MDA e Ingeniería de Requisitos para Líneas de Producto

Bruno González-Baixauli, Miguel A. Laguna
Departamento de Informática, Universidad de Valladolid,
Campus M. Delibes, 47011 Valladolid
{bbaixauli, mlaguna}@infor.uva.es

Resumen

Uno de los elementos clave para el éxito de una línea de productos software es la elicitación, representación y gestión de la variabilidad. En este contexto, se explora el papel que puede jugar la iniciativa *Model Driven Architecture* (MDA) en la construcción de modelos de dominio (expresados como grafos de características o *features*) y su posible transformación en modelos arquitectónicos. Dado que nuestro enfoque tiene como punto de partida un modelo de metas (*goals*), la imagen global es una secuencia de transformaciones (no necesariamente automáticas) de modelos: de *goals* a *features* y de ambos a una arquitectura inicial para la línea de producto. La trazabilidad de estas transformaciones representa al mismo tiempo una necesidad y un beneficio adicional.

1. Introducción

El desarrollo de líneas de producto software es, desde el punto de vista práctico, el enfoque que mayores éxitos ha alcanzado en el campo de la reutilización del software, gracias a la combinación de un desarrollo sistemático y a la utilización de componentes reutilizables de grano grueso. Sin embargo, el concepto de línea de producto es complejo y requiere un gran esfuerzo, tanto técnico como organizativo [1][5] de las empresas que se plantean su utilización. Nuestro planteamiento trata de suavizar el paso de un modelo de desarrollo convencional a otro que aproveche las ventajas de las líneas de producto. Para ello, entre otras iniciativas, hemos propuesto un proceso utilizando SPEM [23] para recoger las técnicas específicas de *Ingeniería de Líneas de Producto* en un proceso paralelo al de *Ingeniería de Aplicaciones* [18].

Una de las bases de nuestra propuesta consiste en utilizar un enfoque de ingeniería de requisitos orientada a metas o *goals*, que modela explícitamente la intencionalidad de un sistema (los *porqués*). La importancia de establecer la intencionalidad de un sistema software ha sido ampliamente reconocida pero no es habitualmente reflejada en la documentación del mismo. La ventaja de este enfoque es que permite estudiar distintas alternativas para los requisitos tanto del punto de vista funcional (*goals*) como no funcional (*soft-goals*), lo que evidentemente se adapta de forma natural al desarrollo de líneas de producto. Utiliza para ello árboles and/or que expresan puntos de variabilidad (diversas formas de alcanzar la misma meta). La diferencia fundamental entre *hard-goals* o simplemente *goals* y *soft-goals* estriba en que la satisfacción de los primeros se puede comprobar de forma clara, en tanto que la de los *soft-goals* no es posible. El modelo NFR [4] establece las correlaciones y la forma de análisis de dichos *soft-goals*.

La iniciativa Arquitectura dirigida por Modelos o *Model Driven Architecture* (MDA) ha sido propuesta por OMG [22] y está organizada sobre el concepto de modelo independiente de plataforma o PIM. Un PIM es una especificación de un sistema en términos de conceptos del dominio y con independencia de plataformas tecnológicas (como Microsoft .NET o Sun J2EE). El PIM puede transformarse automáticamente en un modelo específico de plataforma o PSM. Puesto que el mayor interés de MDA se encuentra en la manipulación y transformación de diferentes modelos, los modelos de variabilidad que proponemos para el análisis y desarrollo de una línea de producto son candidatos naturales a ser considerados como PIMs. Por tanto merece la pena analizar el papel que puede jugar MDA para transformar estos PIMs en modelos convencionales basados en UML.

El resto del artículo se organiza del siguiente modo: en la siguiente sección se introduce la ingeniería de requisitos para el desarrollo de líneas de producto y su relación con los conceptos de *goal* y *soft-goal*, utilizados como soporte para eñ análisis de la variabilidad. La sección 3 profundiza en las variadas posibilidades que ofrece MDA para relacionar y quizá transformar los distintos modelos involucrados. Finalmente, la sección 4 concluye el trabajo e indica los caminos a seguir para avanzar en esta línea.

2. Ingeniería de Requisitos orientada a Metas y Líneas de Producto

Nuestro trabajo en el campo de desarrollo de líneas de producto está basado en el modelo *mecano* de reutilización que utiliza una estructura compleja basada en tres niveles de abstracción: requisitos, diseño e implementación [9]. Estas tres categorías se corresponden con los artefactos fundamentales del desarrollo de líneas de producto [2]. El nivel de requisitos está representado por el modelo de dominio (o de negocio), los requisitos de la línea de producto y el modelo de variabilidad. Justamente la disciplina de *Requisitos para Líneas de Producto* es una de las claves para el éxito en el desarrollo de una familia software y por esta razón estamos dedicando nuestros mayores esfuerzos a mejorar la elicitación, representación y gestión de esos requisitos con especial hincapié en los aspectos de variabilidad.

