*), un lenguaje para el modelado de interfaces de usuario (un aspecto muy poco tratado hasta el momento) y como su uso conjuntamente con técnicas de generación de código puede aumentar drásticamente la productividad en el desarrollo de aplicaciones web.

Finalmente, en *Adopción Industrial de la Ingeniería del Software Dirigida por Modelos para Software Empotrado*, ***Aitor Murguzur***, ***Xabier De Carlos***, ***Xabier Mendialdua*** y ***Salvador Trujillo*** nos cuentan cómo para trabajar en dominios específicos como el suyo (aerogeneradores en esta experiencia que nos relatan) la mejor solución fue crear un entorno de modelado completamente ad-hoc, incluyendo la creación de un DSL y sus editores correspondientes.

Referencias

[1] Jesús J. García Molina, Ana Moreira, Gustavo Rossi (eds). "UML e Ingeniería de Modelos". *Novática, No. 168, marzo-abril, 2004*. También publicada en: "*UML and Model Engineering*", *Upgrade Journal, Vol. V, Num. 2* (2004).

[2] Jean Bézivin, Antonio Vallecillo, Jesús García Molina, Gustavo Rossi (eds). "Desarrollo de Software Dirigido por Modelos". *Novática, No. 192, marzo-abril, 2008*. También publicada en: "Model-Driven Software Development", *Upgrade Journal, Vol. IX, Num. 2* (April, 2008).

[3] James Rumbaugh, Grady Booch, Ivar Jacobson. *El lenguaje unificado de modelado*, 2ª Edición. Pearson, 2006.

[4] Grady Booch. Growing the UML. *Software and System Modeling 1* (2): pp. 157-160 (2002).

[5] Marian Petre. UML in practice. *International Conference on Software Engineering, ICSE 2013*: pp. 722-731.

[6] Markus Völter. MD* Best Practices. *Journal of Object Technology 8*(6): pp. 79-102 (2009).

[7] Steven Kelly, Juha-Pekka Tolvanen. *Domain-Specific Modeling - Enabling Full Code Generation*. Wiley 2008.

[8] Anneke Kleppe, Jos Warmer, Wim Bast. "MDA Explained", Addison-Wesley, 2003.

[9] Jack Greenfield, Keith Short. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons, 2004.

[10] Sergey Dmitriev. "Language-Oriented Programming. The next programming paradigm". *JetBrains, November, 2004*. <http://www.onboard.jetbrains.com/is1/articles/04/10/lop/>. Último acceso: 30 de junio de 2014.

[11] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks. "Eclipse Modeling Framework", 2nd Edition, Addison-Wesley, 2008.

[12] Javier Luis Cánovas Izquierdo, Jesús García Molina. Extracting models from source code in software modernization. *Software and System Modeling 13*(2): pp. 713-734 (2014).

[13] Marco Brambilla, Jordi Cabot, **Manuel Wimmer**. "*Model-Driven Software Engineering in Practice*", Morgan & Claypool Publishers 2012.

[14] Jesús J. García Molina et al. (eds). "*Desarrollo de Software Dirigido por Modelos: Conceptos, técnicas y herramientas*", Editorial RA-MA, 2013.

[15] Javier Luis Cánovas Izquierdo, Jesús García Molina. An Architecture-Driven Modernization Tool for Calculating Metrics. *IEEE Software 27*(4): pp. 37-43 (2010).

[16] Bran Selic. What will it take? A view on adoption of model-based methods in practice. *Software and System Modeling 11*(4): pp. 513-526 (2012).

[17] Andrew Forward, Timothy C. Lethbridge. Problems and Opportunities for Model-Centric Versus Code-Centric Development: a Survey of Software Professionals. *Proceedings of the International Workshop on Models in Software Engineering (MiSE´08) @ International Conference on Software Engineering, ICSE'08*, pp. 27–32, New York, 2008. ACM.

[18] Martin Fowler, Rebecca Parsons. "*Domain-Specific Languages*". Addison-Wesley, 2010. ISBN-10: 0321712943.

[19] Markus Voelter. "*DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*". dslbook.org, 2013.

Notas

[1] <http://www.metacase.com/products.html>.
[2] <http://www.isis.vanderbilt.edu/Projects/gme>.
[3] <http://www.omg.org/mda/>.
[4] <http://www.eclipse.org/atl/>.
[5] <http://www.omg.org/spec/QVT/1.1/>.
[6] <http://www.eclipse.org/acceleo/>.
[7] <http://www.eclipse.org/gmt/mofscript/>.
[8] <http://adm.omg.org/>.