Desarrollo de software con aspectos dirigido por modelos*

Lidia Fuentes y Pablo Sánchez

Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Málaga (España)
{lff,pablo}@lcc.uma.es

Resumen Las técnicas orientadas a aspectos han demostrado ser una excelente solución para manejar la separación de propiedades (separation of concerns[1]), permitiendo la construcción de software en menor tiempo, con menor coste, más fiable y de más fácil mantenimiento y evolución. Sin embargo el uso de las técnicas aspectuales en proyectos de gran escala se ha visto limitado por la falta de métodos que permitan aplicarlas desde las fases más tempranas del desarrollo software, estando bajo desarrollo diversas propuestas para cada fase. para realizar la separación de propiedades en las diferentes etapas del ciclo de vida. Por otra parte, en el desarrollo dirigido por modelos se propone la construcción de software como la transformación sucesiva de modelos, desde un alto nivel de abstracción hasta el nivel de implementación en un plataforma concreta. En este trabajo comentamos brevemente algunas de las propuestas existentes para el desarrollo orientado a aspectos, y sus posibilidades de integración en un desarrollo dirigido por modelos.

1. Introducción

El enfoque orientado a aspectos [1] ha demostrado ser una tecnología potente para manejar la separación de propiedades [2]. Esta separación de propiedades permite la construcción de software con una mayor modularidad, facilitando su reusabilidad, fiabilidad, mantenimiento y evolución. El principal problema para la adopción de estas técnicas en proyectos de gran escala es la falta de un proceso claro de desarrollo orientado a aspectos. Hasta ahora la identificación, especificación e implementación de los aspectos la realizaban los programadores, principalmente *ad-hoc*, en la fase de implementación, viéndose obligados a *rediseñar* el sistema. Al no disponer de metodologías de desarrollo suficientemente maduras los desarrolladores ven dificultada su tarea al ser sobrecargados con nuevas funciones.

No existe una definición consensuada de aspecto que sirva para los diferentes niveles de desarrollo. Existen aspectos a nivel de ingeniería de requisitos y diseño

* Este trabajo ha sido financiado en parte por los proyectos IST-2-004349-NOE AOSD-Europe y MCYT TIC2002-04309-C02-02

[1] El término inglés *concern* a castellano se traduce como propiedad en este artículo

(denominados *early-aspects*), a nivel de implementación, etc. En general un aspecto, usando la terminología de AspectJ [3], puede verse como una pieza de comportamiento extra (*advice*) que se ejecuta en un *punto de corte* (*pointcut*) específico dentro la ejecución de una aplicación. Los puntos de corte se especifican de acuerdo con un *modelo de puntos de intercepción* (*joinpoint*) en los cuales se permite modificar la ejecución de una aplicación. Aunque en esta definición se ha usado terminología heredada de AspectJ, pretende incluir otra serie de enfoques como filtros de composición [4], hiperespacios [5], etc. donde existen conceptos similares expresados en otros términos.

Para cada fase del desarrollo software existen diferentes propuestas para realizar la separación de propiedades, y que resultarán más o menos apropiadas en función de las características del sistema a construir y del equipo de desarrollo involucrado. Las diferentes propuestas varían en diversas cuestiones: La composición de los aspectos se puede realizar de forma estática, en tiempo de compilación, de forma dinámica, en tiempo de carga o incluso ejecución. Los puntos de intercepción pueden hacer referencia a elementos internos de clases y componentes (modelo invasivo) o sólo a su comportamiento externo (modelo no-invasivo). Los puntos de intercepción se pueden componer mediante diferentes operadores para formar diferentes puntos de corte. Respecto a todas estas opciones no existen ni estándares ni modelos consensuados acerca de que conceptos deberían representarse en cada nivel y cómo.

Por tanto, además de un proceso de desarrollo que nos permita identificar aspectos desde las fases más tempranas del ciclo de vida y manejarlos en la totalidad de él, sería interesante disponer de un método que nos permita tener la libertad de elegir entre las diferentes propuestas existentes para un nivel y movernos a propuestas del mismo nivel o de nivel inferior sin tener que refactorizar o rehacer los modelos ya creados. Es particularmente interesante desde un nivel poder generar diferentes niveles destino, sin tener que modificar el nivel fuente.

