Implicaciones de transformaciones oblicuas en el desarrollo de un framework generador de aplicaciones orientadas a aspectos

Antonia Mª Reina Quintero
Dpto. de Lenguajes y Sistemas informáticos
Universidad de Sevilla
Avda. Reina Mercedes, s/n
41012 Sevilla
reinaqu@lsi.us.es

Jesús Torres Valderrama
Dpto. de Lenguajes y Sistemas informáticos
Universidad de Sevilla
Avda. Reina Mercedes, s/n
41012 Sevilla
jtorres@lsi.us.es

Resumen

La orientación a aspectos mejora la evolución del software localizando los cambios, mientras que el desarrollo de software conducido por modelos (MDD) basado en el estándar conocido como Model Driven Architecture (MDA) mejora la adaptación a los cambios tecnológicos, y por tanto, la evolución del software. En este artículo se propone un enfoque híbrido, de tal forma que podamos sacar partido de las ventajas proporcionadas por ambas propuestas. En MDA se convierten en elemento clave las transformaciones. El artículo se centra, por tanto, en el análisis del tipo de transformaciones necesarias para llevar a cabo este enfoque híbrido, y en cómo la tipología de las mismas afecta al transformador de modelos de un framework que genere aplicaciones en base a este enfoque híbrido.

1. Introducción

Debido a los constantes cambios que tienen que enfrentar los ingenieros software en los últimos años han surgido propuestas tanto para conseguir una mejor evolución de los sistemas software, como para poder adoptar los cambios de tecnología con mínimo coste. En la primera línea, se encuentra lo que se conoce como el nuevo paradigma de desarrollo software orientado a aspectos (DSOA)[4]. En la segunda se puede incluir el desarrollo conducido por modelos (MDD), sobre todo con la propuesta MDA[5] de la Object Management Group (OMG).

Puesto que ambas líneas de trabajo no son incompatibles, sino más bien, todo lo contrario, es factible pensar que aunando ambos paradigmas podemos conseguir aplicaciones que se pueden adaptar más fácilmente a los cambios de requisitos y a los cambios de tecnología. Así, nuestra propuesta se puede considerar como un enfoque híbrido, en el sentido de que proponemos trabajar con conceptos separados a cualquier nivel modelado, entendiendo por niveles de modelo los niveles propuestos por MDA: modelos independientes de la plataforma, modelos dependientes de plataforma y código).

En [6] se detallaba la arquitectura de un *framework* generador de aplicaciones, basado en MDA, que daba la posibilidad de modelar conceptos por separado. Estos aspectos se especificaban mediante lenguajes de modelado específicos de dominio, convirtiendo así las transformaciones en elementos clave, ya que pueden llegar incluso a realizar funciones de *weaver*. Desde esta perspectiva, componer un aspecto y un modelo que especifique la funcionalidad básica, se convertiría en la aplicación de una serie de transformaciones, a dos modelos, o a dos metamodelos.

En este artículo se estudian los distintos tipos de transformaciones existentes, para ver con cuál de ellas trabajará nuestro *framework*.

Además, se analizan las implicaciones de las mismas sobre el diseño de un componente del *framework*, el transformador de modelos. Para ello, el trabajo se estructura como sigue:

En la sección 2 se presenta una categorización del tipo de transformaciones que nos podemos encontrar en la bibliografía, para posteriormente determinar con cuáles de ellas debe trabajar nuestro framework generador de aplicaciones. El tipo de transformaciones será determinante para definir los requisitos del framework, sobre todo del transformador de modelos.

A continuación, en la sección *Estructura de aplicaciones generadas por el framework*, se hace una revisión de la estructura de las aplicaciones con las que va a trabajar el framework, con objeto de determinar los requisitos que imponen al mismo.

En la sección sección 4 se definen las características del transformador de modelos en base al tipo de transformaciones que vamos a contemplar en el *framework*.

Luego, en la sección 5, se exponen las opciones de implementación de las reglas de trasformación y motiva por qué se elige SQL embebido para implementarlas. Finalmente, se concluye el trabajo y se apuntan futuras líneas de investigación.