En una primera versión del proceso, propuesto en [18], utilizamos casos de uso y *features* para definir la variabilidad de la línea de producto, basándonos en los trabajos del Software Engineering Institute [3]. La utilización de *features*, como forma de expresar los puntos de variación en un modelo está muy extendida y se basa en los trabajos de FODA [14] y FORM [13]. Los casos de uso son bien conocidos y fueron propuestos por Jacobson como base para su médodo [12]. Sin embargo y a pesar de haber sido utilizadas en muchos proyectos, ambas técnicas tienen en común un punto débil: están enfocadas más a la solución que al problema. Por esta razón y teniendo en cuenta lo expuesto en la introducción hemos decidido utilizar los conceptos de *goal* y *soft-goal* en la determinación de la variabilidad de las líneas de producto.

La figura 1 muestra un modelo estratégico utilizado en el método i* [26]. Tanto los agentes como los recursos y en especial los *goals* (Communicate) y *soft-goals* (Usability, Mobility) son elementos fuente de variabilidad. Las figuras 2 y 3 muestran parcialmente los árboles de variabilidad empleados.

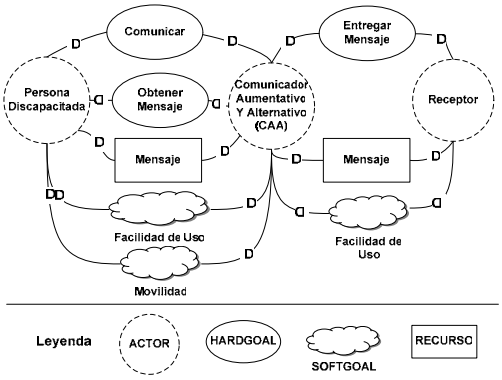

Figura 1. Modelo i* para un comunicador

Merece la pena destacar que los *goals* y *soft-goals* están muy relacionados: una selección de ciertos *soft-goals* implica que la forma óptima de alcanzar las metas va a ser distinta (por ejemplo, la combinación de los *subgoals* de la *soft-goal* "Accesibilidad": "poca precisión" pero sin "deficiencia mental" influye en la decisión de seleccionar el *goal* "entrada por palabras"). Este tipo de relación se conoce como *contribución* y proporciona un mecanismo de ayuda para la toma de decisiones sobre la combinación óptima de *goals* (formas de alcanzar los requisitos funcionales) cuando se imponen ciertas restricciones por medio de los *soft-goals* o requisitos no funcionales.

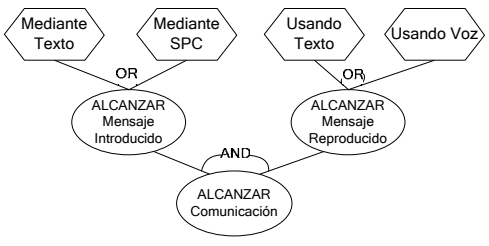

Figura 2. Grafo de variabilidad de *goals*

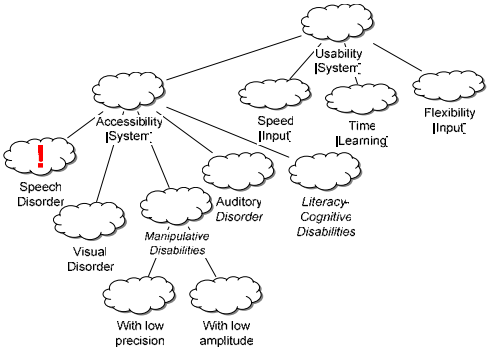

Figura 3. Grafo de variabilidad de *soft-goals*

La ventaja de este enfoque se pone mucho mejor de manifiesto si se dispone de una herramienta que soporte los cálculos necesarios. Se ha construido un prototipo que presenta gráficamente al analista los resultados de aplicar las contribuciones a las metas generales deseados [11].

A partir de estas metas, siempre y cuando se haya registrado cuidadosamente la trazabilidad, se puede obtener el sub-grafo de *features* o el modelo inicial de un producto concreto dentro de la línea de producto. La figura 4 muestra el esquema de dependencias de los modelos manejados. La parte de la propuesta relacionada con los modelos de *goals* y *soft-goals* y sus dependencias ha sido desarrollada en otros trabajos [11], [10]. En la siguiente sección se analiza el papel que puede jugar MDA para completar este enfoque de elicitación y especificación de requisitos para líneas de producto y su conexión con los modelos de *features* y la arquitectura del dominio.

