Una valoración de la Modernización de Software Dirigida por Modelos

Óscar Sánchez Ramón, Francisco Javier Bermúdez Ruiz y Jesús García Molina

Departamento de Informática y Sistemas
Universidad de Murcia
{osanchez,fjavier,jmolina}@um.es

Resumen Las técnicas de la Ingeniería de Software Dirigida por Modelos son aplicables a la modernización de software. Los modelos son muy apropiados como representaciones abstractas de los sistemas a modernizar y las transformaciones de modelos automatizan las tareas de los procesos de modernización. A lo largo de la pasada década, estas técnicas se han aplicado en diversos escenarios de modernización, especialmente en migración de aplicaciones. Nuestro grupo de investigación ha participado en dos proyectos de migración con empresas, y como resultado de esta experiencia, en este trabajo presentamos una valoración de la modernización basada en modelos, destacando una serie de ventajas e inconvenientes.

1. Introducción

Los enfoques más extendidos de la Ingeniería del Software Dirigida por Modelos (*Model-Driven Engineering*, *MDE*[1]) están destinados al desarrollo de nuevas aplicaciones software [1] [2]. Sin embargo, las técnicas de este paradigma también son aplicables en la evolución del software [3] [4]. Desde los primeros años de la pasada década, en los que MDE emergió como una nueva disciplina de la ingeniería del software, la utilidad de los modelos y de las transformaciones de modelos en la ingeniería inversa y modernización de software fue evidente, como se señaló en [4] [5]. Los modelos son útiles como representaciones abstractas de diferentes aspectos de un sistema heredado y las trasformaciones de modelos permiten automatizar algunas tareas implicadas en la modernizacion de tales sistemas. A lo largo de estos años, las técnicas MDE han sido aplicadas en una variedad de escenarios de modernización, en especial en migración de aplicaciones [6] [7] y han aparecido algunas herramientas de modernización dirigida por modelos [8]. Otro esfuerzo destacable es la iniciativa ADM (*Architecture-Driven Modernization*) [3] destinada a ofrecer un conjunto de metamodelos estándares para representar información comúnmente manejada en modernización, que fue lanzada por OMG en 2003.

Nuestro grupo ha participado en dos proyectos con empresas relacionados con la migración de software. En el primero abordamos la generación automática de

[1] También conocido como *MDSE* (*Model-Driven Software Engineering*).

envoltorios (*wrappers*) Java para el acceso a aplicaciones PL/SQL heredadas y en el segundo, un proyecto mucho más ambicioso, nos enfrentamos a la modernización de aplicaciones Oracle Forms a la plataforma Java, en concreto a la migración de la interfaz de usuario y los datos. La ejecución de estos proyectos implicó la investigación en áreas como la inyección de modelos a partir de código fuente y la ingeniería inversa de código y datos.

A partir de la experiencia adquirida, en este artículo valoraremos la aplicación de las técnicas MDE en modernización de software (*Model-Driven Software Modernization*, MDSM). Conviene aclarar que no tratamos de describir nuestra experiencia en un caso de estudio particular, sino que presentamos algunas de las lecciones aprendidas en la práctica de MDSM en forma de beneficios e inconvenientes. Por otro lado, un proyecto de modernización, como es el caso de una migración de una aplicacion, es una actividad compleja que puede suponer un esfuerzo de decenas de programadores/año. Por ello, también aclaramos que no se abordará cómo el uso de modelos afecta a todos los aspectos del proceso de modernización (definición de la estrategia de migración, evaluación de coste, gestión de tareas, manejo de versiones, etc.), sino que nos centraremos principalmente en la definición de las arquitecturas basadas en modelos que automatizan la generación de artefactos de la aplicación destino.

El trabajo se ha organizado del siguiente modo. A continuación se introducen conceptos básicos relacionados con MDSM. En la Sección 3 se presenta la actividad realizada por nuestro grupo de investigación en MDSM, en especial los dos proyectos con empresas que nos han permitido experimentar con aplicaciones reales. Una vez explicada nuestra experiencia, en las Secciones 4 y 5 se plantean las ventajas e inconvenientes (respectivamente) que hemos detectado. En la Sección 6 se describen diversos esfuerzos realizados en el ámbito del MDSM. Finalmente se presentan las conclusiones y la bibliografía.

2. Conceptos básicos de modernización de software

En esta sección introduciremos terminología relacionada con la modernización de software y explicaremos el significado de aplicar técnicas MDE en tareas de migración de software.

