Gestión de la Variabilidad en Líneas de Productos

Miguel A. Laguna
Universidad de Valladolid, Departamento de Informática
Campus M. Delibes, 47011 Valladolid, Spain
mlaguna@infor.uva.es

Bruno González-Baixauli
Universidad de Valladolid, Departamento de Informática
Campus M. Delibes, 47011 Valladolid, Spain
bbaixauli@infor.uva.es

Oscar López
Instituto Tecnológico de Costa Rica,
Campus Regional de San Carlos, Costa Rica
olopez@ic-itcr.ac.cr

Abstract

The study of the variability in software systems has become a primary aspect of software development, in particular in the fields of software customization or software product lines. The customary way to manage the variability in product lines is by means of feature models that also allow the configuration of each specific application within the product line to be selected. However the control of traceability between the features and the architectural models (generally based on UML) is not simple. The fundamental reasons are the complexity of the traceability relationships, and the fact that the same modeling mechanisms (e.g., the specialization in a class diagram) serve to express the variation points in the models that represent the product line and in each specific application. The aim of this work is to use the *package merge* mechanism of the UML 2 infrastructure meta-model as the representation (and support for the configuration) of the variability in the product line while reserving the classical mechanisms (the specialization in the structural models, the <<extends>> relationship in the use case models, etc.) to express the variants in execution time of each specific application. The structure of the feature models is directly reflected in the relationships between packages in the architectural models, so the traceability of the configuration decisions is straightforward. The implementation of the package merge mechanism uses the facility of partial classes of languages such as C#.
Keywords: software product line, feature model, variability, traceability

Resumen

El interés por el estudio de la variabilidad en el desarrollo del software ha aumentado significativamente durante los últimos años. Esto se debe a su interés en diversos campos, desde la personalización del software a las líneas de productos. La forma más común de gestionar la variabilidad en líneas de productos software es mediante modelos de características o *features* que permiten además seleccionar la configuración de cada aplicación concreta dentro de una línea de productos. Sin embargo la trazabilidad entre los modelos de características y los modelos de diseño (generalmente basados en UML) no es sencilla. La propuesta de este trabajo consiste en utilizar el mecanismo de combinación de paquetes ("package merge") de UML 2 como herramienta de representación y configuración de la variabilidad de la línea de productos y reservar los mecanismos clásicos de modelado (la especialización de los modelos estructurales o la relación <<extiende>> de los modelos de casos de uso) para expresar las variantes válidas en tiempo de ejecución de cada aplicación concreta. La estructura de los modelos de características se refleja directamente en las relaciones entre paquetes del modelo arquitectónico de modo que la trazabilidad de las decisiones de configuración es automática. La forma de implementación que hemos elegido se basa en la utilización de clases parciales, disponibles en lenguajes como C#.
Palabras clave: línea de productos software, modelos de características, variabilidad, trazabilidad

1. Introducción

Durante los últimos años los sistemas software cada vez tienden a soportar una mayor variabilidad, entendida como la habilidad de cambio o de personalización de un sistema. Esta variabilidad se debe a las demandas de los usuarios de sistemas más adaptables a sus necesidades (sistemas personalizables) y, sobre todo, a la presión del mercado, que hace más rentable fabricar líneas de productos software para reutilizar la mayor parte del esfuerzo de desarrollo [2]. Sin embargo, el enfoque de líneas de productos es complejo y representa un gran salto para las empresas que pretenden abordarlo [1]. Nuestro trabajo trata de simplificar el paso de un modelo de desarrollo convencional a otro que aproveche las ventajas de las líneas de productos. Para ello, entre otras iniciativas, hemos propuesto una adaptación del *Proceso Unificado* de desarrollo [16] para recoger las técnicas específicas de *Ingeniería de Línea de Productos* en un proceso paralelo al de *Ingeniería de Aplicaciones,* como se esquematiza en la Figura 1.