En el desarrollo software dirigido por modelos (DSDM) los modelos ya no son simples medios para describir software, sino la pieza fundamental de su desarrollo. Los modelos existentes en una fase (análisis, diseño, etc.) se transforman (automática, semiautomáticamente o manualmente) sucesivamente hasta derivar la aplicación. MDA [6] (*Model Driven Architecture*) es la propuesta de la OMG para el desarrollo dirigido por modelos, los cuales se basarían en los estándares propuestos por esta organización: UML [7], XMI [8], MOF [9], etc.

El propósito de este artículo es presentar las diferencias entre las propuestas existentes para manejar la separación de propiedades a lo largo del ciclo de vida, dando indicaciones de como hacer uso de MDA como elemento integrador entre las diferentes propuestas.

En adelante este artículo se estructura como sigue: En la Sección 2 se comentan algunas ideas y retos planteados a la hora de integrar desarrollo de software dirigido por modelos y aspectos. En la sección 3 se comentan algunas de las propuestas para realizar el análisis de requisitos, en la Sección 4 algunas de las correspondientes al diseño, y en la Sección 5 se comentan algunas de las

plataformas disponibles para implementar aspectos. Por último en la Sección 6 se exponen algunas conclusiones.

2. MDA y DSOA

El desarrollo de una aplicación dentro del marco de trabajo de MDA implica la definición de un conjunto de modelos pertenecientes a diversos niveles conceptuales, así como un conjunto de transformaciones que permitan pasar de un nivel a otro.

El patrón básico de MDA se compone de un modelo independiente de la plataforma (PIM, *Platform Independent Model*), el cual se transforma en un modelo dependiente de la plataforma (PSM, *Platform Specific Model*). Qué se entiende por plataforma, y por independiente o dependiente depende del nivel donde nos encontremos y cual sea nuestro objetivo.

En este trabajo consideramos cada fase del ciclo de vida como un nivel conceptual o *plataforma*. Dentro de cada fase es posible a su vez diferenciar subniveles conceptuales. Las transformaciones pueden darse por tanto entre modelos de un nivel superior a uno inferior (ej. de diseño a implementación), como entre modelos a un mismo nivel (ej. de un modelo de diseño a otro). Así por ejemplo, un modelo de diseño Theme/UML [10] podemos transformarlo en un modelo de diseño detallado, listo para su implementación en un lenguaje concreto como AspectJ [3] o transformarlo a otro modelo de diseño a alto nivel, con un enfoque diferente, como CAM [11].

Para la implantación de MDA precisamos de un serie de elementos: (1) un conjunto de modelos y herramientas de modelado, (2) un formato de almacenamiento e intercambio de modelos (2) un lenguaje y una herramienta para construir transformaciones. Para el primer requisito tenemos dos opciones principales: definir nuestro propio metamodelo (basado en MOF [9]) o modificar UML [7] mediante perfiles [12]. En el primero de los casos poseemos una mayor libertad, pero carecemos de herramientas que nos permitan trabajar adecuadamente con MOF. En el segundo de los casos podemos hacer uso de editores UML. En ambos casos, XMI [8] ofrece un formato estándar para almacenar cualquier modelo basado en MOF, incluido UML. QVT [13] (*Query, View, Transform*) es el estándar propuesto por la OMG para la construcción de lenguajes de transformación. No obstante, QVT es un estándar en sus primeras fases que no ha obtenido aún la suficiente penetración. Existen otros lenguajes de transformación no estándares, pero ampliamente usados, como ATL [14].

