Modelando aspectos con lenguajes específicos de dominio

A. M. Reina, J. Torres, M. Toro, and M. J. Escalona

Dpto. de Lenguajes y Sistemas Informáticos,
Universidad de Sevilla,
Avda. Reina Mercedes, s/n
41012 Sevilla, España
{reinaqu, jtorres, mtoro, escalona}@lsi.us.es

Resumen La industria del software se tiene que enfrentar tanto a la rápida evolución del entorno tecnológico como al cambio de los requisitos definidos por los clientes. Para cubrir estas necesidades han surgido nuevas propuestas. Por una parte, la filosofía MDA nació teniendo como objetivo la adaptación del software a diferentes tecnologías y plataformas a un coste mínimo. Por otra parte, la separación avanzada de conceptos se enfrenta a la variabilidad de los requisitos de los clientes. Tras detectar la necesidad de elevar el nivel de abstracción de los actuales lenguajes de modelado de aspectos, en este artículo proponemos el uso de la filosofía MDA junto con la definición de lenguajes de modelado específicos de dominio para cada aspecto para conseguir esta elevación. Para definir estos lenguajes de modelado específicos de dominio se propone el uso de extensiones UML, bien metamodelos, bien perfiles. Por último, se da una guía para decidir cuál es la técnica de extensión más apropiada para definir los constructores de los aspectos.

1. Introducción

En la actualidad uno de los principales problemas a los que se tiene que enfrentar la industria del software son los cambios, producidos tanto por la rápida evolución de los entornos tecnológicos como por la variabilidad de los requisitos de los clientes. Los investigadores han tomado consciencia de estas necesidades, y así, en la bibliografía se pueden encontrar líneas que están atacando estos problemas desde diferentes perspectivas. Por una parte, MDA (Model Driven Architecture) [25] es una propuesta que tiene como objetivo el minimizar el impacto de los cambios tecnológicos en el proceso de desarrollo del software. Por otra parte, la programación orientada a aspectos [16] se centra en la mejora de la evolución del software, ya que al tener bien localizados los conceptos y minimizadas las dependencias entre ellos, es más fácil realizar modificaciones.

Aunque las primeras propuestas en el campo de la orientación a aspectos fueron principalmente para separar adecuadamente los conceptos a nivel de programación [15,26,7,19], pronto surgió la necesidad de elevar el nivel de abstracción, apareciendo, así propuestas para modularizar los elementos que cortan

transversalmente los sistemas software en cualquier parte del ciclo de vida. Como consecuencia, se ha acuñado el término Desarrollo de Software Orientado a Aspectos (DSOA) para recoger al conjunto de técnicas y herramientas aplicadas s todo el ciclo de vida.

Centrándose en la etapa de diseño, se puede comprobar que la mayoría de las propuestas [6,11,31,30,1,35] están inspiradas en AspectJ [27], que muchas tienen como objetivo la generación de código, y que al final, acaban generando código de AspectJ. Como consecuencia de la revisión de las distintas propuestas para modelar aspectos durante esta etapa [28], se ha detectado la necesidad de elevar el nivel de abstracción de las mismas. Teniendo como principal objetivo el atacar esta necesidad, en este artículo se propone el uso de lenguajes específicos de dominio para el modelado de los diferentes conceptos junto con la adopción de la filosofía MDA. Así, en esta propuesta se trabaja, principalmente, con dos niveles de modelos: modelos independientes de plataforma (PIMs) y modelos dependientes de plataforma (PSMs). Además, para la definición de los lenguajes específicos de dominio se propone el uso de extensiones UML.

En la actualidad, un arquitecto puede escoger entre extensiones pesadas (definiendo metamodelos) o extensiones ligeras (usando perfiles). En este artículo, se propone también una guía para ver cuál es la técnica más adecuada para definir los constructores necesarios para expresar un aspecto concreto.