A diferencia del mantenimiento, la **modernizacion** de software implica cambios importantes en la estructura y/o funcionalidad de una aplicación existente. Estos cambios son destinados a mantener el valor estratégico de la aplicación para la empresa. La **reingeniería** de software es un tipo de modernización destinada a transformar aplicaciones de una forma disciplinada con el propósito de mejorar su calidad [9]. Un proceso de reingeniería (y en general una modernización) se suele organizar en tres etapas. Primero se aplica **ingeniería inversa** para extraer una descripción lógica del sistema. A continuación se realiza la **reestructuración** del sistema a través de transformaciones de las descripciones lógicas obtenidas en la etapa anterior, y finalmente se ejecuta la etapa de **ingeniería directa** o **generación** del nuevo sistema a partir de las nuevas descripciones lógicas. Una **migración** es un caso especial de reingeniería en el que

un sistema software heredado se traslada a una tecnología más actual. Como sucedió en el caso de nuestro proyecto de migración de Oracle Forms a Java, una migración supone un cambio en la arquitectura del sistema.

Metamodelado y transformaciones de modelos son las dos técnicas básicas sobre las que se construyen las soluciones MDE. En el caso de la reingeniería, las etapas de ingeniería inversa, reestructuración y generación del nuevo sistema son organizadas como una cadena de transformaciones de modelos, como ilustra la Figura 1. Antes de aplicar la ingeniería inversa es necesaria una etapa previa para inyectar modelos (T2M) a partir de los artefactos a modernizar (por ejemplo, código fuente o un esquema relacional). Después de esta inyección, se aplica la ingeniería inversa como una cadena de transformaciones modelo a modelo (M2M). La siguiente etapa es la reestructuración, que implica otra cadena de transformaciones M2M, y finalmente se genera el nuevo sistema mediante una cadena que debe tener una transformación modelo a texto (M2T) final.

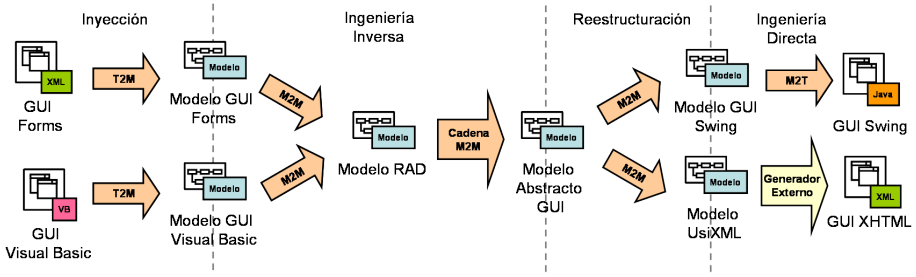

Figura 1. Proceso de migración de la GUI

3. Experiencia en MDSM del grupo Modelum

En el año 2007, como parte de un proyecto de transferencia de tecnología MDE a la empresa Sinergia Tecnológica S.L., se identificaron algunas tareas de desarrollo manuales que podían ser automatizadas, con el fin de ilustrar las ventajas de MDE. Se automatizó la generación de unos *wrappers* Java para el acceso a la lógica de negocio implementada como procedimientos PL/SQL almacenados en base de datos. La empresa disponía de reglas de codificación y guías para la construcción de los *wrappers* que permitieron su generación automática. La solución consistió en una cadena de transformaciones de modelos que generaba el código Java de los *wrappers* a partir del código PL/SQL. Esta cadena comenzaba con una inyección de modelos, seguía con una transformación M2M de los modelos PL/SQL de bajo nivel a unos modelos que conformaban a un metamodelo que representaba explícitamente las reglas y guías de la empresa, y finalmente se aplicaba una transformación M2T para generar el código Java de los *wrappers*. Este proyecto se describe con detalle en [10] y se utilizó en [11] para ilustrar la

utilidad de las transformaciones M2M intermedias frente a soluciones basadas sólo en una transformación M2T. Para inyectar los modelos se utilizó una herramienta de definición de Lenguajes Específicos del Dominio (*Domain Specific Languages*, *DSL*), en concreto Xtext [12]. Aunque el metamodelo era generado automáticamente, los modelos inyectados eran de baja calidad y fue necesario definir una transformación sobre un metamodelo de PL/SQL. Esta experiencia motivó la construcción del lenguaje Gra2MoL [13] para extraer modelos a partir de código fuente de lenguajes de programación, dado que herramientas como Xtext realmente no eran apropiadas para esta tarea al estar destinadas a la creación de DSLs.

Nuestro grupo también participó con la empresa Sinergia en un proyecto CDTI cuyo objetivo fue construir una herramienta que facilitase la migración semi-automatica de aplicaciones Oracle Forms a Java (proyecto *Forms2Java*). Se definió una arquitectura dirigida por modelos para automatizar la migración de la interfaz de usuario [14] y de la capa de persistencia [15]. Además, se generó un entorno de asistencia basado en tareas para la migración semi-automatica y manual de la lógica de negocio de la aplicación. Tanto para la migración de la GUI como de la persistencia, se definieron arquitecturas reutilizables que son independientes de la plataforma destino.

En el caso de la migración de la GUI, la solución ideada es aplicable a cualquier tecnología RAD (*Rapid Application Development*), no solo a Forms, gracias a la definicion de un metamodelo que representa los aspectos comunes a las GUI de aplicaciones RAD. El proceso de ingeniería inversa parte de ventanas con posiciones absolutas de los controles e infiere un layout de acuerdo a los tipos soportados en los frameworks GUI modernos (e.g. *GridLayout* or *FlowLayout*).