La propuesta de MDA recoge diversas tipos de transformaciones: (1) Transformación basada en metamodelos, donde primero se transforman los metamodelos independiente y dependiente, y luego se transforman los modelos como instancias de dichos metamodelos. (2) Transformación basada en marcas, donde el PIM se *marca* con conjunto de marcas extraído del metamodelo del PSM para proporcionar la información adicional necesaria para guiar la transformación. (3) Transformaciones basadas en tipos, donde se especifican dos conjuntos de tipos, uno para construir el PIM y otro para construir el PSM, y la transforma-

ción se realiza en base a la correspondencia entre ambos conjuntos de tipos. (4) Transformaciones basadas en patrones, donde patrones del PIM se transforman en patrones del PSM (5) Transformación por mezcla de modelos, donde la transformación toma como entrada un PIM más uno o más modelos adicionales, con objeto de producir un PSM.

3. Fase de análisis de requisitos

La primera etapa dentro de nuestro proceso es la captura de aquellos requisitos que los usuarios esperan que el sistema satisfaga. Actualmente existen diversas propuestas para la captura de requisitos: puntos de vista [15], casos de uso [16], dirigida por objetivos [17], etc. muchas de las cuales están siendo extendidas para dar soporte al concepto de propiedad o aspecto.

El enfoque basado en casos de uso [18][19] propone modelar las funcionalidades propias del sistema como *casos de uso base*. que pueden poseer *puntos de extensión*, puntos concretos dentro de un determinado caso de uso, donde es posible añadir comportamiento, el cual se especificaría mediante casos de uso denominados *extensiones*. De este modo podemos realizar la captura de requisitos en dos fases: una primera donde se modela la funcionalidad básica y propia del sistema mediante los casos de uso base, que posteriormente, en la segunda fase, se *adornan* con los requisitos representando aspectos. Dado que las extensiones son comunes a diversas aplicaciones, podrían reutilizarse, tomándose de un repositorio genérico.

En [20] se propone una descomposición de los requisitos con independencia de su naturaleza funcional o extra funcional. Cada propiedad se representaría como la cara de un hipercubo, de forma que podemos escoger cualquier conjunto de propiedades como base para proyectar la influencia de otro conjunto de propiedades sobre ellas. La propuesta propone un mecanismo para la composición de propiedades y el análisis de la influencia de unas propiedades sobre otras.

En general las diversas propuestas para la captura de requisitos orientadas a aspectos proponen técnicas donde se pueden distinguir varios elementos: requisitos (funcionales o extra funcionales) y reglas de composición.

Uno de los principales problemas a resolver es como representar los modelos de requisitos de forma que se puedan integrar dentro de la propuesta de MDA. En el primer caso se trata de un modelo UML, que puede almacenarse fácilmente XMI. En el segundo de los casos se hace uso de documentos XML, fácilmente procesables por una computadora, pero no en la línea de MDA. En este último caso la solución sería la definición de un metamodelo MOF acorde al Schema XML usado para la construcción de los documentos de requisitos.

4. Fase de diseño

La segunda fase del ciclo de vida comprende un diseño de alto nivel de la arquitectura del sistema, que debe mostrar como éste funciona, sin comprom-

eterse con detalles específicos de implementación. Existen también una amplia variedad de modelos de diseño, la mayoría extensiones de UML.

Theme/UML [10] permite modelar la estructura y comportamiento de los aspectos usando UML. Theme/UML se basa en el concepto de *tema*, definido como 'cualquier característica, propiedad o requisito de interés que debe ser manejado por nuestro sistema'. Cada tema encapsula dentro de un paquete UML, contieniendo todos los elementos que lo modelan. Existen dos clases diferentes de temas: parametrizados y no parametrizados. Los parametrizados son plantillas UML, conteniendo una serie de elementos como parámetros. El modelo final de la aplicación se obtiene especificando relaciones de ligadura, que instancien los parámetros, entre los temas parametrizados y los no parametrizados.

CAM [11] es un modelo para el desarrollo basado en componentes y aspectos, en el cual los componentes se comunican usando nombres de rol, de acuerdo con un modelo de bajo acoplamiento. Los aspectos se aplican en tiempo de ejecución al envío/recepción de mensajes y eventos de acuerdo con un modelo no invasivo.