2. Categorizando transformaciones

Según la OMG [5], la transformación de modelos es el proceso de convertir un modelo en otro. Según France y Bieman [2], las transformaciones se pueden clasificar en transformaciones verticales y transformaciones horizontales. Las transformaciones verticales son aquellas que se aplican a un modelo para obtener otro expresado con un nivel de abstracción diferente, mientras que las transformaciones horizontales se aplican a un modelo para obtener otro del mismo nivel de abstracción. Un ejemplo de transformaciones verticales en el ámbito de MDA son las transformaciones necesarias para transformar un Modelo Independiente de Plataforma (PIM) en un Modelo Específico de Plataforma (PSM), mientras que una transformación horizontal, puede ser la composición de un aspecto en un modelo sin cambiar el nivel de abstracción.

Czarnecki [1], sin embargo, distingue entre tres tipos de transformaciones: verticales, horizontales y oblicuas. De forma que las transformaciones verticales involucran transformar un modelo de alto nivel en un modelo de más bajo nivel, llamándolas también refinamientos hacia adelante. Las transformaciones horizontales modifican la estructura modular de una aplicación al mismo nivel. Mientras que las transformaciones oblicuas tienen componentes tanto verticales como horizontales. En la 1 se muestra un esquema de los tipos de transformaciones que considera Czarnecki.

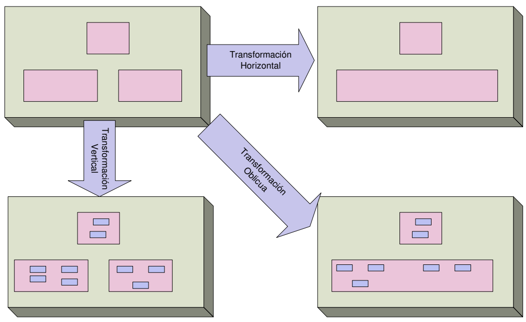

Figura 1: Transformaciones tomadas de [1]

En [3], además se identifican los distintos tipos de transformaciones verticales y horizontales que podemos aplicar a los modelos. Así, nos podemos encontrar con los siguientes tipos de refinamientos:

- Especialización.

  Este tipo de transformación toma como entrada un objeto o conjunto de objetos menos especializados y da como resultado un conjunto de objetos más especializados.

- Elaboración.

  Toma como entrada una configuración de objetos menos detallada y da como resultado configuraciones de objetos más detalladas.

- Realización.

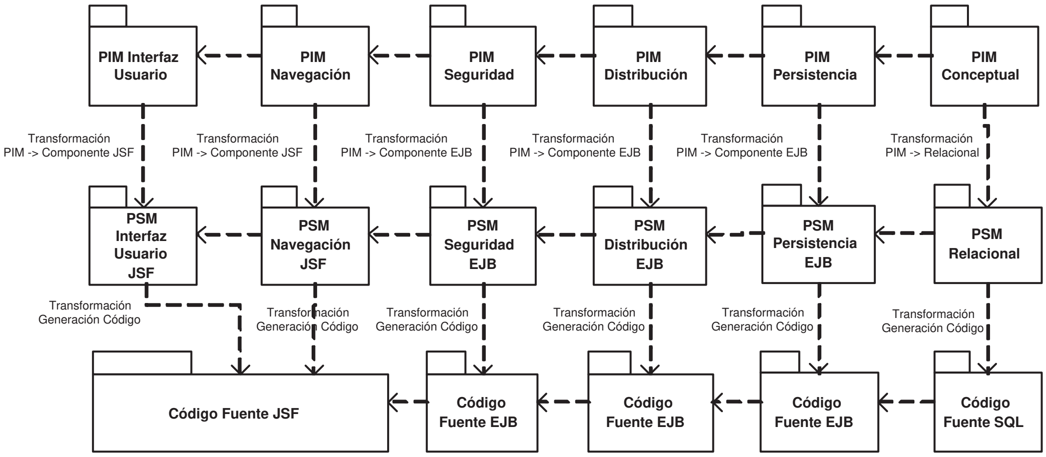

Figura 2: Estructura de una aplicación generada por el framework

Las configuraciones de objetos de la entrada se transforman en objetos que representan su implementación.

- Derivación.

  Nuevas especificaciones de objetos se derivan de las configuraciones de objetos que se toman como entrada.