En el caso de la migración de los datos, la solución incluye una mejora de la calidad de la representación de los datos y la migración a un framework de persistencia Java. El proceso de ingeniería inversa analiza el estado de las restricciones entre tablas (claves ajenas) y dentro de una misma tabla (restricciones de chequeo sobre una columna), e infiere claves ajenas a partir de los registros de las tablas y del código de las aplicaciones cliente. La reestructuración verificaba la forma normal de los datos y aplicaba un proceso de normalización si era necesario.

Para inyectar modelos GUI se utilizó la herramienta disponible en Eclipse/EMF para convertir documentos XML[2] en modelos, y para inyectar los modelos de los esquemas relacionales se utilizó Gra2MoL, para lo que se definió una gramática de un subconjunto de SQL y un metamodelo de esquemas relacionales.

4. Ventajas

En la Sección 2 indicamos la utilidad de MDE en modernización, que consiste en representar el conocimiento extraído explícitamente en forma de modelos y

[2] Oracle dispone de una herramienta que convierte formularios Forms binarios en documentos XML.

la aplicación de transformaciones para reestructurar y generar un nuevo sistema. En esta sección comentaremos una serie de ventajas derivadas de esta idea fundamental.

V1. Productividad y rentabilidad.

Uno de los beneficios bien conocidos que se obtiene al aplicar las técnicas MDE es el incremento de la productividad en el desarrollo, como consecuencia de la automatización que se introduce en la generación de los artefactos del sistema final. El uso de MDE en proyectos de migración es más rentable cuanto mayor sea el número de artefactos cuya migración sea repetitiva siguiendo unas pautas bien definidas, y cuanto mayor sea el número de aplicaciones del mismo tipo a migrar.

En el proyecto de generación de *wrappers*, el retorno de inversión se producía con la creación de 105 *wrappers* [10]. En el proyecto *Forms2Java* se estimó un ahorro de esfuerzo del 60-85 % para la GUI, del 60-90 % para la persistencia y del 5 %-10 % para la lógica de negocio, según la información proporcionada por la empresa.

V2. Metamodelos, un formalismo unificador e integrador.

Diferentes formalismos han sido usados en la modernización de software para representar el conocimiento extraído en la ingeniería inversa. A principios de los noventa, XML surgió como tecnología para soportar la definición de cualquier metadato usado por una aplicación.

Sin embargo, los lenguajes de metamodelado como Ecore o MOF superan a XML por su mayor potencia expresiva y la existencia de potentes lenguajes de transformación de modelos para implementar las transformaciones requeridas en las etapas del proceso de reingeniería, sin olvidar la posibilidad de definir restricciones en los metamodelos y la libertad de definir cualquier tipo de notación para un modelo (principio de separación de las sintaxis abstracta y concreta).

V3. Independencia de la tecnología origen y destino.

Cuando se define un proceso de modernización es importante que sea independiente de las tecnologías origen y destino. Esto se consigue definiendo una representación que desacople las tecnologías origen y destino del resto del proceso. En MDSM, los modelos actuan como elementos de desacople para origen y destino. La independencia se consigue con inyectores y transformaciones M2M que generan los modelos independientes del origen, y mediante transformaciones M2M y M2T a partir de los modelos independientes, para obtener el destino.

Por ejemplo, cuando abordamos la migración de la GUI de aplicaciones RAD (véase la Figura 1) se definió un metamodelo capaz de representar la GUI de cualquier entorno RAD, que nos proporciona independencia del origen al resto del proceso. También se definió un metamodelo abstracto de la GUI para representar la GUI de cualquier framework, que es por tanto independiente de la tecnología destino. Se puede observar que la cadena de transformación entre

el metamodelo RAD y el metamodelo abstracto de la GUI es reutilizable para cualquier par de tecnologías origen/destino. Buena parte del valor de una solución MDSM reside en la definición de estos metamodelos, lo que requiere gran capacidad de abstracción y buen conocimiento de las diferentes tecnologías.

V4. Extensibilidad.

Un proceso de modernización puede ser extendido con nuevas etapas o incluso ser modificado con etapas que reemplazan a las existentes. En una cadena de transformación, los modelos constituyen un punto de extensión válido para incorporar nuevas etapas. De hecho, ha sido el mecanismo utilizado para conseguir la independencia del origen y destino comentada anteriormente. Para extender una cadena con una nueva etapa basta con incluir una cadena de transformación que tenga como entrada el modelo que actúa como punto de extensión y como resultado un modelo que pueda ser incorporado al proceso de modernización en cualquier punto, conformando al correspondiente metamodelo. Nótese que podría ser incorporado en el mismo punto de extensión para lo que debería conformar con el metamodelo asociado a dicho punto.