3. MDA e Ingeniería de Requisitos para Líneas de Producto

OMG da cuenta de varias experiencias de éxito en la aplicación de MDA. La pregunta pertinente en lo que respecta a la aplicación de MDA al desarrollo de líneas de producto es si se dispone del suficiente grado de libertad para representar todas las posibles variaciones con independencia no sólo de plataformas sino de recursos, metas no funcionales, etc. Los ejemplos que aparecen reflejados en diversas fuentes parecen contradecir esta necesidad. Por ejemplo, en el libro de Kleppe et al. [16] se muestra la traducción de un PIM en tres PSM complementarios, basados en soluciones predefinidas: tecnología Web, java Beans y una base de datos relacional. Otros ejemplos, como los propuestos por Mellor y Balzer en su paradigma de UML ejecutable [20] se refieren a un tipo específico de sistemas y requieren una definición muy precisa del comportamiento del sistema utilizando un lenguaje de semántica de acciones muy cercano al código convencional.

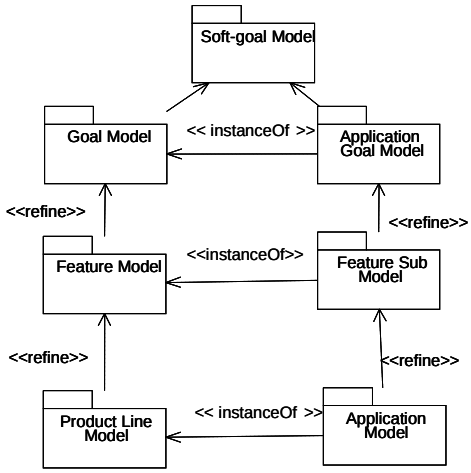

Figura 4. Modelos en el desarrollo de líneas de producto y sus dependencias

Sin embargo a la hora de especificar un sistema (o una línea de producto en nuestro caso) es preciso documentar requisitos de calidad, fiabilidad, etc. Estos requisitos no funcionales influirán de forma decisiva en la solución que se debe desarrollar. La impresión es que en muchos ejemplos la existencia de los requisitos no funcionales está oculta y no aparece de forma explícita en el PIM. De todas formas, merece la pena analizar las posibilidades que la idea de MDA puede aportar a nuestro campo de estudio. En el fondo, lo que estamos propugnando es la derivación (idealmente de forma automática) de un producto específico en una línea de producto software a partir de una serie de modelos previos. En concreto queremos llegar a la variante óptima para un conjunto de requisitos funcionales y no funcionales a través de la trazabilidad entre los

modelos de *goals*/*features*/arquitectura. La transformación automática entre esos modelos representaría una mejora considerable en relación con la forma de trabajo convencional en líneas de producto.

La disciplina de *Requisitos para Líneas de Producto* incluye varias actividades fundamentales. La principal de ellas es la especificación del modelo del dominio con inclusión de los requisitos de variabilidad. El diseño de una solución para esos requisitos constituye la arquitectura de la línea de producto, típicamente expresada como un *framework* orientado a objetos.

Posteriormente, en el proceso de desarrollo de una aplicación concreta (miembro de la línea de producto), ésta debe derivarse a partir del *framework* del dominio. En este proceso, se deben seleccionar aquellas variantes que resultan apropiadas para los requisitos funcionales y no funcionales expresados por los usuarios. Esta actividad es esencialmente una transformación dirigida por la selección de *features* efectuada por el ingeniero de aplicación, originando un sub-modelo de *features* y que a su vez (por las relaciones de trazabilidad) generará el modelo arquitectónico inicial de la aplicación. Las variantes seleccionadas lo son en función de los requisitos particulares de la aplicación, que deben ser tenidas en cuenta por el ingeniero.

La novedad que propone MDA es la posibilidad de automatizar (al menos parcialmente) la transformación que especifica cómo se convierten las instancias del modelo de *features* en una aplicación ejecutable. En su estado más avanzado la transformación sería equivalente a la compilación de un modelo descrito por un lenguaje específico de dominio. El requisito previo antes de evaluar esta posibilidad es disponer de un meta-modelo que represente de modo uniforme los conceptos de cada una de las técnicas utilizadas. En la figura 5 se puede apreciar el correspondiente a los modelos de *goals* y *soft-goals* y más adelante se discute el meta-modelo de *features*.