El artículo se estructura de la siguiente manera: en primer lugar, en la sección 2 se presenta un resumen de las conclusiones obtenidas al analizar las distintas propuestas de modelado de aspectos a nivel de diseño. A continuación, se explica la propuesta 3, tomando un ejemplo para hacer más fácil la exposición. Por último, se concluye el trabajo y se señalan las líneas futuras de investigación.

2. Analizando las propuestas de modelado de aspectos

En [28] se han examinado y analizado las distintas propuestas de la bibliografía para modelar aspectos. En este apartado se hace un resumen de las principales conclusiones obtenidas, ya que éstas son el principal motor de nuestra propuesta. Los resultados de este análisis son:

1. La gran mayoría de las propuestas analizadas definen un lenguaje de modelado de propósito general, es decir, se propone el mismo conjunto de constructores para modelar cualquier aspecto.
2. No hay un acuerdo general acerca del tipo de mecanismo de extensión más apropiado para definir dichos constructores, y así existen tanto propuestas que utilizan *extensiones pesadas* [12,9] como otras que utilizan *extensiones ligeras* [35].
3. Una gran mayoría de las propuestas trabajan con constructores que están muy cercanos a los definidos en los lenguajes de programación.
4. Un gran número de propuestas se inspira en AspectJ (solamente [9] y [12] se basan en HyperJ y las Colaboraciones Aspectuales [13], respectivamente).

Queda patente, por tanto, que las actuales propuestas son dependientes de plataforma, en el sentido de que se pensaron buscando la generación de código orientado a aspectos. Pero en determinadas ocasiones, el cliente puede imponer trabajar con plataformas que no sean orientadas a aspectos. También se pone de manifiesto la necesidad de elevar el nivel de abstracción en el modelado de aspectos, ya que los constructores propuestos son demasiado cercanos a los definidos en lenguajes de programación.

Finalmente, cabe resaltar que de las propuestas analizadas, las más cercanas a la nuestra, en el sentido de que mezclan orientación a aspectos y MDA, son [22] y [18]. En la primera se propone un *framework* para incorporar conceptos de orientación a aspectos en el modelado y la filosofía MDA. Por otra parte, en [18], se propone la especificación de componentes y aspectos utilizando su propio lenguaje de modelado, aunque se deja patente la necesidad de estudiar cómo se pueden modelar los mismos.

3. MDA y el modelado de aspectos con lenguajes específicos de dominio

Una vez analizadas las propuestas de la sección anterior aparecen dos cuestiones relevantes: primero, que las propuestas son dependientes de plataformas orientadas a aspectos, y, segundo, que la mayoría utilizan constructores demasiado cercanos a los lenguajes de programación.

Para afrontar estos inconvenientes, se ha echado la vista un poco hacia atrás, hacia los orígenes de la programación orientada a aspectos, de tal forma que se propone una situación parecida a la de entonces, y con la salvedad de que ahora se pueden solventar los principales problemas que hicieron rechazar este tipo de propuestas: En los inicios de la programación orientada a aspectos había planteamientos para trabajar con lenguajes de aspectos de propósito específico como Cool y RIDL [21], pero este tipo de lenguaje tenía el problema de que no podían soportar aspectos distintos de aquellos para los que fueron diseñados. Por contra, presentaban un nivel de abstracción más alto que el del lenguaje base, manejando conceptos del dominio del aspecto.

Si se traslada esta situación al diseño, podemos gozar de las ventajas de este tipo de planteamiento, y superar sus inconvenientes gracias a la utilización de UML como lenguaje de modelado, ya que, aunque es un lenguaje de modelado de propósito general, proporciona mecanismos de extensión que permiten ajustarlo a un dominio concreto. Aunque se mantiene el inconveniente de que un diseñador puede tener que tratar con multitud de extensiones, puede ser suavizado por el hecho de que los diseñadores pueden especializarse, y tratar así, solamente con un conjunto pequeño de extensiones.