Durante la modernización de datos en *Forms2Java*, un modelo de datos es obtenido de los scripts DDL. Luego este modelo es reestructurado para la mejora de la calidad del esquema, y a partir del modelo obtenido se genera el esquema mejorado. Esta cadena tuvo que ser extendida para incluir una normalización del modelo de datos en la etapa de reestructuración, como muestra la Figura 2.

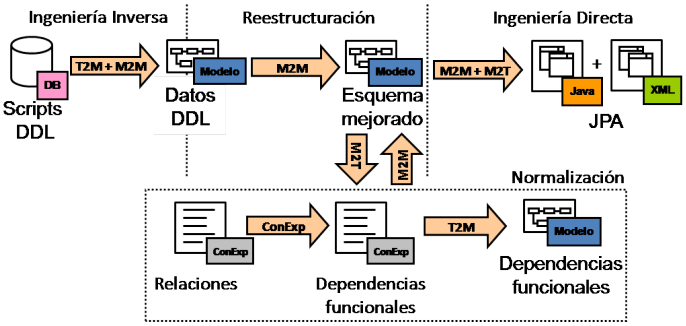

Figura 2. Normalización en la migración de datos

V5. Reutilización.

Los fragmentos de las cadenas de transformación pueden reutilizarse definiendo metamodelos que desacoplen unas partes de otras, y definiendo las transformaciones hacia y desde esos metamodelos. En el caso de la independencia del origen y destino antes mencionada (V3) ya observamos que se consigue la reutilización de cadenas de transformación usando modelos para el desacople. Este

mecanismo se puede utilizar para reutilizar metamodelos y transformaciones de terceros, mediante transformaciones M2M que conviertan nuestros modelos en modelos conformes al metamodelo de terceros.

Por ejemplo, como se muestra en la fase de reestructuración de la Figura 1, hemos implementado una transformación entre nuestro modelo abstracto de GUI y UsiXML que es un lenguaje de definición de interfaces (UIDL) definido por un metamodelo. Así podemos aprovechar las herramientas de generación de código disponibles para UsiXML, como por ejemplo el generador para XHTML.

V6. Interoperabilidad con aplicaciones de terceros.

Los modelos constituyen un punto de extensión (como se indicó en V4) que facilita la integración con aplicaciones de terceros, aunque sean cerradas y no estén implementadas con un enfoque MDE. Una aplicación de terceros que requiera una entrada serializada y genere una salida serializada puede ser integrada en una cadena de transformación. Habría que seleccionar dos puntos de extensión (modelos), uno para la entrada y otro para la salida de datos de la aplicación de terceros. La incompatibilidad para el punto de entrada se soluciona con una transformación M2T y la del punto de salida con una transformación T2M.

Un ejemplo de este tipo de interoperabilidad lo hemos aplicado al integrar la herramienta *Concept Explorer* (*ConExp* [16]) en la identificación de dependencias funcionales durante la etapa de normalización de datos. Esta herramienta utiliza para la entrada y salida un formato propietario. Se crea una transformación M2T para transformar el modelo que representa un esquema relacional en el formato de entrada propietario de *ConExp*, y se crea una transformación T2M para inyectar las dependencias funcionales generadas por esta herramienta en un modelo de dependencias funcionales que se incorpora al proceso de modernización (Figura 2).

V7. Fácil obtención de modelos a partir de artefactos software.

Un proceso de modernización requiere la construcción de analizadores sintácticos (*parsers*) que analicen los artefactos de la aplicación existente (por ejemplo, código fuente o esquemas relacionales). En el caso de una modernización dirigida por modelos, la existencia de DSLs para la inyección de modelos, como Gra2MoL y Schemol [17], y de frameworks como Modisco [18] facilita considerablemente el trabajo ya que evita la creación de tales analizadores manualmente. Otra ventaja con respecto a la reingeniería de datos, es que los inyectores implementados para datos relacionales usando Gra2Mol son reutilizables en la modernización de cualquier sistema legado con origenes de datos relacionales.

En el proyecto *Forms2Java*, Gra2MoL resultó muy útil para extraer los modelos del esquema legado y los datos contenidos en dicho esquema. En este proyecto también se usó el inyector XML de Eclipse para extraer automáticamente modelos desde los formularios expresados en XML.

V8. Integración de espacios tecnológicos.

Un espacio tecnológico es un contexto de trabajo que tiene asociado un conjunto de conceptos, métodos, técnicas y herramientas [19] y que se construye en torno a un par de conceptos básicos que se relacionan a través de una relación *instancia-de*. Las gramáticas *grammarware* (par gramática/programa) y los modelos *modelware* (par metamodelo/modelo) son dos ejemplos de espacios tecnológicos. En la modernización de software frecuentemente están involucrados diferentes espacios tecnológicos que es necesario integrar. Los inyectores de modelos permiten transformar artefactos de cualquier espacio tecnológico en modelos, aplicar transformaciones M2M y entonces generar artefactos del mismo o diferente espacio tecnológico.