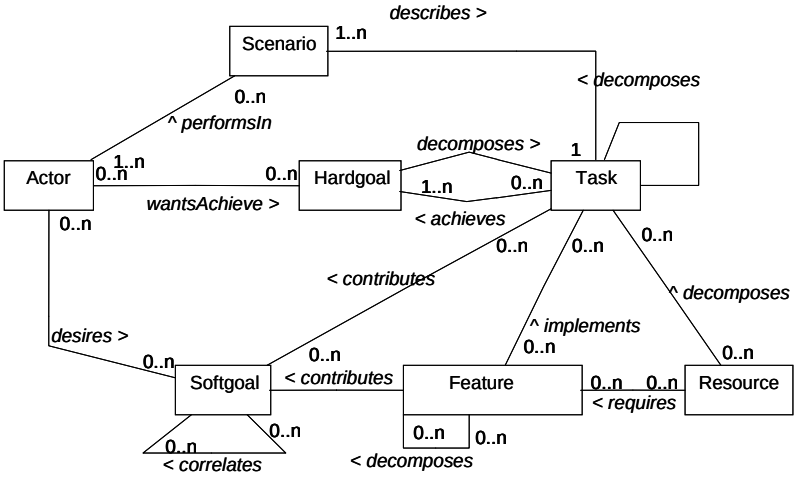

Figura 5. Meta-modelo de *goals* y *soft-goals*

En [8] MDA se utiliza como un medio para derivar productos específicos para una plataforma a partir de un PIM de dominio para un caso particular de líneas de producto, las familias de productos configurables. En este caso el punto de variación que se instancia es la plataforma de destino del sistema. La principal ventaja de la aplicación de MDA en este contexto es que la variabilidad de plataformas puede separarse del modelo del dominio e integrarse en la

transformación en sí. Aunque la plataforma evidentemente es un punto de variabilidad, en el modelo del dominio existen otros muchos más a tener en cuenta incluyendo requisitos funcionales y no funcionales.

La cuestión fundamental es si MDA puede gestionar las diversas alternativas si se añade información sobre la variabilidad utilizando trazabilidad entre las metas (*goals* y *soft-goals*), las *features* que los recogen y los puntos de variación en el *framework* que representa la arquitectura del dominio. Si eso es así, seleccionando el conjunto de *goals* y *soft-goals* en función de las necesidades de una aplicación concreta se podría obtener una configuración de *features* que guiaría de forma automática la instanciación del *framework* de dominio en un modelo de aplicación. Finalmente el modelo de la aplicación podría someterse a una transformación convencional PIM a PSM. La figura 6 muestra el esquema general. En esa figura puede apreciarse que los únicos modelos específicos de plataforma serán las aplicaciones concretas para EJB o .NET. El resto son realmente modelos independientes de la plataforma. El objetivo de este trabajo es precisamente examinar que tipo de transformaciones son plausibles entre estos PIMs y en qué grado son automatizables.

Existen claramente dos tipos de transformaciones:

- Horizontales: selección de *goals/soft-goals*, configuración de un modelo de *features* e instanciación de la arquitectura del dominio.
- Verticales: transformación del modelo de *goals/soft-goals* de la línea de producto en un modelo de *features* y de éste en un modelo arquitectónico inicial para la línea de producto.

También son verticales las transformaciones equivalentes a las anteriores para cada aplicación concreta pero, teniendo en cuenta la forma de instanciar el modelo de aplicación utilizando *goals* y *features* como guías, el camino marcado en la parte derecha de la figura no va a ser transitado directamente: la selección de un subconjunto de *goals* y *soft-goals* implica un subconjunto de *features* que a su vez define por trazabilidad una instanciación del *framework* en el modelo de aplicación.

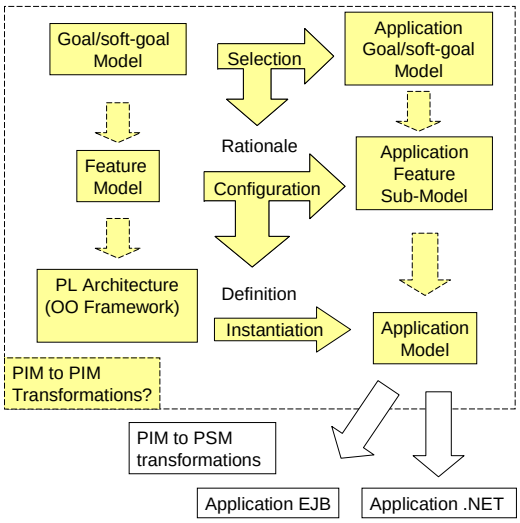

Figura 6. Ingeniería de Línea de Producto y MDA. La línea de puntos limita el estudio

Algunos modelos de *features* incorporan información sobre tecnología, plataformas, etc. Sin embargo, puesto que en nuestro caso este tipo de requisitos se incorporan a los *soft-goals*, los modelos de *features* son básicamente funcionales. Esto representa una ventaja adicional puesto que permite separar los aspectos funcionales de los no funcionales y estos últimos en varios submodelos independientes cumpliendo con las recomendaciones del paradigma de separación de aspectos [15].

