Propuesta para generar artefactos basados en modelos*[1]

[Proposal for generation of software model-based artifacts]

Vicente GARCÍA DÍAZ[2], Edward Rolando NÚÑEZ VALDEZ[3], B. Cristina PELAYO GARCÍA-BUSTELO[4], Jordan PASCUAL ESPADA[5], Carlos Enrique MONTENEGRO MARÍN[6]

Recibo: 26.01.2012 - Aprobación: 20.05.2012

Resumen

La Ingeniería Dirigida por Modelos es una aproximación de desarrollo en continua evolución. Prueba de ello son los numerosos estándares que están surgiendo y la reciente aparición de herramientas que facilitan el trabajo con este reciente paradigma de la ingeniería del software. Al trabajar con modelos, un aspecto clave es la generación automática de código de menor nivel de abstracción. Sin embargo, la forma de llevar a cabo dicha generación no tiene en cuenta la evolución de los sistemas, y esa

* Modelo para citación de este Artículo de investigación científica y tecnológica: GARCÍA DÍAZ, Vicente; NÚÑEZ VALDEZ, Edward Rolando; PELAYO GARCÍA-BUSTELO, B. Cristina; PASCUAL ESPADA, Jordan y MONTENEGRO MARÍN, Carlos Enrique (2012). Propuesta para generar artefactos basados en modelos. En: Ventana Informática. No. 26 (ene. – jun., 2012). Manizales (Colombia): Facultad de Ciencias e Ingeniería, Universidad de Manizales. p. 45-60. ISSN: 0123-9678

1 Artículo proveniente del proyecto *MDCI: Model-Driven Continuous Integration*, ejecutado en el periodo *2009-2011*, e inscrito en el grupo de investigación *MDE-OOTLAB.*

2 Ingeniero Técnico en Informática de Sistemas; Ingeniero en Informática; PhD. en Informática. Profesor en el Departamento de Informática. Universidad de Oviedo (España). Correo electrónico: garciavicente@uniovi.es

3 Licenciado en Informática; Master en Ingeniería de Software; PhD. en Informática. Investigador en el Departamento de Informática. Universidad de Oviedo (España). Correo electrónico: edwardnu@gmail.com

4 Ingeniera Técnica en Informática; Ingeniera en Informática, PhD. Profesora en el Departamento de Informática. Universidad de Oviedo (España). Correo electrónico: crispelayo@uniovi.es

5 Ingeniero Técnico en Informática de Gestión; Máster en ingeniería Web. Investigador en el Departamento de Informática. Universidad de Oviedo (España). Correo electrónico: jordansoy@gmail.com

6 Ingeniero de sistemas; Magister en Ciencias de la información y las comunicaciones; PhD. Profesor en la Universidad Distrital Francisco José de Caldas, Bogotá (Colombia). Correo electrónico: cemontenegro@udistrital.edu.co

es la razón por lo que el proceso es lento y repetitivo. En este trabajo se presenta una propuesta para generar código incrementalmente a partir de modelos, de forma que se minimice el impacto sobre aplicaciones que ya podrían estar en funcionamiento y se permita obtener la evolución exacta que han tenido los sistemas desde su origen.

Palabras Clave: *Evolución, metamodelo, modelo, generación de artefactos.*

Abstract

The Model-Driven Engineering is a software development approach that continues evolving. In fact, there are some emerging standards and tools that facilitate working with this new paradigm of software engineering. Working with models, a key point is the automatic generation of source code of lower level of abstraction. However, the natural evolution of systems is not taken into account and that is the reason for which the process is usually slow and repetitive. In this work, we are going to show a proposal for incremental generation of source code from models. That way, we will minimize the impact on deployed applications and we will make the traceability of the evolution of systems.

Keywords: *Artifacts generation, evolution, metamodel, model.*

Introducción

La Ingeniería Dirigida por Modelos, MDE, de acuerdo con Kent (2002), es un paradigma relativamente novedoso en el desarrollo de software que ha ido ganando importancia desde su aparición, sobre todo, según Clements & Northrop (2002), en el desarrollo de líneas o familias de productos software. Para Seidewitz (2003), consiste en utilizar modelos como elementos principales en el desarrollo. Así, se logra un mayor nivel de abstracción que el logrado por lenguajes de programación tradicionales (p.e., C++, Java). Dichos modelos, generalmente, se transforman automáticamente, reduciendo su nivel de abstracción, a artefactos que pueden ser, por ejemplo, código fuente Java o documentación en formato HTML, facilitando así la generación de sistemas software, afirman Völter & Stahl (2006). La manera de proceder es la siguiente: un desarrollador crea un modelo y el generador se encarga de crear los artefactos que se correspondan con el modelo. Tales artefactos podrían ser por si mismos un sistema software completo o un subsistema.