En nuestros proyectos nos hemos encontrado con la necesidad de integrar en el *modelware* artefactos de diferentes espacios tecnológicos: gramáticas, código fuente, datos, scripts DDL y DML, documentos XML, páginas HTML, etc.

V9. Decoración de modelos y trazabilidad.

En cualquier proceso, con frecuencia, la información representada tiene cierta variabilidad que da lugar a una representación compleja si se tiene en cuenta todo a la vez. Desde el enfoque MDE, es posible definir un metamodelo para los aspectos comunes y separar en diferentes metamodelos, normalmente pequeños, los aspectos variables. De este modo, un modelo puede ser anotado o decorado (nótese la semejanza con el patrón *decorador*) con otros modelos para añadirle cierta información variable.

Durante el proceso de reingeniería se aplican alteraciones al sistema original (e.g. abstracciones o refactorización) que frecuentemente suelen omitir detalles del sistema original. Además, se pierde la relación (traza) entre los elementos creados en un modelo a partir de los elementos de otros. En ocasiones es necesario mantener esta información de trazabilidad para utilizarla en posteriores transformaciones. El mecanismo de decoración puede ser empleado para mantener la *trazabilidad* del proceso de modernización por medio de una solución no intrusiva, que son los modelos de trazas.

La Figura 3 muestra dos ejemplos de decoración en la migración del código de los eventos en el proyecto *Forms2Java*. El código original PL/SQL mezcla lógica de negocio con código relativo a la presentación. Por tanto, se creó un modelo que decoraba el modelo PL/SQL original para indicar qué partes del código corresponden a cada capa y así lograr la separación de aspectos. Para no perder los detalles del código original, la transformación M2M genera un modelo de trazas entre el modelo destino Java y el modelo original PL/SQL.

5. Limitaciones

En esta sección comentaremos las limitaciones que hemos encontrado en nuestra práctica de MDSM, algunas de las cuales son comunes a MDE.

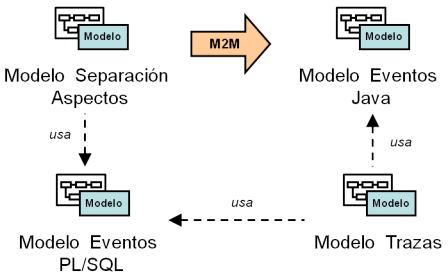

Figura 3. Mecanismo de decoración

D1. Insuficiencia de detalle de KDM.

KDM (*Knowledge Discovery Metamodel*) es el metamodelo central de ADM el cual está destinado a representar los artefactos software (código y datos) de una aplicación de forma independiente a las plataformas usadas (lenguajes, bases de datos, etc.), y que establece varios niveles de abstracción. Fue ideado principalmente para soportar la interoperabilidad entre herramientas de modernización. En [20] ya presentamos un análisis de ADM, en especial de KDM. En algunos aspectos, como las interfaces de usuario, KDM ofrece un nivel de detalle reducido y no permite considerar conceptos comunes a numerosas tecnologías y es necesario extender el metamodelo (dejando de ser estándar) o bien usar estereotipos (mecanismo menos invasivo, pero que dificulta la interoperabilidad).

En el caso de *Forms2Java*, no fue posible representar directamente algo tan común como una casilla de verificación (*checkbox*). Hay que definir un estereotipo *'Checkbox'* que se aplicaría a la metaclase *UIField* y que contendría un valor etiquetado *'checked'*. KDM permite una representación detallada de código y datos y sin embargo ofrece escasa expresividad para representar GUIs. Detrás de KDM subyace la antigua idea de un lenguaje universal (UNCOL) que, como es bien sabido, nunca llegó a hacerse realidad.

D2. Dificultad para trabajar con modelos de gran escala.

En el escenario de un proceso de modernización, con frecuencia se requiere manipular modelos de gran escala (para código y datos). Dado que XMI mostró importantes limitaciones para trabajar con modelos de gran escala, han aparecido herramientas basadas en el uso de bases de datos relacionales como CDO [21]. Pero como la naturaleza del modelo relacional se ha demostrado inadecuada para tratar el problema [22], han surgido nuevas propuestas de repositorios de modelos, como Morsa ([23]) que emplean bases de datos basadas en documentos y otros prototipos basados en grafos [22]. Estas alternativas hacen viable el manejo de modelos de gran escala, a diferencia de XMI o CDO, pero siguen adoleciendo de características básicas para un uso productivo en un contexto industrial (e.g. carencia de un lenguaje para consultas complejas, de transacciones ACID, de soporte para la concurrencia, etc.)

En el proyecto *Forms2Java*, durante la etapa de ingeniería inversa de datos, se requiere persistir los datos almacenados en el sistema legado sobre modelos, para inferir (mediante consultas sobre dichos modelos) las claves ajenas no declaras en el origen de datos.

D3. Carencia de inyectores para lenguajes de programación.