La disciplina de *Ingeniería de Requisitos para Líneas de Productos* es clave en este tipo de desarrollo y por ello es preciso mejorar la elicitación, representación y gestión de requisitos con especial hincapié en los aspectos de variabilidad y trazabilidad. El problema tiene dos aspectos principales: por un lado, la definición del modelo que represente la línea de productos y sus variaciones (y permita configurar el conjunto óptimo de las mismas para cada aplicación concreta) y, por otro, el registro de la trazabilidad entre las variaciones del modelo y el reflejo de las mismas en la arquitectura que implementa la línea de productos (para facilitar la derivación de cada aplicación concreta). En nuestro caso, utilizamos modelos de metas y de características para definir la variabilidad de la línea de productos. Existe un amplio consenso en torno a la necesidad de expresar la variabilidad mediante modelos de *features* (que traducimos por características) en alguna de sus múltiples versiones como FODA [15] o FORM [14]. Una *feature* es una "característica relevante para algún interesado y que representa un aspecto común o variable de un producto software" [6]. Las características se organizan en una jerarquía que representa tanto la parte común como las variaciones de una familia de productos software. La técnica tiene numerosas aplicaciones y puede ser aplicada en distintas fases; por ejemplo, se utiliza para representar el resultado del análisis de dominio o como herramienta de configuración de una aplicación concreta dentro de la línea de productos [5]. Como alternativa, los casos de uso son bien conocidos y fueron propuestos por Jacobson como base para su método [12]. Sin embargo ambas técnicas tienen en común un punto débil: están enfocadas más a la solución que al problema. Por esta razón hemos propuesto utilizar los conceptos de metas (*goals* y *soft-goals*) para el análisis de la variabilidad de una línea de productos [9], incluyendo una herramienta que permite la selección del conjunto óptimo de características en función de los deseos de los usuarios, expresados como un conjunto de metas con distintas prioridades [10]. El segundo aspecto del problema aborda las relaciones de trazabilidad entre el modelo de características y el diseño de la arquitectura de la línea de productos (generalmente expresada como un *framework* orientado a objetos). Esta conexión es la clave que permite automatizar la instanciación del *framework* para cada aplicación concreta, según muestra el esquema de la Figura 2.

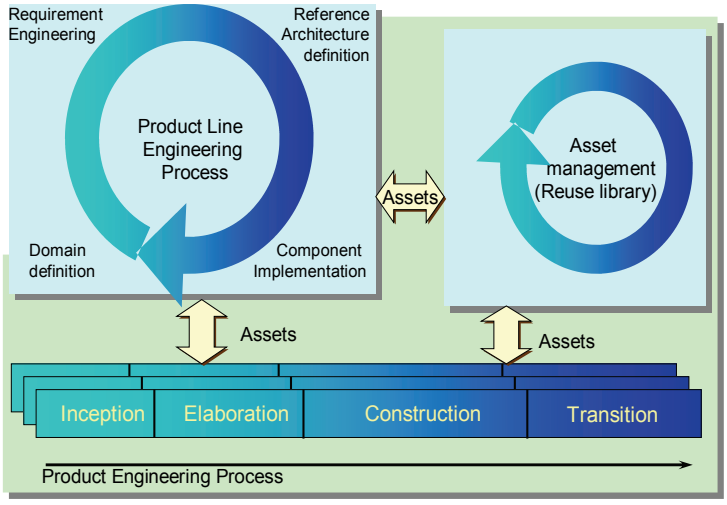


Figura 1. Procesos de *Ingeniería de Línea de Productos* y de *Ingeniería de Aplicaciones*. Un tercer proceso, *Gestión de Activos,* completa el esquema de desarrollo de líneas de productos basado en UP

Sin embargo esa trazabilidad no es fácil de establecer por varias razones: por un lado, una característica opcional puede originar varios elementos en los modelos de diseño y viceversa. Por otro lado, los mismos mecanismos de modelado se utilizan para expresar las variaciones de la línea de productos y de cada aplicación concreta. La solución a estos problemas se ha abordado modificando o adaptando UML, es decir alejándose del estándar, lo que supone una barrera de entrada para las organizaciones que desean adoptar este tipo de desarrollo. Desde nuestro punto de vista, los modelos de diseño de la línea de productos deben mantenerse dentro del estándar. Para lograr este objetivo proponemos en utilizar el mecanismo de combinación de paquetes o *package merge* de UML 2 [20]. Esta solución permite representar de forma ortogonal las variaciones arquitectónicas de la línea de productos, su relación directa con las características opcionales e incluso, utilizando paquetes de clases parciales, con la estructura del código que implementa el diseño. De esa forma se cubre todo el ciclo de vida de la línea de productos en consonancia con el esquema de la Figura 2.