Los modelos, aseguran Mens et al. (2005), están sometidos a los mismos desafíos que cualquier sistema software durante su evolución y están sujetos a modificaciones a lo largo de todo su ciclo de vida debido, por ejemplo, a cambios en el modelo de 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: 1- no se optimizan los recursos computacionales de los que se dispone, 2- se dificulta el mantenimiento de aplicaciones en producción y 3- se imposibilita la realización de trazabilidad, según Seibel, Hebig & Giese (2012), de los artefactos generados.

La generación incremental de artefactos en MDE hace referencia a la capacidad que pueden tener los generadores de artefactos para no tener que reconstruir todos los artefactos cada vez que se hace una modificación en un modelo, considerándose los modelos como los elementos clave a partir de los cuales se construyen sistemas software. La idea puede verse como una rama alternativa a otros trabajos que ya empiezan a resaltar la importancia de una evolución controlada dentro de la ingeniería dirigida por modelos, según consideran Karanam & Akepogu (2012).

Hay que tener en cuenta que en la generación incremental de artefactos no hay intervención manual y nada tiene que ver con los sistemas en los que se mantienen las modificaciones realizadas manualmente por los desarrolladores, típicamente utilizando técnicas como las regiones protegidas de código.

El objetivo de este trabajo es ofrecer una propuesta que permite generar artefactos software de manera incremental a partir de modelos software. De esa forma se evita tener que regenerar todos los artefactos cada vez que surge un cambio en un modelo. Así, se aumenta la mantenibilidad de las aplicaciones y se reduce el impacto que sobre ellas pueden tener los cambios en los modelos.

El resto del documento está estructurado de la siguiente forma: en la Sección 1 se detalla qué es la generación incremental de artefactos. En la Sección 2 se presenta un estado del arte y trabajos relacionados con la generación incremental de artefactos utilizando el paradigma MDE. En la Sección 3, se propone una solución para cumplir con los objetivos de crear un generador incremental MDE. En la Sección 4 se ofrece una validación del sistema presentado. Finalmente, en la Sección 5, se muestran las conclusiones y el futuro trabajo a ser realizado.

1. Concepto de generación incremental de artefactos

En la Figura 1, mediante una secuencia de tres pasos se ilustra cómo se comportaría un generador incremental de artefactos. Por simplicidad, se va a suponer 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 cuatro clases A, B, C y D. Sin embargo, en el caso 2, habría que eliminar las clases C y D porque ya no están en el diagrama los elementos que las representan. En el caso 3, sin embargo, habría que añadir una hipotética clase E puesto que se ha añadido al diagrama un elemento para representarla, junto con una modificación producida en la clase B con B'.

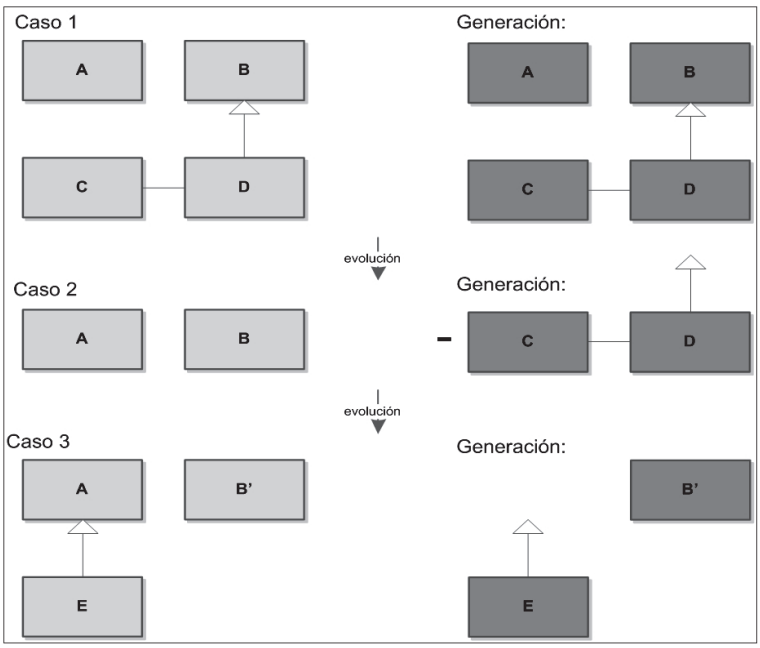

Figura 1. Ejemplo de secuencia de generación incremental de artefactos

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. 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 qué se ha cambiado y cuándo, siendo ello una poderosa herramienta de mantenimiento y de depuración.

A pesar de las ventajas que puede suponer su empleo, la generación incremental de artefactos es un tema que aún está poco tratado por la literatura científica. Así, el objetivo de este trabajo es ofrecer una propuesta para lograr una adecuada generación de artefactos utilizando una aproximación incremental.