La utilización de lenguajes como Gra2MoL o la creación de discoverers en Modisco puede suponer un esfuerzo considerable. En el caso de Gra2MoL son necesarios una gramática ANTLR del lenguaje, el metamodelo destino y escribir la transformación. El framework MoDisco ofrece varios discoverers que generan un modelo KDM a partir de Java, JSP y XML entre otros, pero para otros lenguajes es necesario implementar el discoverer, lo cual no es sencillo. Ademas, si el proyecto no usara KDM sería necesario escribir una transformación M2M desde KDM al metamodelo deseado.

Existen sitios web[3] que ofrecen numerosos metamodelos para lenguajes de programación ampliamente extendidos como PL/1 y Cobol (al igual que existen *zoos* de gramáticas), pero no existen inyectores disponibles de manera abierta y gratuita para estos lenguajes, por lo que suele ser necesario implementarlos. La existencia de un repositorio de inyectores para estos lenguajes facilitaría el uso de MDE.

En el proyecto *Forms2Java* nos encontramos con la necesidad de inyectar modelos a partir de código PL/SQL. Dado que no había disponible un inyector, utilizamos Gra2MoL para crear uno, lo que supuso crear también el metamodelo PL/SQL.

D4. Carencia de repositorios de transformaciones directas.

En ocasiones es conveniente trasladar una representación del sistema para que conforme con otro metamodelo que nos ofrece ciertas ventajas o facilidades. Por ejemplo, para beneficiarse de las facilidades disponibles para ontologías (como los razonadores) se hacen necesarias conversiones entre formalismos como, por ejemplo, de modelos Ecore a ontologías OWL. Estas transformaciones son en gran medida correspondencias directas que no implican decisiones de diseño por parte del usuario. Sería beneficioso disponer de un repositorio con estas transformaciones para metamodelos estandarizados (como puede ser KDM).

En el proyecto *Forms2Java* queríamos detectar dependencias entre los controles de la GUI. El uso de la representación de KDM era más adecuado que analizar el modelo PL/SQL. Esto requería implementar una transformación de PL/SQL a KDM que, en cierto modo, es una simple correspondencia entre lenguajes.

D5. Inexistencia de soluciones predefinidas para ingeniería inversa

Cuando se aborda la migración de un sistema heredado, en ocasiones aparecen una serie de problemas de refactorización y marcado de código que son comunes

[3] Por ejemplo, http://www.emn.fr/z-info/atlanmod/index.php/Zoos

a los primeros lenguajes imperativos como Cobol, Basic o Fortran (por ejemplo, la identificación de objetos o la eliminación de saltos incondicionales (GOTOs)). Además, durante el análisis estático de código, se realizan ciertas tareas previas como la eliminación de código muerto o la identificación de bloques básicos. Dichas tareas son relativamente generales y pueden ser aplicadas a un gran número de proyectos de migración. Por lo tanto, sería interesante disponer de soluciones predefinidas, esto es, transformaciones que se apoyen en metamodelos de código estándar (como por ejemplo KDM).

En el proyecto *Forms2Java*, surgió la necesidad de fragmentar el código PL/SQL en bloques básicos para separar la lógica de negocio del código de acceso a datos. Fue necesario implementar un algoritmo de detección de bloques básicos sobre un metamodelo de código independiente.

D6. Carencia de estructuras de datos complejas en lenguajes de transformación.

Frecuentemente cuando se realiza reingeniería del software es necesario tratar con algoritmos relativamente complejos que requieren de estructuras de datos también complejas. Se necesita de un lenguaje de transformación de modelos capaz de definir estructuras de datos complejas y/o que proporcione tipos de datos complejos (listas, tablas, etc.). Por otra parte, estos lenguajes deberían ser capaces de expresar algoritmos de cualquier complejidad. No todos los lenguajes de transformación son, por tanto, apropiados para la reingeniería. Por ejemplo, en la migración de interfaces de usuario RAD, se optó por representar las relaciones espaciales entre controles como un grafo dirigido atribuido. Se comenzó la transformación en ATL y fue descartado por no soportar el tipo grafo ni su definición. Se optó por usar EMF dinámico directamente desde código Java.

En ocasiones es conveniente recurrir a otro tipo de lenguajes, como lenguajes de transformación de modelos basados en grafos o de transformación de programas, pues la solución se puede expresar a un nivel de abstracción más alto que una implementación de un lenguaje imperativo.

D7. Inmadurez de los entornos de desarrollo MDE.

Los entornos de desarrollo MDE actuales son menos productivos y eficientes que otros entornos más maduros como los existentes para los lenguajes de programación, siendo necesario una mayor robustez y grado de integración entre las herramientas que incluyen. Por ejemplo, un aspecto que ha limitado nuestra productividad ha sido la depuración de las transformaciones M2M que, independientemente del lenguaje y el entorno usado, ha resultado ser una tarea complicada pues los errores se suelen envolver en mensajes poco descriptivos. Además, los entornos MDE actuales carecen de facilidades maduras de depuración de transformaciones, como depuración interactiva o visualización de información de alto nivel más allá de las variables. Aunque existen trabajos de investigación avanzados como [24], todavía no se han incorporado a los entornos de desarrollo. También es necesario que se integren los esfuerzos que se están realizando en otras áreas, como las pruebas de transformaciones de modelos [25].