3.1. Transformaciones horizontales

Analizando la figura 6 desde el punto de vista de las relaciones horizontales se pueden extraer varias conclusiones. En los procesos convencionales, la configuración de un modelo de *features* para obtener la combinación idónea de las mismas para un producto software concreto es un paso que requiere la habilidad de un experto del dominio puesto que es el único que tiene la experiencia para decidir porqué unas *features* se adaptan mejor que otras para resolver un problema particular. Existen herramientas (desde asistentes hasta lenguajes gráficos) que sirven para definir esta configuración pero no ayudan en la toma de decisiones. Lo que si es posible, cuando la

trazabilidad *features/framework* está almacenada en los modelos, es que las decisiones tomadas en el nivel de *features* se trasladen automáticamente a la instanciación del *framework* [25].

La ventaja de utilizar nuestro modelo complementario de *goals* y *soft-goals* es que nos da los motivos por los que determinada *feature* es mejor que otra (cumple con cierta funcionalidad y además es la mejor variante para cierta combinación de metas no funcionales priorizadas). En la práctica esto supone una elevación del nivel de abstracción desde las decisiones tomadas en el nivel de la solución a la toma de decisión en el nivel de los requisitos. En conclusión estas transformaciones son automatizables aunque no en el sentido habitual del paradigma MDA. De hecho disponemos ya de una herramienta que permite la selección del conjunto óptimo de *goals* necesarios en función de los deseos de los usuarios, expresados como un conjunto de *soft-goals* con distintas prioridades [11]. La conexión explícita con el modelo de *features* será la clave que permita automatizar la instanciación del *framework* de dominio para cada aplicación concreta.

3.2. Transformaciones verticales

La relación entre los modelos de *goals* y *features* para la línea de producto difícilmente puede ser considerada una transformación susceptible del enfoque MDA. La razón es que los objetivos (y en consecuencia también el modo de construcción) de ambos modelos son completamente diferentes. En el caso de los *goals* se pretende definir el conjunto de metas del sistema y los diversos modos de alcanzar esas metas (desde los puntos de vista funcional y no funcional). Sin embargo el objetivo de los modelos de *features* es separar la parte común de la parte variable de modo que sirvan de guía para la construcción del *framework* de dominio. Las tareas o *tasks* tienen una fuerte relación con las *features* puesto que éstas últimas las implementan pero en general la relación es de tipo muchos-a-muchos (ver figura 5), con lo que no es fácil derivar un modelo de otro. En consecuencia y hasta hoy, la construcción de ambos modelos debe hacerse de manera independiente aunque coordinada por parte del ingeniero experto en el dominio. Como hipótesis de trabajo pensamos que la introducción de la restricción "cada *task* sólo puede ser implementada por una *feature*" facilitará al menos la trazabilidad de los modelos (e incluso la obtención de un primer modelo de *features* atómicas).

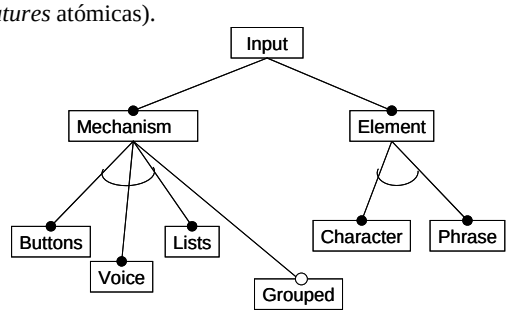

Figura 7. Modelo de *features* de un comunicador

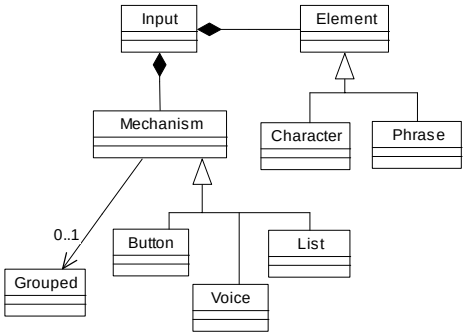

Figura 8. Una solución al problema de la figura 7 mediante un *framework* simple

La segunda transformación, modelo de *features* en *framework* si que encaja de forma natural en la filosofía MDA. En la figura 7 se puede ver un modelo que representa la parte de la entrada del menseaje de un comunicador para personas discapacitadas usando *features*, y en la siguiente un *framework* obtenido a partir de ese modelo utilizando especialización como método para introducir la variabilidad. Aunque existen varias formas de resolver un punto de variabilidad [6], se puede anotar cada uno de ellos con la técnica elegida de modo que la transformación se adapte a las preferencias del ingeniero de dominio.

La manera de definir una transformación es relacionar los conceptos del modelo de *features* con los conceptos del *framework* de dominio. Para ello se necesita una definición precisa del meta-modelo de cada uno de los lenguajes. En el caso del meta-modelo de destino la elección está clara: se debe utilizar UML, bien el formato original, bien alguna de las extensiones para la representación de *frameworks* que se han definido utilizando fundamentalmente estereotipos. Para las primeras experiencias que estamos llevando a cabo utilizamos básicamente un pequeño porcentaje del meta-modelo completo de UML 1.5. Realmente sólo necesitamos las meta-clases que aparecen en la figura 9 y que son las mismas que se manejan en XMI.