- Descomposición.

  Objetos individuales se transforman en configuraciones de objetos en la salida.

Mientras que con respecto a las transformaciones horizontales nos podemos encontrar los siguientes tipos:

- Refactorización.

  Son transformaciones realizadas para mejorar la estructura de la especificación dada por los desarrolladores.

- Optimización.

  Su objetivo es mejorar algunas características de una especificación, tales como su rendimiento o el uso de los recursos.

- Deslocalización.

  Es un tipo de transformación que se utiliza para implementar aspectos, es decir, para encapsular elementos que cortan la estructura o el comportamiento de otros elementos, lo que en la terminología de orientación a aspectos se conoce como crosscutting concerns.

3. Estructura de aplicaciones generadas por el framework

En este apartado, analizaremos de qué tipo de aplicaciones se tiene que ocupar el *framework*, para posteriormente, comprobar qué implicaciones tendrán estas decisiones en el diseño del transformador de modelos.

Como ya se ha comentado en la introducción, algunas transformaciones servirán como mecanismo de composición. Supongamos que tenemos separados los distintos conceptos de nuestro sistema a nivel de modelo independiente de la plataforma, pero la plataforma específica en la que queremos implementar nuestro sistema no proporciona mecanismos de separación. En este caso, o bien a la hora de generar los modelos específicos de plataforma, o bien a la hora de generar el código, tendríamos que componer esos modelos separados.

Para clarificar un poco más este concepto, véase el ejemplo de la Figura 2. En él se mues-

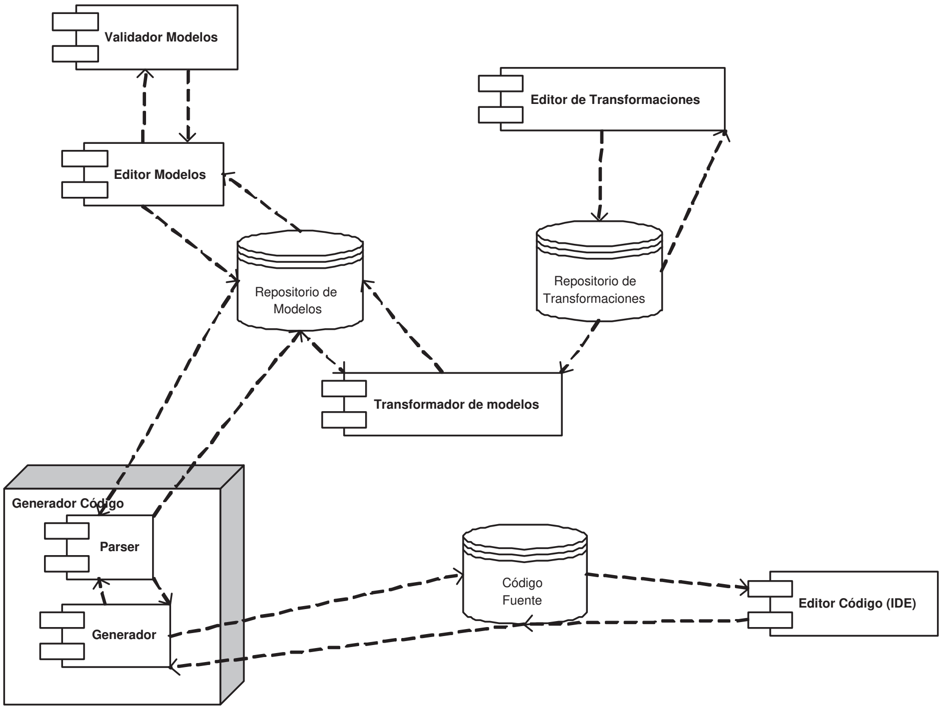

Figura 3: Componentes del framework generador de aplicaciones

tra la estructura que tendrá una aplicación generada por el framework. Cuenta con tres niveles de modelos (PIM, PSM y código), que se pueden distinguir en horizontal, y varios aspectos separados (conceptual, persistencia, distribución, seguridad, navegación e interfaz de usuario.