6. Trabajo relacionado

En esta sección comentaremos algunos de los principales esfuerzos de MDSM a lo largo de los diez años transcurridos desde la propuesta de ADM, hito que se puede considerar como el arranque de este enfoque.

Los fundamentos de la modernización de software dirigida por modelos fueron expuestos por Jean Marie Favre en [4]. Un año antes, en 2003, ADM emerge como una iniciativa para impulsar la aplicación de técnicas MDE en escenarios de modernización de software (por ejemplo, migración de aplicaciones, migración a MDA, mejora de la calidad de los datos, conversión de lenguajes, etc.). No se trata de ningún enfoque sobre modernización sino una propuesta de definición de metamodelos estándares para representar información que normalmente se maneja en la modernización de software. El metamodelo KDM, que es el elemento central de ADM, se utiliza para representar código a diferentes niveles de abstracción, desde instrucciones de bajo nivel a reglas de negocio. Además de KDM, actualmente ADM ha publicado otros tres metamodelos que son: SMM (*Software Metrics Metamodel*) para definir métricas de código, ASTM (*Abstract Syntax Tree Metamodel*) para representar código de cualquier lenguaje como un árbol de sintaxis abstracta, y SACM (*Structured Assurance Case Metamodel*) para la discusión y valoración con los *stakeholders*. El objetivo de ADM con sus metamodelos estándar es favorecer la interoperabilidad entre herramientas de modernización, facilitar la integración de herramientas de ingeniería directa e inversa, y reducir el tiempo, coste y riesgo de los proyectos a través de la automatización.

Entre las herramientas de modernización basadas en modelos cabe destacar MoDisco [18]. Se trata de un framework extensible para desarrollar herramientas basadas en modelos para asistir en la modernización de sistemas existentes. Con este objetivo MoDisco proporciona: metamodelos para describir sistemas existentes (KDM, ASTM, SMM), descubridores (*discoverers*) para crear modelos a partir de esos sistemas de forma automática (para Java, JSP y XML entre otros), herramientas genéricas para comprender y transformar modelos complejos, y casos de uso que ilustran como el framework puede asistir en procesos de modernización.

Hasta la fecha se han publicado pocos trabajos sobre experiencias industriales de modernización dirigida por modelos. En [6] se presenta un estudio de la factibilidad de la reingeniería de sistemas hacia MDA. Se incluye un caso de estudio de un sistema PL/SQL utilizado en el dominio de las aseguradoras. Concluyen con las lecciones aprendidas a modo de consejos para abordar una reingeniería utilizando MDA. En [7], los autores defienden el uso de MDE como una aproximación eficiente, flexible y fiable para procesos de migración, así como rentable comparado con una migración manual. El trabajo se ilustra con una migración de un sistema bancario de gran escala a J2EE. Se compara el enfoque con modelos con un proceso manual y se plantean algunos beneficios y limitaciones.

En cuanto a proyectos europeos relacionados con MDSM, podemos destacar tres. El proyecto MOMOCS (6º Programa Marco UE) [26] cuyo objetivo era definir una metodología basada en modelos para reingeniería de software y crear

las herramientas de soporte. De acuerdo a los resultados presentados, el impacto de este proyecto ha sido muy reducido. Ahora está en marcha el proyecto REMICS (7º Programa Marco UE) [27] cuyo objetivo es definir una metodología basada en modelos para la migración de sistemas heredados a servicios en la nube. Por último, Dynamod [28] es un proyecto de un consorcio de empresas y universidades alemanas para combinar técnicas de análisis estático y dinámico para definir un enfoque de ingeniería inversa dirigida por modelos.

7. Conclusiones

Finalizaremos este artículo con un resumen de las lecciones aprendidas. En cuanto a los beneficios hay que destacar el aumento de la productividad. Además los metamodelos ofrecen una mayor potencia expresiva para definir los metadatos que los formalismo tradicionales y los modelos facilitan la integración de soluciones y herramientas. En cuanto a la reutilización de las soluciones hay que subrayar la necesidad de definir metamodelos que independicen de las plataformas origen y destino.

La existencia de metamodelos estándares para modernización supone un importante ahorro de esfuerzo y facilita la interoperabilidad de herramientas. Sin embargo los metamodelos KDM y ASTM son demasiado generales y normalmente no se adaptan a las necesidades del proyecto. Además, KDM es un metamodelo complejo y no existe buena documentación con ejemplos que ilustren su uso. Otras dificultades son que ASTM requiere una adaptación dependiendo del lenguaje de programación cuyo código se quiere modernizar y que la integración de ASTM con KDM no es sencilla. También es preciso señalar que las empresas que ofrecen soluciones de modernización se resisten a adoptar KDM y los enfoques basados en modelos.