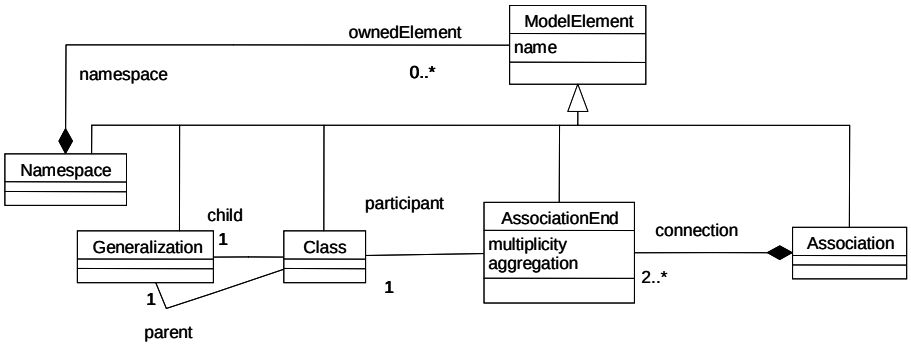

Figura 9. Meta-modelo de UML/XMI utilizado en este trabajo como subconjunto de UML1.5

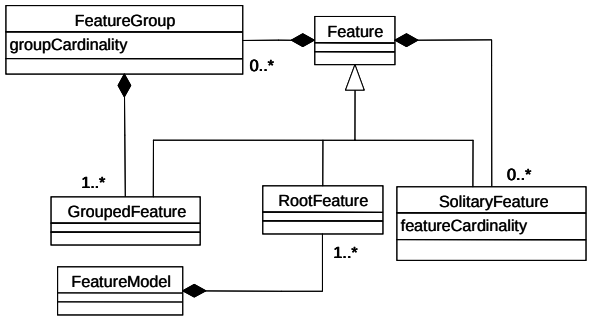

Figura 10. Meta-modelo de *features*, simplificado a partir de [7]

En cuanto al meta-modelo de *features* no existe un consenso ni mucho menos un estándar y sí bastantes propuestas. En los primeros análisis hemos observado que la elección del meta-modelo facilita o complica en gran manera la transformación que queremos definir. Por esa razón el meta-modelo elegido es el más adecuado para simplificar la transformación, concretamente el propuesto por Czarnecki et al. [7]. La característica fundamental es la importancia que otorga a las cardinalidades y que tienen traducción directa en las multiplicidades de los modelos UML. En la figura 10 se pueden observar los tres tipos de *features* que contempla: individuales, agrupadas y una especial que representa la raíz de cada árbol.

La transformación consiste en:

- Transformar el modelo de *features* en un paquete
- Transformar cada *feature* en una clase
- Si la *feature* es de tipo Solitary*Feature*, asociarla con la clase generada por la *feature* propietaria con una multiplicidad igual a la definida para la *feature* (*feature*Cardinality)
- Transformar cada *Feature*Group en una superclase del conjunto de *features* de tipo Grouped*Feature* de su propiedad. Además, asociar la nueva clase con la clase generada por la *feature* propietaria con una multiplicidad igual a la definida para el grupo (groupCardinality)

La estrategia se basa en los tres subtipos de *features*: se transforma la *feature* raíz del árbol y recursivamente cada *feature* asislada o grupo de *features* se transforma utilizando la cardinalidad del modelo para definir la multiplicidad de las asociaciones generadas. La presencia de un grupo implica una clase con tantos subtipos como *features* agrupadas. Las figura 11 muestra la definición de la transformación para el modelo y cada *feature* raíz y la figura 12 las transformaciones de un grupo de *features*. Para la definición de las transformaciones se ha utilizado la sintaxis de la última versión de la respuesta remitida a OMG sobre la solicitud Query/View/Transformation (QVT) [24]. La transformación en su estado actual permite obtener una primera versión de un *framework* de dominio. La transformación debe completarse con elementos de trazabilidad, de modo que aunque el *framework* se complete posteriormente con detalles de diseño, no se perderá en ningún momento la conexión entre la implementación y la especificación de la variabilidad.

En cuanto a la implementación, se manejan varias posibilidades. La más directa utiliza hojas de estilo XML o java para tratar documentos XML. La razón es que después de examinar varias herramientas disponibles, la mejor elección es utilizar X*Feature* (disponible en http://www.pnp-software.com/XFeature/) para describir los modelos de *features*. X*Feature* es una herramienta basada en XML y Eclipse que tiene como ventaja fundamental el hecho de que es totalmente configurable, incluyendo la definición del meta-modelo de *features* que se utiliza. En cuanto al resultado de la transformación lo más sencillo es generar modelos UML en formato XMI, a pesar de los problemas de compatibilidad de versiones. Estos documentos son legibles por cualquier herramienta CASE basada en UML.