En el resto del artículo se analizan distintas soluciones propuestas, sus debilidades y las ventajas de nuestro enfoque. En la siguiente Sección, se analiza la relación entre los modelos de características y diseño y se establece un conjunto de requisitos que debe cumplir una solución óptima. La Sección 3 presenta la propuesta y sus ventajas y la Sección 4 extiende la idea a todo el ciclo de vida de desarrollo de una línea de productos y sus miembros. La Sección 5 resume los trabajos relacionados y, finalmente, la Sección 6 concluye el trabajo y plantea el trabajo futuro.

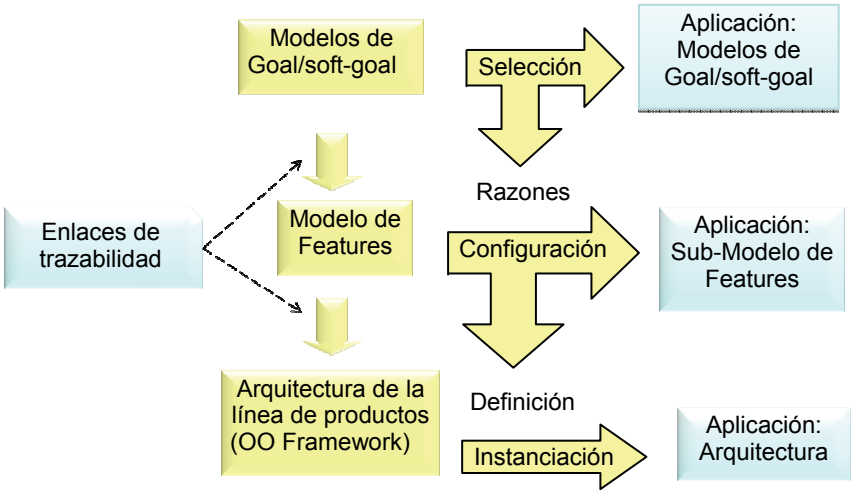


Figura 2. Relaciones entre los modelos de una línea de productos

2. Requisitos de trazabilidad

En el enfoque de desarrollo de línea de productos, cada aplicación concreta se deriva del *framework* que implementa la arquitectura de la línea de productos. 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 selección de características efectuada por el ingeniero de aplicación, generando un sub-modelo de características, que a su vez (por las relaciones de trazabilidad) generará por derivación toda o la mayor parte del código de la aplicación. Como se ha mencionado en la introducción, la clave reside en la trazabilidad desde las metas/características hasta el código pasando por los modelos de diseño.

Sin embargo, la trazabilidad no es fácil de gestionar por varias razones. En primer lugar, una característica opcional puede originar varios elementos en un modelo de diseño (por ejemplo, seleccionar una variante "pago por domiciliación bancaria" implica, al menos, algún atributo o asociación que represente una "cuenta corriente" asociada con el comprador y el concepto de "orden de cobro al banco" que deben reflejarse en el modelo estructural, además de varios casos de uso y los diagramas de secuencia relacionados). Es decir, tenemos que asignar la relación de trazabilidad entre elementos de los dos niveles con una multiplicidad uno a varios, y lo mismo en sentido contrario (en realidad y desde el punto de las relaciones tendríamos una estructura de híper-grafos), lo que complica rápidamente el modelo global haciendo que sea muy poco escalable.

El segundo problema tiene que ver con el hecho de que los mecanismos básicos de modelado de la variabilidad (la especialización en los diagramas de clases o la relación <<extend>> de los casos de uso) se

utilizan en muchas ocasiones para expresar dos niveles de variabilidad distinta: el diseño de la arquitectura de la línea de productos (que se corresponde con requisitos opcionales) y el diseño de una aplicación concreta, que sigue teniendo variaciones en tiempo de ejecución (por ejemplo, dos formas de pago alternativas). En el ejemplo típico de una línea de productos de comercio electrónico, el mecanismo de extensión de los casos de uso permite mostrar distintas formas de pago. Ahora bien, en una solución concreta todas las formas de pago pueden ser válidas en tiempo de ejecución y por tanto estarán presentes. Pero es posible que otras aplicaciones de la línea de productos admitan sólo determinadas formas de pago. El problema está en que el mismo mecanismo sirve para mostrar variaciones en tiempo de ejecución y en tiempo de configuración. Sin embargo, en el trabajo pionero de Jacobson [12], no hay diferencia fundamental entre la representación de la variabilidad en ambos niveles: por ejemplo la relación <<extend>> de los casos de uso es uno de los pilares con los que construye su método. En resumen, este enfoque tiene como problema principal la falta de separación entre la variabilidad global de la línea de productos y la variabilidad residual de cada aplicación concreta. El mismo razonamiento puede extenderse a los diagramas estructurales, donde la especialización, las asociaciones con multiplicidad opcional (0..1, 0..*), etc. se pueden utilizar en los dos niveles lo que introduce ambigüedad en los modelos de la línea de producto.