Las herramientas MDE todavía no proporcionan la funcionalidad deseada y entre las limitaciones destacamos la falta de escalabilidad para manejar grandes modelos, la falta de depuradores apropiados, y las carencias de los lenguajes de transformación para definir estructuras complejas. También son necesarias herramientas que soporten tareas comunes a proyectos de modernización: inyectores para lenguajes comunes, transformaciones genéricas para algunas operaciones de ingeniería inversa y repositorios de transformaciones simples y directas.

En cuanto al futuro de MDSM, una mayor aceptación requerirá de trabajos de investigación que muestren claramente sus ventajas en áreas tan maduras como la reingeniería de código y datos. En este sentido, son muy interesantes proyectos como Dynamod [28] que aplica un enfoque dirigido por modelos para combinar análisis estático y dinámico. Por otro lado, también son necesarios más proyectos industriales que apliquen una MDSM, y sobre todo que se documenten este tipo de experiencias y muestren datos sobre beneficios en productividad. En cuanto a ADM, su futuro no está claro después de 10 años y cabe preguntarse si no se ha tratado de un nuevo UNCOL.

Referencias

1. OMG: MDA Guide Version 1.0.1, http://www.omg.org/mda. (2003)
2. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling. Wiley (2008)
3. OMG: Architecture-Driven Modernization (ADM). http://adm.omg.org/ (2007)
4. Favre, J.M.: Foundations of model (driven) (reverse) engineering: Models. In: Dagstuhl Seminar Proceedings. (2004)
5. Bezivin, J., Others: On the applicability scope of model driven engineering. In: Proceedings of the MOMPES '07, IEEE Computer Society (2007) 3–7
6. Reus, T., et al.: Harvesting software systems for mda-based reengineering. In: Proceedings of the ECMDA-FA'06
7. Fleurey, F., et al.: Model-driven engineering for software migration in a large industrial context. In: Proceedings of the MoDELS'07
8. Perez-Castillo, R., de Guzman, I.G.R., Piattini, M., Ebert, C.: Reengineering technologies. IEEE Software **28**(6) (2011) 13–17
9. Tilley, S.R., Smith, D.B.: Perspectives on legacy system reengineering. Technical report, Software Engineering Institute, Carnegie Mellon University (1995)
10. Sánchez Cuadrado, J., et al.: Applying model-driven engineering in small software enterprises. Science Computer Programming (pendiente de publicación) (2013)
11. Cánovas Izquierdo, J.L., Sánchez Ramón, Ó., et al.: Utilidad de las transformaciones modelo-modelo en la generación automática de código. JISBD '07
12. Eclipse-Project: Xtext user guide. http://eclipse.org/Xtext (April 2008)
13. Canovas Izquierdo, J.L., Garcia Molina, J.: Extracting Models from Source Code in Software Modernization. Software & Systems Modeling (2012)
14. Sánchez Ramón, O., et al.: Model-driven reverse engineering of legacy graphical user interfaces. In: Proceedings of the ASE'10, ACM (2010) 147–150
15. Bermúdez Ruiz, F.J., García Molina, J.: Un framework basado en modelos para la modernización de datos. In: Actas de la JISBD'12
16. Yevtushenko, S.: System of data analysis concept explorer. In: Proceedings of the 7th National Conference on Artifical Intelligence KII-2000. (2000) 127–134
17. Cánovas Izquierdo, J.L., et al.: Schemol: Un lenguaje especifico del dominio para extraer modelos de bases de datos relacionales. In: Póster en JISBD'11
18. Bruneliere, H., et al.: MoDisco: a generic and extensible framework for model driven reverse engineering. In: Proceedings of the ASE'10
19. Kurtev, I., Bézivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: CoopIS, DOA'2002 Federated Conferences, Industrial track. (2002)
20. García Molina, J., Cánovas Izquierdo, J.L.: Una aplicación práctica de architecture-driven modernization (adm). In: JISBD. (2010) 339
21. Eclipse: Cdo model repository. http://www.eclipse.org/cdo/ (April 2013)
22. Barmpis, K., Kolovos, D.S.: Comparative analysis of data persistence technologies for large-scale models. In: Proceedings of the MoDELS'12, ACM (2012)
23. Espinazo Pagán, J., Sánchez Cuadrado, J., García Molina, J.: Morsa: A scalable approach for persisting and accessing large models. In: MoDELS. (2011) 77–92
24. Schoenboeck, J., et al.: Catch me if you can: debugging support for model transformations. In: Proceedings of the MoDELS'09, Springer-Verlag (2010) 5–20
25. Vallecillo, A., et al.: Formal specification and testing of model transformations. In: Proceedings of the SFM'12, Springer-Verlag (2012) 399–437
26. MOMOCS. http://www.momocs.org/ (2008)
27. Mohagheghi, P., et al.: REMICS- REuse and Migration of Legacy Applications to Interoperable Cloud Services - REMICS Consortium (2010)
28. van Hoorn, A., et al.: DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization (2011)