2. Estado de arte y trabajos relacionados

Existen muchas herramientas dirigidas por modelos para generar artefactos. Sin embargo, se han encontrado carencias en lo referente a la generación incremental. En la Tabla 1 se listan algunas de las herramientas más conocidas, citadas en trabajos como Palacios et al. (2008), indicándose si tienen soporte nativo para realizar generaciones incrementales. Como se puede observar, actualmente no hay un soporte adecuado para generar artefactos incrementalmente.

Tabla 1. Herramientas de generación de artefactos

Herramienta	Web	Soporte incremental
AndroMDA	www.andromda.org	No
ArcStyler	www.arcstyler.com	No
ArgoUML	www.argouml.tigris.org	No
BlueAge	www.bluage.com	No
Borland Together	www.borland.com/products	No
CodeGenie	www.codegenie.se	No
EclipseUML	www.uml2.org	Limitado
EMP	www.eclipse.org/modeling	No
ExpertCoder	www.expertcoder.sourceforge.net	No
Jamda	www.jamda.sourceforge.net	No
Kennedy Carter iUML	www.kc.com	No
objectiF	www.microtool.de/objectif	Limitado
OlivaNova	www.sosyinc.com	No
OpenAmeos	www.openameos.org	No
OptimalJ	www.compuware.com	Limitado

Algunas herramientas ofrecen características de generación incremental de artefactos, aunque todas ellas de forma limitada. Un ejemplo es *objectiF*, que sólo regenera las partes que se tocan en el modelo, y que

tiene carencias como solo admitir modificaciones en tiempo de diseño y que no considera que un cambio en una parte del modelo podría afectar a otras partes.

OptimalJ 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. Mientras tanto, *EclipseUML* es una herramienta propietaria integrada en Eclipse que permite diseñar sistemas software utilizando varios tipos de diagramas UML. 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.

Por otra parte, la propuesta ofrecerá las siguientes características, que ninguna de las herramientas citadas soporta en su totalidad:

- 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 (p.e., *EclipseUML*, *OptimalJ*). Sin embargo, el usuario de la herramienta de modelado no tiene por qué querer o necesitar generar artefactos cuando modela su negocio. Por ejemplo, las herramientas de integración continua 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.
- Utilización de metamodelos prefijados. No es conveniente restringirse al empleo de un único metamodelo sin tener la posibilidad de incorporar otros (p.e., *EclipseUML*, *OptimalJ*).
- Generación de artefactos no ligada a ninguna plataforma. Algunas herramientas únicamente generan artefactos incrementalmente para una arquitectura y plataforma específica (p.e., *EclipseUML*, *OptimalJ*).
- Implementación abierta no comercial. Varias herramientas utilizan tecnologías propietarias comerciales (p.e., *EclipseUML*).

Además, siguiendo el principio de que el reúso del conocimiento y de la experiencia es muy importante, la propuesta presentada está basada en técnicas que ya se están utilizando e investigando en otros ámbitos de MDE. Generar artefactos incrementalmente requiere saber qué cambios se han producido en una versión de un modelo respecto a una versión anterior, es decir, requiere conocer la diferencia entre las versiones por las que pasa un modelo, consideran Oliveira, Murta & Werner (2005). Conocer dichas diferencias no es un proceso sencillo (Read & Cornell, 1977) y comprende tres fases (Brun & Pierantonio,

2008), de las que resultan útiles las dos primeras (es decir, cálculo de las diferencias entre modelos y representación de dichas diferencias), puesto que la visualización de las diferencias en un formato amigable de cara al usuario no es estrictamente necesaria debido a que los artefactos se generan automáticamente.

En las próximas líneas se explicará brevemente para qué se utiliza actualmente el cálculo de la diferencia entre modelos. Posteriormente, se tratarán los algoritmos actuales para calcular la diferencia entre modelos y las técnicas de representación de las diferencias detectadas por los algoritmos de cálculo. Como se verá, todo ello no será suficiente para generar incrementalmente artefactos de forma satisfactoria.

2.1 Ámbitos de aplicación de la diferencia entre modelos

Dentro de la gran cantidad de investigaciones que, relacionadas con MDE, según Oliveira, Murta & Werner (2005), surgen cada año, está un número considerable de trabajos enfocados a la necesidad de desarrollar sistemas de control de versiones específicos para modelos (MVCS) La idea, estima Tichy (1985), es suplir las carencias que, respecto a modelos, tienen los sistemas de control de versiones (MVS) tradicionales. Así, en trabajos de Reiter et al. (2007) o Altmanninger et al. (2008) se tratan la gestión de fuentes optimista utilizando modelos, en la que múltiples usuarios trabajan sobre un mismo activo, obligando al sistema a hacer *merging* con las diferentes modificaciones realizadas al mismo tiempo por varios desarrolladores. Por ello, se hace imprescindible el empleo de algoritmos de comparación de modelos para representar sus diferencias y/o realizar uniones entre ellos, según consideran Bendix & Emanuelsson (2008). Es de notar que existen otras muchas aplicaciones a partir de la diferencia entre modelos como por ejemplo la señalada en el trabajo de Mazanek & Rutetzki (2011).