Las soluciones más recientes propuestas para mostrar la variabilidad de una línea de productos con UML pasan por modificar o anotar los modelos, tanto estructurales como funcionales o incluso dinámicos (en la Sección 5 se resumen las propuestas más conocidas). De este modo se resuelve el segundo problema, pero raramente se ataca el primero (múltiples dependencias entre elementos UML y características). Además, es preciso cambiar el meta-modelo de UML o al menos extenderlo mediante estereotipos, introduciendo nuevas notaciones (ejemplos representativos son los trabajos de Gomaa [8] o John y Muthig [13] comentados en la Sección de trabajos relacionados). La propuesta más reciente de Czarnecki [5] sí permite que cada característica opcional se refleje en una o varias partes de un diagrama, pero de nuevo es necesario introducir elementos auxiliares en el meta-modelo de UML.

Como resultado del estudio de los puntos fuertes y débiles de estas propuestas, podemos establecer un conjunto mínimo de requisitos que debe cumplir una técnica útil de representación, y gestión de la variabilidad en el nivel de diseño de una línea de productos software:

1. Localizar en un solo punto del modelo de diseño todas las variaciones que origina cada característica opcional, de forma que se mantenga una correspondencia uno a uno y se facilite la gestión de la trazabilidad
2. Separar la variabilidad originada en el nivel de la línea de productos de la variabilidad intrínseca en el nivel de las aplicaciones concretas, eliminando ambigüedades
3. Mantener inalterado el meta-modelo de UML para eliminar la barrera de entrada a este paradigma para cualquier desarrollador además de permitir el uso de herramientas CASE convencionales
4. Conectar con los modelos de imple-mentación para acercarnos al ideal de desarrollo sin costuras o *"seamlesss development"* [22], propugnado por el paradigma de orientación a objetos pero desechado muchas veces por irrealizable

En la siguiente Sección se expone la solución que estamos utilizando en el desarrollo de líneas de productos para PYMES, basada en el mecanismo de composición de paquetes de UML 2 y que cumple con los requisitos anteriores.

3. Representación de la variabilidad mediante combinación de paquetes

Para alcanzar los objetivos expuestos en la Sección anterior, nuestra propuesta aboga por expresar la variabilidad en los modelos UML utilizando el concepto de combinación de paquetes (o "*package merge"*), presente en el meta-modelo de infraestructura de UML 2 y utilizado de forma exhaustiva en la definición misma de UML 2 [20].

El mecanismo "*package merge*" consiste fundamentalmente en añadir detalles de forma incremental (ver Figura 3, extraída del documento de oficial de UML 2). Según la especificación UML 2, se define como una relación entre dos paquetes que indica que los contenidos de ambos se combinan. Es similar a la generalización y se utiliza cuando elementos en distintos paquetes tienen el mismo nombre y representan el mismo concepto, comenzando por una base común. Dicho concepto se extiende incrementalmente en cada paquete añadido. Seleccionando los paquetes deseados es posible obtener una definición a la medida de entre todas las posibles. Aunque en este trabajo nos interesan sobre todo los diagramas de clases, el mecanismo se puede extender a cualquier diagrama de UML, en particular los de casos de uso y los de secuencia.