Junto a los lenguajes de modelado podemos usar lenguajes de descripción de arquitecturas. La arquitectura de las aplicaciones CAM se puede describir fácilmente usando el lenguaje DAOP-ADL [21]. Existen otras propuestas que añaden aspectos a lenguajes de descripción de arquitecturas, como AO-Rapide [22]. Una de las opciones más interesantes de DAOP-ADL es la equivalencia entre un modelo CAM y su su descripción DAOP-ADL, pudiéndose generar uno a partir de otro.

El objetivo de una trasformación de un modelo de requisitos a un modelo de diseño sería transformar ciertos requisitos a elementos propios del sistema, mientras que otros requisitos se transformarían a aspectos. Algunas propiedades extrafuncionales se identificarán como aspectos a nivel de diseño, mientras que para otras propiedades (funcionales o no) podría no existir su correspondiente artefacto en la fase de diseño. Por ejemplo, la propiedad de coste podría no corresponderse con ningún artefacto concreto a nivel de diseño, aunque varios elementos podrían estar condicionados por dicha propiedad. La trazabilidad de esta propiedad debería mantenerse, de manera que para posteriores modificaciones del sistema, se pueda estudiar su impacto sobre ésta propiedades, y que elementos del diseño fueron originalmente motivados por su presencia. Las reglas de composición de requisitos deberían poder transformarse a artefactos en el nivel de diseño. Por ejemplo, una relación composicional entre dos requisitos podría corresponderse con una relación de ligadura en Theme/UML o con la aplicación de un aspecto a un mensaje en CAM.

5. Plataformas y lenguajes orientados a aspectos

Para la implementación de nuestra aplicación debemos elegir un lenguaje o plataforma que de soporte al concepto de aspecto. Existen multitud de plataformas y lenguajes posibles, cada uno con sus ventajas o inconvenientes: JAC [23], CAM/DAOP [24], AspectJ [3], etc.

Estas plataformas se diferencian en: (1) El momento en que se realiza la composición de los aspectos (tiempo de compilación, carga o ejecución), (2) Donde y cómo se declaran los puntos de corte (de manera conjunta o separada al comportamiento de los aspectos y de acuerdo con un modelo invasivo o no invasivo), (3) Cómo se aplica un aspecto en un punto de corte (antes, después, en sustitución, etc.) (4) Cómo se despliegan los aspectos (por instancia, por clase, etc.) (5) Sobre que entidades se aplican los aspectos (componentes, clases, etc.)

A la hora de transformar una modelo de diseño en un modelo de implementación debemos tener en cuanta esta serie de elementos. Dependiendo de las características de nuestro sistema habrá ciertos sistemas que encajen peor o mejor y otros que, directamente, no nos sirvan. Para la transformación del modelo de puntos de corte, por ejemplo, puede ser que nuestra plataforma destino no permita recoger el modelo de intercepción planteado en el modelo de diseño. Por ejemplo, un diseño siguiendo un modelo invasivo requiere de un lenguaje o plataforma con un modelo de puntos de intercepción invasivo. Sin embargo, un modelo no invasivo podría implementarse sobre un modelo invasivo

6. Conclusiones

En este artículo se ha mostrado como integrar un proceso de desarrollo software orientado a aspectos, desde sus primeras fases, con el desarrollo dirigido por modelos. La idea es detectar los aspectos lo más pronto posible dentro del ciclo de desarrollo, y generar la mayor parte posible de los modelos necesarios para la siguiente etapa a partir de los modelos de la etapa anterior. Aparte de las propuestas mostradas en este artículo existen muchas otras para abordar la separación de propiedades. Según las características de nuestra aplicación y equipo de desarrollo deberíamos elegir una u otra. Interesantes trabajos acerca del estado del arte de las diferentes propuestas aspectuales pueden encontrarse en [25].