En la actualidad, un arquitecto tiene que determinar qué técnica de extensión es la mejor para modelar un dominio específico: una basada en metamodelo, o una basada en perfiles. Así que, para especificar el lenguaje de modelado de un aspecto, lo primero que habrá que decidir será la técnica de extensión que se va a utilizar. En esta propuesta se sugiere adaptar la lógica que define Desfray en [10]

para elegir la técnica de extensión más adecuada para expresar los constructores del lenguaje del aspecto. En la figura 1 se muestra una tabla con los puntos adaptados de Desfray a tener en cuenta para elegir una técnica de metamodelado (primera columna) o de perfil (segunda columna).

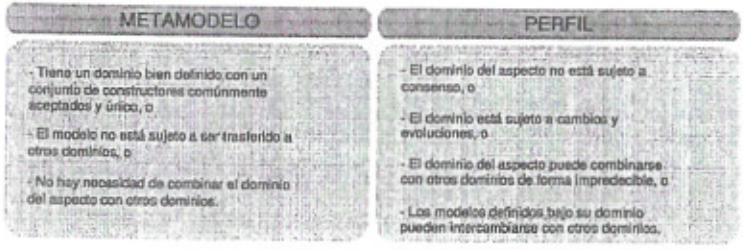

Figura 1. Adaptación de la guía de Desfray para la elección de la técnica de extensión más adecuada

Además de las cuestiones planteadas en Desfray, una ventaja importante de elegir una extensión mediante perfil es que se pueden utilizar herramientas genéricas que trabajan con UML, mientras que es más probable que las extensiones pesadas sean más dependientes de vendedores concretos y de que éstos extiendan sus herramientas, es decir, que no sean estándares. La principal desventaja de utilizar perfiles es que no son tan potentes semánticamente como la extensión mediante metamodelos.

3.1. Aplicando la propuesta

Para que la idea global de la propuesta quede más clara, se tomará como ejemplo una aplicación web distribuida genérica y se intentará aplicar estas ideas a la misma. El primer paso a dar será determinar cuáles son los conceptos de alto nivel que se van a especificar de forma separada. Una aplicación web distribuida típica se compone, al menos, de los siguientes conceptos: interfaz de usuario, navegación, seguridad, distribución y persistencia. Se necesita, por tanto, mantener modelos separados para cada uno de ellos.

Al seguir la filosofía de MDA, hay que definir al menos dos niveles de modelos: modelos independientes de la plataforma y modelos dependientes de la plataforma. En la figura 2 se muestra la estructura de esta aplicación web genérica. En este caso, y en el nivel de modelos independientes de la plataforma, se tienen separados en paquetes (al igual que en [12] consideramos que los paquetes deberían ser "ciudadanos de primera clase") los distintos conceptos que componen el sistema. A este nivel, cada aspecto estará modelado con un lenguaje específico para su dominio.