2.2 Cálculo de las diferencias entre modelos

Para Kolovos et al. (2009), existen muchos algoritmos para calcular las diferencias entre distintos modelos, que pueden ser clasificados según diferentes aproximaciones:

- La aproximación basada en un identificador universal único estático (p.e., Alanen & Porres, 2003), que perdura durante todo la vida de cada elemento, permite realizar cálculos rápidos sin necesidad de ningún tipo de configuración. Sin embargo, no puede ser utilizada cuando dos modelos se construyen de forma independiente o cuando las tecnologías no soportan el mantenimiento de identificadores únicos.

- La aproximación basada en semejanzas (p.e., Lin, Gray & Jouault, 2007), trata a los modelos como grafos en los que buscar semejanzas entre los diferentes elementos. Típicamente requieren una configuración manual para establecer el peso de cada elemento del modelo.
- La aproximación específica del lenguaje (p.e., Toulmé, 2006), ofrece un gran rendimiento porque puede tener en cuenta la semántica específica de cada lenguaje pero tiene que pagar el precio de la dependencia y de la cantidad de código específico generado en cada caso. Generalmente, los trabajos que están incluidos en esta categoría también lo están en la basada en semejanzas.

2.3 Representación de las diferencias entre modelos

Para representar las diferencias entre modelos, han surgido diversas iniciativas durante los últimos años. Además, Cicchetti, Ruscio & Pierantonio (2007), 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, Frankel (2003) afirmó que para el éxito de MDE es totalmente fundamental el empleo de un metamodelo común. En cuanto a basarse en modelos, un principio también comúnmente aceptado es el de *Everything is a model* de Bézivin (2005), que junto con el empleo de metamodelos y meta-metamodelos, permiten trabajar con las numerosísimas herramientas que existen para trabajar bajo el paradigma MDE utilizando un mayor nivel de abstracción. Por otra parte, otras características como la compactabilidad, la transformatividad y la componibilidad pueden ser interesantes y ofrecen grandes posibilidades, pero no son necesarias para la generación incremental de artefactos.

Una de las técnicas más utilizadas para representar las diferencias entre modelos son los scripts de editado, utilizados en trabajos como el de Alanen & Porres (2003). Consisten en crear secuencias de operaciones primitivas que describen las modificaciones que ha sufrido un modelo respecto a otro. Se basa en una secuencia de transformaciones que permiten representar las diferencias como una Δ. Para describir cómo es la Δ, utilizan siete transformaciones elementales que han de cumplir con una serie de restricciones que limitan el empleo de esta técnica. Dichas transformaciones, que describen los cambios de un modelo respecto a otro, pueden ser: *new, del, set, insert, remove, insertAt o removeAt*.

Otra técnica muy utilizada es la basada en colores, tratada en trabajos como Ohst, Welle & Kelter (2003), consiste en mostrar gráficamente las diferencias mediante el empleo de un diagrama que 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, 2006), un proyecto ubicado dentro del *Eclipse Modeling Framework* (EMF) (Budinsky, Brodsky & Merks, 2003), el cual sirve para representar y visualizar las diferencias entre dos modelos de forma gráfica.

El último salto, se da en el trabajo de Cicchetti, Ruscio & Pierantonio (2007), cuya 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 actualizar, borrar o añadir elementos. Gracias al empleo de esta técnica 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, como las citadas por Beydeda, Book & Gruhn (2005).

3. Solución propuesta

Los trabajos citados en este artículo, 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 puede observar 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. De hecho, al representar la diferencia entre un modelo original y otro nuevo modelo realizado posteriormente, se podría averiguar, en principio, 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. Esto se debe a 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.

Respecto al algoritmo de cálculo de las diferencias entre modelos, no tiene especial importancia en este trabajo puesto que lo único que se requiere es que calcule correctamente las diferencias, pudiendo ser optimizado en el futuro. Además, la elección depende fundamentalmente de cada problema concreto (Kolovos et al., 2009). Para lograr que el algoritmo de

cálculo sea sencillo, genérico y que no requiera grandes esfuerzos de configuración, se ha utilizado un algoritmo basado en una función que depende únicamente de una propiedad de los elementos del modelo.

Por otra parte, para no perder la esencia y las ventajas del uso de una propuesta basada en modelos, la Figura 2 muestra la arquitectura propuesta basada en la representación de las diferencias entre modelos de Cicchetti, Ruscio & Pierantonio (2007). Se muestran los artefactos utilizados como entrada del proceso, los artefactos obtenidos en la salida y las herramientas que se están desarrollando durante este trabajo para llevarlo a la práctica en un prototipo. La idea, cuando se pretenden 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, podría estar gobernado por un MVCS y hacer uso de las técnicas para conocer las diferencias entre modelos expuestas en este y en otros trabajos. Un paso fundamental es extender el metamodelo original, creando uno con las nuevas 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 la necesidad de 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 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 para generar artefactos, en todo momento, se trabaja con una diferencia independiente del modelo (Konemann, 2009), la cual es auto-contenida, es decir, los cambios se describen sin hacer referencia a otros modelos.