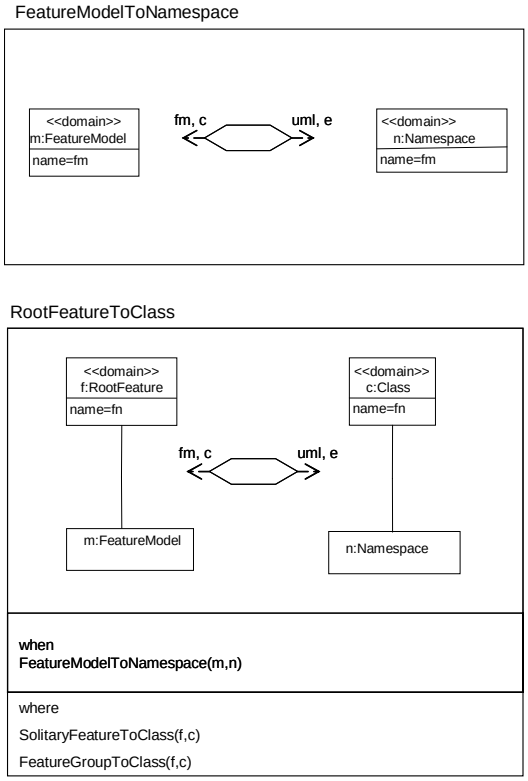

Figura 11. Transformación del modelo y de la raíz de cada árbol de *features*

Como trabajo futuro se plantea utilizar Eclipse Modeling Framework (EMF, disponible en http://www.eclipse.org/emf), también basado en XML y Eclipse. EMF es un *framework* basado en las ideas de MDA para construir herramientas de generación automática de código. Utiliza su propio meta-modelo que originalmente era una implementación simplificada de MOF/UML (Ecore). De este modo se enlazarían las transformaciones definidas en este trabajo con las propias de la intención original de MDA.

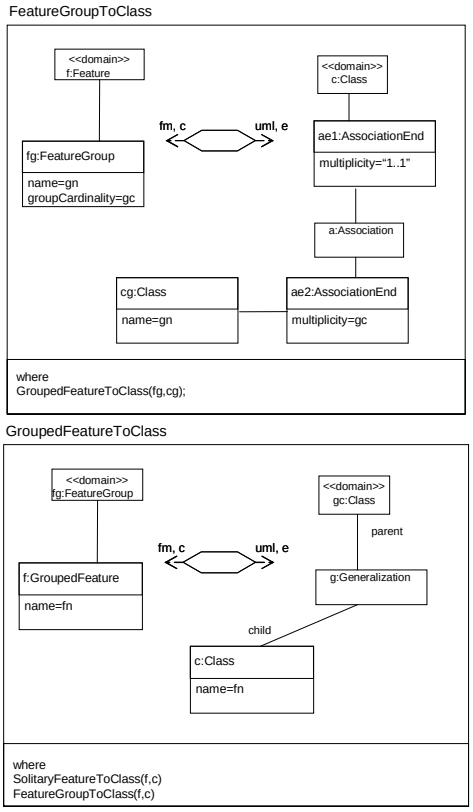

Figura 12. Transformación en dos pasos de un grupo de *features*

4. Conclusiones y trabajo futuro

En este artículo se exploran las posibilidades de incorporación de una visión MDA en el proceso de elicitación, análisis e implementación de la variabilidad (y su trazabilidad), todo ello en un contexto de desarrollo de líneas de producto software.

La solución se sustenta en varios pilares:

a) Separar los diferentes aspectos de la variabilidad utilizando *goals* y *soft-goals* para construir posteriormente el modelo de *features* de la línea de producto.
b) Transformar este conjunto de PIMs en un nuevo PIM que represente la arquitectura inicial de la línea de producto.
c) Utilizando *goals* y *soft-goals* (y una herramienta como la descrita en [11]), derivar el conjunto óptimo de *features* para un producto concreto.
d) Obtener el PIM para la arquitectura del producto concreto mediante instanciación de la arquitectura de la línea de producto, en función de las decisiones tomadas en el punto anterior.

En cuanto al trabajo inmediato, es preciso introducir de forma explicita la trazabilidad de los modelos. Para ello será necesario completar los meta-modelos utilizados e incluir esa información en los documentos XML manejados.

Agradecimientos

Este trabajo ha sido financiado por el MEC/fondos Feder (proyecto TIN2004-03145). El primer autor disfruta de una beca FPU financiada por el MEC.

Referencias

[1] Bass, L., Clements, P., Donohoe, P., McGregor, J. and Northrop, L. "*Fourth Product Line Practice Workshop Report*". Technical Report CMU/SEI-2000-TR-002 (ESC-TR-2000-002), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 2000.