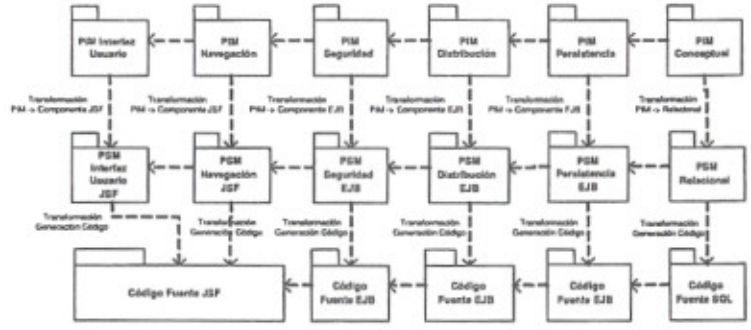

Figura 2. Estructura de una típica aplicación web distribuida

En una situación ideal, esta separación conseguida en el nivel de los modelos independientes de plataformas debería mantenerse hasta llegar al código fuente, pero hay que tener en cuenta que algunas veces se han de utilizar plataformas que no separan los conceptos de forma adecuada debido a las imposiciones del cliente. En la figura 2 se aplican una serie transformaciones a los modelos independientes de la plataforma que especifican los aspectos de interfaz de usuario y navegación, obteniéndose como resultado unos modelos dependientes de la plataforma JSF (Java Server Faces)[8]. En este caso, JSF no proporciona una separación completa de navegación e interfaz de usuario, por lo que se puede optar, en este caso por mantener la separación de conceptos hasta nivel de modelos dependiente de la plataforma, pero, sin embargo, el código generado no presentará una separación limpia de los mismos. Así, en este caso, la mezcla se realizará a nivel de PIM, y será el transformador de modelos el que se encargue de realizarla. En caso de que la plataforma soporte aspectos, la mezcla se podrá retrasar hasta nivel de compilación (en el caso de aspectos estáticos) o hasta ejecución (si hablamos de aspectos dinámicos).

En la bibliografía ya se pueden encontrar propuestas de extensiones a UML, que pueden ser válidas como lenguajes específicos de dominio de nuestros conceptos, tanto a nivel de PIM, como a nivel de PSM. En la tabla 1, se han resumido algunas de ellas, representándose, por columnas, el nombre del aspecto al que modelan, el mecanismo de extensión que han utilizado, si es una propuesta para aplicar a nivel de PIM o de PSM, la referencia bibliográfica donde se detalla la propuesta; y, por último, un breve comentario sobre alguna de ellas.

4. Conclusiones y trabajo futuro

En este artículo se ha puesto de manifiesto la necesidad de elevar el nivel de abstracción de las actuales propuestas para el modelado de aspectos a nivel de diseño. Para conseguirlo, se propone utilizar la filosofía MDA y añadir un

Aspecto	Mecanismo Extensión	Nivel	Ref.	Comentario
Interfaz de usuario y navegación	Metamodelo	PIM	[23]	Se argumenta por qué es mejor la técnica de metamodelado .
	Metamodelo	PIM	[5]	Separa navegación e interfaz.
	Metamodelo	PIM	[17]	Separa navegación e interfaz.
Seguridad	Metamodelo	PIM	[20]	Define un lenguaje de modelado llamado SecureUML enfocado en el control de acceso .
Distribución	Perfil	PIM	[29]	
Persistencia	Perfil	PIM	[34]	
	Perfil	PSM	[3]	Modelado de persistencia para bases de datos relacionales.

Tabla 1. Extensiones a UML encontradas en la bibliografía aplicables a los aspectos

nuevo nivel con modelos independientes de la plataforma, dejando para la capa de modelos dependientes de la plataforma las actuales propuestas de modelado. Aunque estas propuestas no se descartan, son cercanas a la implementación, y por lo tanto, se dejan a nivel de PSM.

Para especificar conceptos a nivel de PIMs se propone el uso de lenguajes de modelado específicos de dominio. Estos lenguajes se especificarán utilizando extesiones de UML bien, metamodelos, bien perfiles (dependerá de la naturaleza del aspecto el elegir una técnica de extensión u otra). También se propone una guía para elegir el tipo de extensión más adecuado para especificar el lenguaje de modelado de un concepto.

Como futura línea de investigación, estamos trabajando en un framework para dar soporte a todas esta ideas, y que trabaje con un repositorio de metamodelos y perfiles. Así mismo, queremos estudiar la problemática del transformador de modelos como *weaver*, centrándonos, especialmente en las incompatibilidades entre aspectos.