La Tabla 2 muestra una comparación de todas las técnicas mencionadas con las características que podría tener cualquier técnica de representación de diferencias entre modelos. A las ya comentadas, se le ha añadido la propiedad generativa que, según Völter (2003), se entiende como la capacidad de generar artefactos de menor nivel de abstracción a partir de cada técnica. 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 todas las posibilidades debido a que se trabaja con todas las alternativas posibles. Dicha capacidad acarrea costes, perdiéndose varias propiedades, las cuales no son importantes en la generación incremental de artefactos. El empleo de tecnologías basadas en metamodelo y modelos son suficientes para posibilitar el empleo de incontables técnicas y herramientas que existen para trabajar bajo el paradigma MDE.

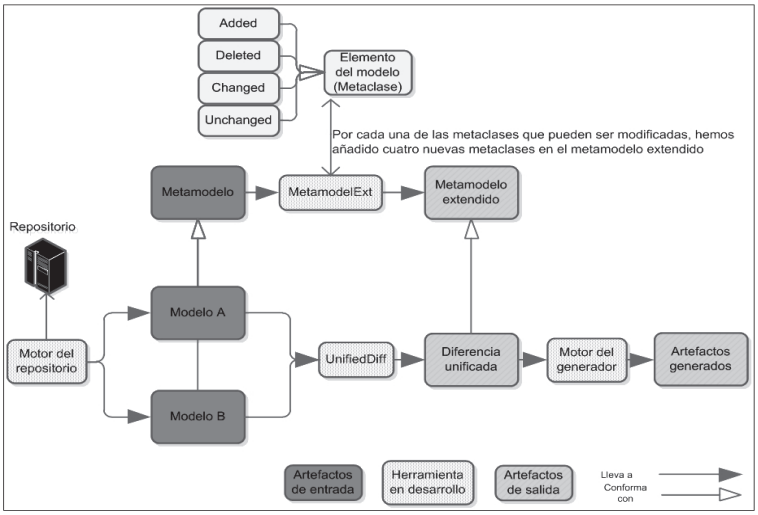

Figura 2. Arquitectura propuesta

4. Validación

Para validar los resultados proporcionados por la propuesta, se ha creado un caso de estudio realizado a partir de un cliente de *Twitter*. La aplicación está compuesta por librerías que proporcionan funcionalidad. Para cambiar dicha funcionalidad se ha construido un lenguaje de dominio específico que permite realizar configuraciones sobre la aplicación gracias a la generación de código con las modificaciones pertinentes. Algunas de las opciones que se pueden cambiar son:

- Añadir o eliminar pestañas de búsqueda.
- Modificar los mensajes ofrecidos al usuario.
- Adaptar la forma en la que se muestran los contactos de los usuarios.
- Alterar la forma en la que se muestran los mensajes a los usuarios.
- Cambiar la *metainformación* de los ensamblados del software.

Tabla 2. Técnicas para representar las diferencias entre modelos (Construida a partir de Cicchetti, Ruscio & Pierantonio, 2007)

Propiedad	Scripts de editado	Coloreado	Modelo de la diferencia	Diferencia unificada
Basada en modelos		SI	SI	SI
Independencia del metamodelo		SI	SI	SI
Generativa				SI
Compacta			SI	
Capacidad de composición	SI		SI	
Transformativa			SI	

Así, con el objetivo de llevar a cabo la evaluación, se han realizado en orden secuencial 15 cambios en los modelos de entrada, causando una evolución (Figura 3).