Las transformaciones de PIM a PSM son transformaciones verticales, ya que pasan de un nivel de abstracción mayor a un nivel de abstracción menor, mientras que las transformaciones PSM a componente Java Server Faces (JSF) pueden ser oblicuas. Es decir, las transformaciones de PSM a componente JSF pueden tener componentes verticales y horizontales, ya que no solamente disminuyen el nivel de abstracción, sino que mezclan dos conceptos, el interfaz de usuario y la navegación.

Por tanto, en el caso de la Figura 2, los aspectos de navegación y presentación se especifican por separado a nivel de PIM, pero como JSF, que es la plataforma elegida para implementarlos, no presenta una separación clara de ambos, en el modelo resultante de la transformación tendremos ambos aspectos entremezclados. Es decir, las transformaciones han realizado un trabajo de *weaving*.

4. Características del transformador de modelos

Como ya se ha mencionado en la introducción, en [6] se proponía la arquitectura de un framework generador de aplicaciones. En la Figura 3, se puede observar que los principales componentes definidos en la misma, son: un editor de modelos, un validador de modelos, un editor de transformaciones, un transformador de modelos, un generador de código y un editor de código. De todos ellos, el componente que se ve más influenciado por el hecho de tener que tratar con transformaciones oblicuas es el transformador de modelos, por lo tanto, en adelante nos centraremos en analizar las implicaciones de implementar este tipo de transformaciones sobre el mismo.

En [1], se distingue entre generadores composicionales, y generadores transformacionales. Los generadores composicionales no realizan transformaciones que puedan redefinir la estructura modular de una aplicación. Los generadores transformacionales, sin embargo, no solamente realizan refinamientos, sino que también son capaces de trabajar con transformaciones que tengan un componente horizontal (bien sean transformaciones horizontales, bien sean transformaciones oblicuas). Este tipo de transformadores es más potente que el composicional.

Así que nuestro transformador de modelos ha de ser transformacional, y este hecho, es determinante en su arquitectura.

La elección de cómo se van a implementar las reglas de transformación va a ser otro punto determinante para diseñar la arquitectura del transformador de modelos. Atendiendo a la clasificación realizada en [7], se pueden adoptar tres enfoques distintos:

- Manipulación directa de modelos. Las herramientas que siguen este enfoque ofrecen al usuario acceso a una representación interna del modelo y una serie de Interfaces de Programador de Apliciones (APIs) para manipular la representación.

  La manipulación directa tiene la ventaja de que el lenguaje que se suele utilizar para trabajar con las APIs suele ser un lenguaje de uso común tal como Visual Basic o Java, con lo cual, los desarrolladores no necesitan un entrenamiento extra para escribir transformaciones. Sin embargo, estas APIs suelen restringir la clase de transformaciones que se pueden realizar. Además, como los lenguajes utilizados son de propósito general, no tienen abstracciones adecuadas para expresar transformaciones, por lo tanto, el escribir una transformación se convierte en una tarea que consume tiempo, y además, suelen ser algoritmos difíciles de mantener.

- Representación intermedia. La herramienta exporta el modelo a un formato estándar, normalmente XML, y una herramienta externa toma el modelo y lo transforma.

  La representación intermedia tiene la ventaja de que existen en el mercado muchas herramientas que exportan o importan modelos descritos en XMI, y que, por tanto, se pueden utilizar heramientas XML ya existentes, como XSLT para realizar transformaciones. Sin embargo, la definición de transformaciones, aunque sean simples, de modelos en XSLT requiere un esfuerzo considerable.

  Otro inconveniente importante de este tipo de arquitectura es que las transformaciones se realizan por lotes, con lo cual es difícil que el usuario tenga un diálogo interactivo con la herramienta.

- Soporte de lenguaje de transformación. La herramienta ofrece un lenguaje que proporciona un conjunto de constructores o mecanismos para expresar, componer y aplicar transformaciones de manera explícita.

  Según los autores de [7], este enfoque es el que ofrece un mayor potencial de los tres.

En la siguiente sección veremos cómo implementaremos las reglas de transformación, y por tanto, nos ayudará a escoger una de estas tres propuestas de arquitecturas.

5. Implementación de las reglas de transformación

A la hora de implementar las reglas de transformación que tiene que procesar nuestro generador de modelos, tenemos cuatro opciones distintas:

- Utilizando un lenguaje de programación y haciendo la codificación a mano.