Referencias

1. O. Aldawud, T. Elrad, and A. Bader. UML profile for aspect-oriented software development. In Aldawud et al. [2].
2. O. Aldawud, M. Kandé, G. Booch, B. Harrison, and D. Stein, editors. *Third International Workshop on Aspect Oriented Modeling*, March 2003.
3. S. W. Ambler. Towards a Relational Persistence Model Profile for UML 2.0. In COM00 [24].
4. *Workshop on Aspect-Oriented Modeling with UML (AOSD-2002)*, March 2002.
5. L. Baresi, F. Garzotto, L. Mainetti, and P. Paolini. Meta-modeling Techniques Meet Web Application Design Tools. In *Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, volume 2306 of *LNCS*, pages 294–307. Springer-Verlag, 2002.
6. M. Basch and A. Sánchez. Incorporating aspects into the UML. In Aldawud et al. [2].
7. L. Bergmans and M. Akşit. Composing crosscutting concerns using composition filters. *Comm. ACM*, 44(10):51–57, October 2001.

8. P. S. Bhogill. An Introduction to Java Server Faces. *Java News Brief*, Aug 2003.
9. S. Clarke. Extending standard UML with model composition semantics. *Science of Computer Programming*, to appear.
10. P. Desfray. UML Profiles versus Metamodeling Extensions... an Ongoing Debate. In COM00 [24].
11. I. Groher and S. Schulze. Generating aspect code from UML models. In Aldawud et al. [2].
12. S. Herrmann. Composable designs with UFA. In AOSD-UML02 [4].
13. S. Herrmann and M. Mezini. Combining composition styles in the evolvable language LAC. In *Workshop on Advanced Separation of Concerns in Software Engineering (ICSE 2001)*, May 2001.
14. Mohamed Kandé, Omar Aldawud, Grady Booch, and Bill Harrison, editors. *Second International Workshop on Aspect-Oriented Modeling with UML (<<UML>>2002)*, September 2002.
15. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W.G. Griswold. Getting started with AspectJ. *Comm. ACM*, 44(10):59–65, October 2001.
16. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Akşit and S. Matsuoka, editors, *11th Europeen Conf. Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag, 1997.
17. N. Koch and A. Kraus. Towards a Common Metamodel for the Development of Web Applications. In J. M. Cueva Lovelle, B. Martín González Rodríguez, L. Joyanes Aguilar, J. E. Labra Gayo, and M. del Puerto Paule Ruíz, editors, *Web Engineering, International Conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003, Proceedings*, volume 2722 of *Lecture Notes in Computer Science*, pages 497–506. Springer, 2003.
18. V. Kulkarni and S. Reddy. Supporting Aspects in MDA. In WISME-UML03 [33].
19. K. Lieberherr, D. Orleans, and J. Ovlinger. Aspect-oriented programming with adaptive methods. *Comm. ACM*, 44(10):39–41, October 2001.
20. T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modelling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on the Unified Modelling Language*, 2002.
21. C. V. Lopes. *D: A Language Framework for Distributed Programming*. PhD thesis, College of Computer Science, Northeastern University, 1997.
22. S. J. Mellor. A Framework for Aspect-Oriented Modeling. In UML-AOM03 [32].
23. P. Muller, P. Studer, and J. Bézivin. Platform Independent Web Application Modeling. In P. Stevens, J. Whittle, and G. Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *LNCS*, pages 220–233. Springer, 2003.
24. OMG. *Proceedings of the First Workshop on UML in the .COM Enterprise: Modeling CORBA, Components, XML/XMI and Metadata*, 2000.
25. OMG. MDA Guide Version 1.0. Technical Report omg/2003-05-01, OMG, May 2003.
26. H. Ossher and P. Tarr. The shape of things to come: Using multi-dimensional separation of concerns with Hyper/J to (re)shape evolving software. *Comm. ACM*, 44(10):43–50, October 2001.
27. Xerox PARC. Aspectj home page. web, 2002.
28. A.M. Reina, J. Torres, and M. Toro. Towards developing generic solutions with aspects. In *Proceeding of the Workshop in Aspect Oriented Modelling held in conjunction with the UML 2004 Conference*, oct 2004.