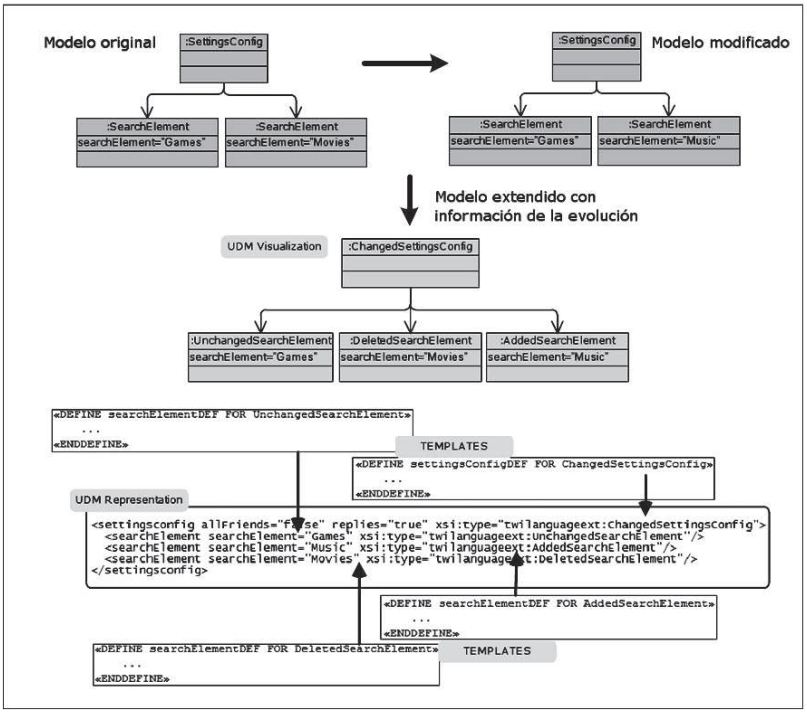

Figura 3. Evolución de los modelos y generación basada en plantillas

Cuando se realizan los cambios, se crea la versión extendida del metamodelo (Figura 2), a partir de la cual se generan los nuevos artefactos utilizando plantillas de generación de código Xpand y el motor de generación de Xtext (Eclipse Foundation, 2011).

Las generaciones que han sido llevadas a cabo como respuesta a los cambios sugieren los beneficios que, extrapolando estos resultados a otros dominios, proporciona la generación incremental de artefactos. Por ejemplo, con la generación incremental de artefactos no es necesario detener la aplicación en casi ningún caso (sólo cuando los cambios afectan al archivo ejecutable). Además, realizando los cambios incrementalmente es posible llevar una trazabilidad de la evolución por la que han pasado los modelos y, por lo tanto, el sistema. Por último, el tamaño de los artefactos generados ha sido reducido de 19,95 KB a 0,09 KB, el tiempo de generación ha descendido de 11,94 segundos a 8,82 segundos, y el tiempo de despliegue se ha recudido de 2,9 segundos a 1,31 segundos.

5. Conclusiones y trabajos futuros

En este trabajo se ha presentado una propuesta para generar artefactos de forma incremental, de modo que se optimizan los recursos computacionales, se minimiza el impacto sobre aplicaciones que ya podrían estar en producción y se permite la realización de trazabilidad a lo largo del ciclo de vida de las aplicaciones. Dichas características son beneficiosas y ofrecen un salto cualitativo en cuanto a la generación de artefactos utilizando el paradigma de desarrollo MDE.

Gracias al empleo de técnicas incrementales, no hay que regenerar la aplicación por completo a partir de cada modificación (como es lo habitual). Así, al haber realizado 15 modificaciones en los modelos de entrada, utilizando técnicas tradicionales se han generado un total de 19,95 KB (1,33 KB por cada generación). Mediante la generación incremental de artefactos la media de KB por generación se reduce a 0,006. Considerando que la aplicación de prueba tiene un tamaño muy reducido se espera que para próximas pruebas que aplicaciones de gran tamaño los resultados sean proporcionales. Tan alta reducción de tamaño podría servir, entre otras muchas cosas, para ahorrar espacio o enviar a través de una red los datos en un tiempo mucho más reducido.

El tiempo medio de generación para las 15 modificaciones se ha visto recudido un 26%. Aunque dicho porcentaje es considerablemente alto, dista de ser tan bueno como en el caso del tamaño de los artefactos. La razón es que la técnica expuesta precisa de operaciones internas para crear el metamodelo extendido a partir del cual generar los artefactos de forma incremental. En cualquier caso, pese a las operaciones extra requeridas, el porcentaje ganado de tiempo es muy elevado.

En cuanto al tiempo medio de despliegue de las aplicaciones para las 15 modificaciones, se ha reducido en un 54,82%, es decir, se ha reducido en más de la mitad el tiempo de despliegue gracias a la nueva técnica. El beneficio podría incluso haber sido superior, pero en las aplicaciones suelen existir dependencias que hacen que aunque un artefacto no haya vuelvo a ser generado, si que puede ser necesario volver a compilarlo cuando depende de otros elementos que sí se han regenerado al realizar cambios en un modelo. No obstante, el beneficio obtenido para el caso de prueba es muy alto.

Un futuro trabajo esencial será la realización de nuevos casos de prueba en diferentes dominios de conocimiento. La idea será focalizar el trabajo en todos los detalles importantes en lo referente a tiempos de generación, tiempos de despliegue, tamaño de los artefactos generados, así como detallar qué cambios se han hecho en cada modelo y cómo

se ven representados gráficamente. Además, será importante estudiar qué artefactos se generarán con cada una de las modificaciones en el modelo de entrada. Para ello, se hará necesario mejorar el prototipo desarrollado, ya que en la actualidad se encuentra en las primeras fases de desarrollo.