[2] Bosch, J. "*Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach*". Addison-Wesley. 2000.

[3] Chastek, G., Donohoe, P., Kang, K. C., Thiel, S. "*Product Line Analysis: A Practical Introduction"*. Technical Report CMU/SEI-2001-TR-001 ESC-TR-2001-001, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213.

[4] Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers 2000.

[5] Clements, Paul C. and Northrop, Linda. *"Software Product Lines: Practices and Patterns"*. SEI Series in Software Engineering, Addison-Wesley. 2001.

[6] Krzystof Czarnecki and Ulrich W. Eisenecker, *"Generative Programming: Methods, Tools, and Applications*", Addison-Wesley, 2000

[7] K. Czarnecki, S. Helsen, and U. Eisenecker. *"Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models"*. To appear in Software Process Improvement and Practice, 10(2), 2005.

[8] Sybren Deelstra, Marco Sinnema, Jilles van Gurp, Jan Bosch, *"Model Driven Architecture as Approach to Manage Variability in Software Product Families"*, in Arend Rensink (Editor), Model Driven Architecture: Foundations and Applications, CTIT Technical Report TR–CTIT–03–27, University of Twente, avalaible in http://trese.cs.utwente.nl/mdafa2003

[9] García, F. J., Barras, J. A., Laguna, M.A., and Marqués, J. M. *"Product Line Variability Support by FORM and Mecano Model Integration*". In ACM Software Engineering Notes. 27(1);35-38. January 2002.

[10] Bruno González-Baixauli, Miguel A. Laguna, Julio Cesar Sampaio do Prado Leite, *"Análisis de Variabilidad con Modelos de Objetivos*". VII Workshop on Requirements Engineering (WER-2004). Anais do WER04, pp 77-87, 2004.

[11] González-Baixauli, B., Leite J.C.S.P., and Mylopoulos, J. *"Visual Variability Analysis with Goal Models*". Proc. of the RE'2004. Sept. 2004. Kyoto, Japan. IEEE Computer Society, 2004. pp: 198-207.

[12] Jacobson I., Griss M. and Jonsson P. *"Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison Wesley Longman. 1997.

[13] Kang, K. C., Kim, S., Lee, J. y Kim, K. *"FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*". Annals of Software Engineering, 5:143-168. 1998.

[14] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. *"Feature-Oriented Domain Analysis (FODA) Feasibility Study"*. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213

[15] G. Kiczalez, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, J. Irwin, *"Aspect Oriented Programming"*, in Proceedings of 11th European Conference on Object Oriented Programming, pp. 220-242, Springer Verlag, 1997.

[16] Anneke Kleppe, Jos Warmer, Wim Bast. *"MDA Explained: The Model Driven Architecture: Practice and Promise"*. Addison Wesley, 2003.

[17] Miguel A. Laguna, and Bruno González-Baixauli, *Goals* and MDA in Product Line Requirements Engineering, Technical Report GIRO-2005-01, available in http://giro.infor.uva.es/docpub/giro-05-01.pdf

[18] Miguel A. Laguna, Bruno González, Oscar López, F. J. García, *"Introducing Systematic Reuse in Mainstream Software Process"*, IEEE Proocedings of EUROMICRO'2003, Antalya, Turkey, pp: 351-358, 2003.

[19] Lee, K., Kang, K. C., Chae, W., Choi, B. W. *"Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse*". Software: Practice and Experience, 30(9):1025-1046. 2000.

[20] S.J. Mellor, M. J.Balcer, *"Executable UML A foundation for the Model-Driven Architecture*", Addison Wesley Professional, 2002

[21] J. Mylopoulos, L. Chung, and E. Yu. *"From object-oriented to goal-oriented requirements analysis"*. Communications of the ACM, 42(1):31–37, Jan. 1999.

[22] Object Management Group, *"MDA Guide Version 1.0"*, 2003

[23] Object Management Group, *"Software Process Engineering Metamodel"*. Available http://www.omg.org/technology/documents/formal/spem.htm.

[24] Object Management Group and *QVT-Merge Group, "Revised submission for MOF 2.0 Query/View/Transformation version 2.0"* Object Management Group doc. *ad/2005-03-02, 2005.*

[25] Oliveira, T. C., Alencar, P., Mathias, I. F., Cowan, D. D., Lucena, C. J. P., *"Enabling Model Driven Product Line Architectures"*, In: Second European Workshop on Model Driven Architecture (MDA), Proceedings of 2nd EWMDA, Canterbury, UK, 2004.

[26] E. S. K. Yu and J. Mylopoulos. *"From E-R to A-R – modelling strategic actor relationships for business process reengineering"*. Int. Journal of Intelligent and Cooperative Information Systems, 4(2–3):125–144, 1995.