- Utilizando un lenguaje de queries embebido, tal como XPath [8] o XQuery [9].
- Utilizando un sistema basado en reglas, tal como CLIPS, o Prolog.
- Definir un lenguaje, bien procedural, bien declarativo, bien una mezcla de ambos.

Con la primera opción se pueden obtener implementaciones más eficientes, pero el tener que escribir la codificación a mano es un gran inconveniente. El elegir esta opción implicaría el elegir una arquitectura de manipulación directa de modelo.

La segunda y tercera opción elimina algún trabajo de escritura, pero ambas presentan el inconveniente de que para utilizarlos, hay que adaptar el formato del árbol de sintaxis abstracta al formato con el que trabaja o bien el sistema de reglas, o bien el lenguaje de queries embebido.

La segunda opción para expresar las transformaciones implicaría una arquitectura de representación intermedia, mientras que la última determina una arquitectura de soporte de lenguaje de transformaciones.

Como uno de los objetivos propuestos a la hora de realizar el framework es aprovechar las herramientas existentes, entre todas estas opciones, la más viable es el uso de XPath o XQuery o XSLT, para poder trabajar con todas las herramientas ya disponibles en XML. Como ya se ha comentado en la sección 4, este enfoque requiere una experiencia considerable para definir las transformaciones, así que, se tendrá que proporcionar una forma de aligerar este esfuerzo, bien definiendo un lenguaje por encima de XSLT, bien definiendo un lenguaje específico de dominio para trabajar con estas transformaciones.

6. Conclusiones y trabajos futuros

En este artículo se ha puesto de manifiesto la importancia del tratamiento de reglas de transformación oblicuas en el caso de que el propio transformador tenga que actuar como `weaver`. Además, se ha visto cómo la decisión de tratar con transformaciones oblicuas en el generador de modelo, obliga a tener un generador de tipo transformacional, descartando los generadores composicionales.

De la misma forma, a la hora de implementar las transformaciones, se ha determinado que la mejor opción, según los requisitos impuestos al framework, es utilizar SQL embebido en algún lenguaje del tipo XQuery o XPath, lo que implica una arquitectura de tipo representación intermedia.

Como trabajos futuros, estamos trabajando en la definición de transformaciones a nivel de metamodelos, para conseguir el tejido de los mismos.

Referencias

[1] Czanercki, K., Eisenecker, U.W. *Generative Programming. Methods, Tools and Applications* . Addison Wesley, 2000.

[2] France, R., Bieman, J.M. *Multi-View Software Evolution: A UML-based Framework for Evolving Object-Oriented Software.* In the Proceedings of the International Conference on Software Mainteinance (ICSM 2001). November, 2001.

[3] Greenfield, J., Short, K., Cook, S. , Kent, S. *Software Factories. Assembling Applications with Patterns, Models, Frameworks, and Tools.* Wiley Publishing, Inc, 2004.

[4] Kiczales, G., Lamping, J., Mendhekar, A. and Maeda, C., Lopes, C.,Loingtier, J.M., Irwin, J. *Aspect-Oriented Programming.* 11th Europeen Conf. Object-Oriented Programming, 1997. Eds. M. Akşit and S. Matsuoka. LNCS 1241, pp. 220-242, Springer Verlag.

[5] Object Management Group. *MDA Guide. Version 1.0.1*, OMG, 2003.

[6] Reina, A. M., Torres, J. *Separación de conceptos y MDA: Arquitectura de un framework*, I Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'04). Celebrado junto con las Jornadas de Ingeniería del Software y Bases de Datos (2004). Málaga, España. November, 2004. ISBN: 84-688-8870-2.

[7] Sendall, S., Kozaczynski, W. *Model Transformation - The Heart and Soul of Model-Driven Sofware Development* IEEE Software. Vol 20, n° 5, pp. 42-45. Sept/Oct 2003.

[8] W3C. *XML Path Language (XPath) 2.0* W3C Working Draft 04 April 2005 (http://www.w3.org/TR/xpath20/)

[9] W3C. *XQuery 1.0: An XML Query Language* W3C Working Draft 04 April 2005.(http://www.w3.org/TR/xquery/)