El futuro trabajo también pasará por seguir madurando la generación incremental de artefactos y madurar el prototipo que permite llevarla a cabo. Un aspecto interesante es la relación entre el diseño del DSL y los artefactos que se generan. Convendría buscar relaciones lo más directas posible para evitar dependencias que provoquen que una modificación en un elemento del modelo haga que se tengan que regenerar artefactos a partir de otro elemento del modelo que no ha sufrido ninguna modificación. Interesante también, es la aplicación del trabajo aquí propuesto en los entornos de desarrollo en los que se sincroniza la vista del modelo con la vista del código. Aunque muchas herramientas ofrecen soporte de sincronización, ella suele estar ligada a un entorno de desarrollo, a un metamodelo o a una arquitectura determinada, no empleándose una técnica genérica como la expuesta en este trabajo.

Bibliografía

ALANEN, M. & PORRES, I. (2003). Difference and union of models. In: UML 2003 The Unified Modeling Language, Modeling Languages and Applications: 6th International Conference, (22-24/10/2003), California (USA): Springer. Lecture Notes in Computer Science Vol. 2863, (Oct.), New York (NY, USA): Springer-Verlag, p. 2–17. ISBN: 978-3540202431

ALTMANNINGER, K.; KAPPEL, G.; KUSEL, A.; RETSCHITZEGGER, W.; SEIDL, M.; SCHWINGER, W. & WIMMER, M. (2008). AMOR - towards adaptable model versioning. In: MCCM 2008 and ACME/IEEE 11th MoDELS, 28/09-03/10/2008), Touluse (Francia): ACM/ IEEE. Proceedings of the 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM 2008) at the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS),

BENDIX, L. & EMANUELSSON, P. (2008). Diff and merge support for model based development. In: CVSM'08, (10-18/05/2008). Leipzig (Germany): ACM. Proceedings of the 2008 international workshop on Comparison and versioning of software models, New York (NY, USA): ACM. p. 31–34, ISBN: 978-1-60558-045-6.

BEYDEDA, S.; BOOK, M. & GRUHN, V. (Eds.) (2005). Model-Driven Software Development. New York (NY, USA): Springer. 414 p. ISBN 978-3-540-25613-7

BÉZIVIN, J. (2005). On the unification power of models. In: Software and System Modeling, Vol. 4, No. 2, (May.). New York (NY, USA): Springer. p. 171-188. ISSN: 1619-1366

BRUN, C. & PIERANTONIO A. (2008). Model differences in the eclipse modeling framework. In: UPGRADE: The European Journal for the Informatics Professional, Vol. 9, No. 2, Frankfurt (Germany): CEPIS, Council of European Professional Informatics Societies. p. 29–34. ISSN: 1684-5285

BUDINSKY, F.; BRODSKY, S. A. & MERKS, E. (2003). Eclipse Modeling Framework. Upper Saddle River (NJ, USA): Pearson Education. 704 p., ISBN: 978-0131425422

CICCHETTI, A.; RUSCIO, D. D. & PIERANTONIO, A. (2007). A metamodel independent approach to difference representation. In: Journal of Object Technology, No. 9, Vol. 6, Zurich (Switzerland): EtH Zurich. p. 165–185. ISSN: 1660-1769

CLEMENTS, P. & NORTHROP, L. (2002). Software Product Lines - Practice and Patterns. Upper Saddle River (NJ, USA): Addison-Wesley. ISBN: 978-0201703320

FRANKEL, D. S. (2003). Model Driven Architecture. Applying MDA to Enterprise Computing. Indiana (USA): John Wiley & Sons. ISBN: 978-0471319207

ECLIPSE FOUNDATION (2011). Xtext 2.1 Documentation [on line]. Ottawa (Ontario, Canada): Eclipse Foundation, Inc. <http://www.eclipse.org/Xtext/documentation/2_1_0/Xtext%202.1%20Documentation.pdf> [consult: 20/02/2012]

KARANAM, M. & AKEPOGU A.R. (2012). mROSE: To Determine Tool Selection and to Understand Model-Driven Software Evolution. In: International Journal of Computer Applications, Vol. 42, No. 1, (Mar.). New York (NY, USA): Foundation of Computer Science. p. 46-55. ISSN: 0975-8887

KENT, S. (2002). Model driven engineering. In: International Conference on Integrated Formal Methods '02, (15-18/05/2002), London (England): Springer. Lecture Notes in Computer Science 2335, p. 286–298. ISBN: 3-540-43703-7

KOLOVOS, D. S.; RUSCIO, D. Di; PIERANTONIO, A. & PAIGE, R. F. (2009). Different models for model matching: An analysis of approaches to support model differencing. In: Workshop on Comparison and Versioning of Software Models '09, (17/05/2009), Washington (EEUU): IEEE Computer Society. Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models, p. 1–6. ISBN: 978-1-4244-3714-6

KONEMANN, P. (2009). Model-independent differences. In: Workshop on Comparison and Versioning of Software Models '09, (17/05/2009), Washington (USA): IEEE Computer Society. Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models, p. 37–42. ISBN: 978-1-4244-3714-6