Sean cuales sean los enfoques elegidos, los modelos construidos en una fase deberían poder transformarse en modelos para la siguiente fase, de acuerdo con la propuesta de MDA. En las transformaciones ciertos aspectos se perderán, otros se transformarán y cada fase del desarrollo podría introducirse aspectos nuevos. No obstante debería preservarse su trazabilidad. Para realizar estas transformaciones deben resolverse varias cuestiones: (1) cómo se definen los aspectos a lo largo del ciclo de vida (extensiones, temas parametrizados, etc.) y como se establecen las relaciones entre ellos (2) cómo se aplican los aspectos a los elementos funcionales de la aplicación (puntos de extensión, ligaduras, puntos de corte, etc.). (3) Qué nivel de detalle debe especificarse en cada etapa del ciclo de vida (cuando especificar el modelo de composición deseado, modelo de puntos de intercepción, etc.).

Referencias

1. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: ECOOP, Berlin (1997)

2. Tarr, P., Ossher, H., Harrison, W., Sutton, S.: N degrees of separation: Multi-dimensional separation of concerns. In: Proc. of the 21st ICSE, Los Angeles, CA, USA (1999) 107–119
3. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of AspectJ. In: Proc. ECOOP 2001, Berlin (2001) 327–353
4. Bergmans, L., Aksit, M.: Composing crosscutting concerns using composition filters. Communications of the ACM **44 (10)** (2001) 51–57
5. Ossher, H., Tarr, P.: Multi-dimensional separation of concerns and the hyperspace approach. In: Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. (2000)
6. OMG: MDA Guide (draft version 2). http://www.omg.org/docs/omg/03-06-01.pdf (2003)
7. OMG: The unified modelling language web site. http://www.uml.org (2004)
8. XMI: Xml metadata interchange (XMI) specification. http://www.omg.org/docs/formal/03-05-02.pdf (2003)
9. OMG: Meta object facility MOF 2.0 core specification. http://www.omg.org/docs/ptc/03-10-04.pdf (2004)
10. Clarke, S., Walker, R.J.: Towards a standard design language for aosd. In: Proc. of the 1st AOSD. (2002)
11. Pinto, M., Fuentes, L., Troya, J.: A component and aspect dynamic platform. The Computer Journal (2005) Accepted to be published.
12. Fuentes, L., Vallecillo, A.: Una introducción a los perfiles UML. Novática (2004)
13. OMG: Revised submission for MOF 2.0 Query/View/Transformation rfp (ad/2002-04-10) (2005)
14. Bézivin, J., Valduriez, P., Jouault, F.: The ATL home page. http://www.sciences.univ-nantes.fr/lina/atl/ (2005)
15. Finkelstein, A., Sommerville, I.: The viewpoints FAQ. BCS/IEE Software Engineering Journal **11(1)** (1996)
16. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992)
17. Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: 5th Int. Symp. on RE. (2001) 249–261
18. Jacobson, I., Ng, P.: Aspect-Oriented Software Development with Use Cases. Addisson-Wesley (2004)
19. Jacobson, I.: Use cases and aspects. working seamlessly together. Journal of Object Technology (2003)
20. Moreira, A., Rashid, A., Araujo, J.: Multi-dimensional separation of concerns in requirements engineering. In: International Conference on Requirements Engineering, IEEE Computer Society (2005) To Appear.
21. Pinto, M., Fuentes, L., Troya, J.M.: DAOP-ADL: An architecture description language for dynamic component and aspect-based development. In: GPCE 2003. (2003) 118–137
22. Navasa, A., Palma, K., Murillo, J., Eterovic, Y.: Dos modelos arquitectónicos para el DSOA. In: 1st DSOA, JISBD, Málaga, España (2004)
23. Pawlak, R., Seinturier, L., Duchien, L., Florin, G.: JAC: A Flexible Framework for AOP in Java. In: Reflection'01, Kyoto, Japan (2001) 1–24
24. Pinto, M.: CAM/DAOP: Component and Aspect Based Model and Platform. PhD thesis, Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga (2004) Available only in Spanish.
25. AOSD-Europe: European Network of Excellence - Public Documents . http://www.aosd-europe.net/documents/index.htm (2005)