LIN, Y.; GRAY, J. & JOUAULT, F. (2007). DSMDIFF: A differentiation tool for domain-specific models. In: European Journal of Information Systems, Vol. 16, Birmingham (UK): The OR Society, p. 349–361. ISSN: 0960-085X

MAZANEK, S. & RUTETZKI, C. (2011). On the Importance of Model Comparison Tools for the Automatic Evaluation of the Correctness of Model Transformations. In: IWMCP' 2011. (20/06/2011), Zurich (Switzerland): Proceedings of the 2nd International Workshop on Model Comparison in Practice, p. 12-15. ISBN: 987-1-4503-0668-3

MENS, T.; WEMELINGER, M.; DUCASSE, S.; DEMEYER, S.; HIRSCHFELD, R. & JAZAYERI, M. (2005). Challenges in software evolution. In: IWPSE' 2005. (5-6/09/2005), Lisbon (Portugal): IEEE. Proceedings of the 8th International Workshop on Principles of Software Evolution, Washington (USA): IEEE Computer Society. p. 13–22. ISBN: 0-7695-2349-8

OHST, D.; WELLE, M. & KELTER, U. (2003). Differences between versions of UML diagrams. In: ESEC/FSE '03 (01-05/09/2003), Helsinki (Finland): ACM., Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, New York (USA): ACM. p. 227–236. ISBN: 1-58113-743-5

OLIVEIRA, H. L. R.; MURTA, L. G. P. & WERNER, C. (2005). Odyssey-VCS: a flexible version control system for UML model elements. In: SCM-12 (05-06/09/2005), Lisbon (Portugal): ACM, Proceedings of the 12th international workshop on Software configuration management, New York (USA): ACM, p. 1–16. ISBN: 1-59593-310-7

PALACIOS GONZÁLEZ, E.; FERNÁNDEZ FERNÁNDEZ, H.; GARCÍA DÍAZ, V.; G-BUSTELO, B. C. P.; LOVELLE, J. M. C. & MARTÍNEZ, O. S. (2008). General purpose MDE tools. In: International Journal of Artificial Intelligence and Interactive Multimedia. Vol. 1, No. 1, Madrid (Spain): Imal-Software, p. 72–75. ISSN: 1989-1660

READ, R. C. & CORNELL, D. G. (1977): The graph isomorphism disease. In: Journal of Graph Theory. Vol. 1, Hoboken (NJ, USA): Wiley Periodicals, Inc., p. 339–363. ISSN: 1097-0118

REITER, T.; ALTMANNINGER, K.; BERGMAYR, A. & KOTSIS, G (2007). Models in conflict - detection of semantic conflicts in model-based development. In: 3rd International Workshop on Model-Driven Enterprise Information Systems in conjunction with 9th International Conference on Enterprise Information Systems (MDEIS-2007), (05/2007), Madeira (Portugal): INSTICC Press. Proceedings of 3rd International Workshop on Model-Driven Enterprise Information Systems in conjunction with 9th International Conference on Enterprise Information Systems, ISBN: 978-989-8111-00-5

SEIBEL, A.; HEBIG, R. & GIESE, H. (2012).Traceability in Model-Driven Engineering: Efficient and Scalable Traceability Maintenance. In: CLELAND-HUANG, J.; GOTEL, O. & ZISMAN, A. (2012). Software and Systems Traceability, Londres (Inglaterra): Springer. p. 215-240. ISBN: 9781447122388

SEIDEWITZ, E. (2003). What models mean. In: IEEE Software, Vol. 20, No. 5, (Sep-Oct.) Washington (USA): IEEE Computer Society Offices. p. 26–32. ISSN: 0740-7459

TICHY, W. F. (1985). RCS - A system for version control. In: Software Practice and Experience, Vol. 15, No. 7, (Jul.). New York (NY, USA): John Wiley & Sons. p. 637–654. ISSN: 1097-024X

TOULMÉ, A. (2006). Presentation of EMF Compare utility. In: Eclipse Summit, (11-12/10/2006), Esslingen (Germany): Eclipse Foundation. Proceedings, Ottawa (Ontario, Canada): Eclipse Foundation

VÖLTER, M. (2003). A catalog of patterns for program generation [on line]. In: EuroPloP2003: Eighth European Conference on Pattern Languages of Programs, (04-08/07/2003), Irsee (Alemania): Hillside Europe e.V.,. Papers EuroPLoP™ 2003. Munich (Germany): Hillside Europe e.V., 34 p. IBSN: 978-3-87940-819-1 <http://hillside.net/europlop/europlop2003/papers/WorkshopB/B6_VoelterM.pdf> [consult: 15/01/2012]

VÖLTER, M. & STAHL, T. (2006). Model-Driven Software Development. Chichester (England): Technology, Engineering, Management: John Wiley & Sons. ISBN: 978-0470025703