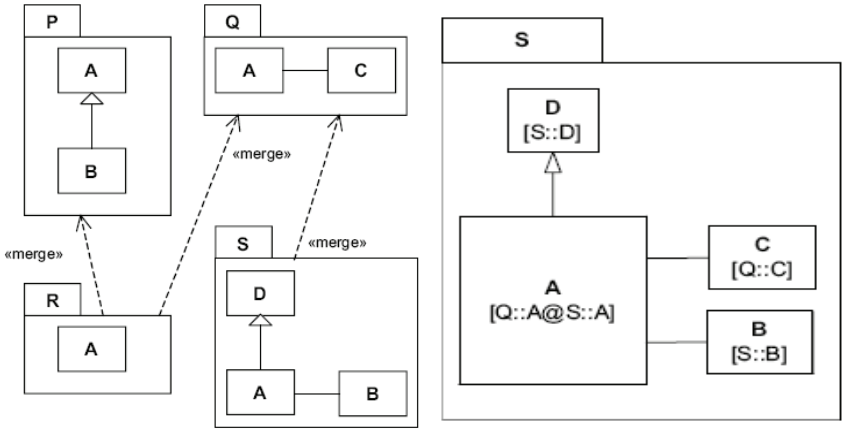


Figura 3. Ejemplo del mecanismo "package merge" extraído de la especificación del meta- modelo de la infraestructura de UML2 [20].

En el ejemplo de la Figura 2, el paquete S incluye las clases A, C, D, y B, con la peculiaridad de que A es una especialización de S::D y además incluye los detalles de Q::A representado por (Q::A@S::A). Evidentemente, las reglas que establece la especificación UML 2 son muy estrictas para evitar inconsistencias. Por ejemplo, no puede haber ciclos, las multiplicidades resultantes son las menos restrictivas de entre las posibles, o las operaciones deben conformar en número, orden y tipo de parámetros.

Además de clases, cualquier otro meta-tipo del meta-modelo de UML se puede utilizar con esta técnica. La semántica está definida en la especificación para los meta-tipos más comunes pero existen reglas generales aplicables al resto. La filosofía general es que el elemento resultante tiene que ser al menos tan capaz como el original antes de la combinación [20].

Este mecanismo nos permite establecer una trazabilidad clara entre los modelos de características y los artefactos UML. La aplicación a nuestro problema consiste en establecer, en primer lugar y para cada modelo UML del diseño, un paquete base que recoge la parte común de la línea de productos. Este paquete base se puede organizar del modo habitual (utilizando la descomposición recursiva en paquetes y las relaciones de <<import>> o <<access>> entre ellos). A este paquete base se añade un paquete por cada característica opcional, de modo que queden localizadas en ese paquete todas las modificaciones necesarias en el modelo de diseño asociadas a la característica. El paquete añadido se conecta mediante la relación <<merge>> con su paquete base en el punto preciso de la jerarquía de paquetes. La representación de los paquetes no es únicamente gráfica: también es una vista lógica de los componentes software y de su organización. En la Figura 4 se aprecia la estructura de paquetes de clases que refleja un modelo de características mostrado parcialmente, correspondiente a una línea de productos de comercio electrónico, adaptado de Lau [17] que utilizamos como caso de estudio en el resto del artículo.

Un catálogo, obligatorio en la línea de productos, tiene opcionalmente una estructura de categorías básica (Paquete *Categories*). Si está presente, cada categoría tendrá al menos una descripción y cada producto tendrá como mucho una categoría asociada. Añadir, por ejemplo, la característica opcional *CategoriesMultilevel* introduce una asociación reflexiva a la clase *Category* tal y como se muestra en el paquete *CategoriesMultilevel*. Al aplicar el mecanismo de combinación, explicado en la Sección anterior, el resultado de combinar estos tres paquetes es que aparece una clase final *Category* con el atributo *description* y tres asociaciones con las clases *Product*, *Catalog* y con ella misma.

En el caso del paquete *MultipleClassification* se aprecia una de la ventajas de la técnica: en la asociación *Category*/*Product* se sustituye automáticamente la multiplicidad más restringida [0..1] por la más abierta pero compatible [0..*] haciendo extensible el modelo. De esta forma, tres de los cuatro objetivos propuestos en la Sección anterior se pueden considerar alcanzados:

1. Se localizan en un solo paquete del modelo todas las variaciones que origina cada característica opcional manteniendo una correspondencia uno a uno
2. Se separan claramente los dos tipos de variabilidad, eliminando las ambigüedades. La variabilidad de la línea de producto viene dada por la selección paquetes. La variabilidad de aplicaciones concretas está dentro de cada paquete.
3. Se mantiene sin cambios el meta-modelo de UML

Para cubrir el último objetivo (desarrollo *sin costuras*) es preciso una visión más amplia que se aborda en la siguiente Sección.

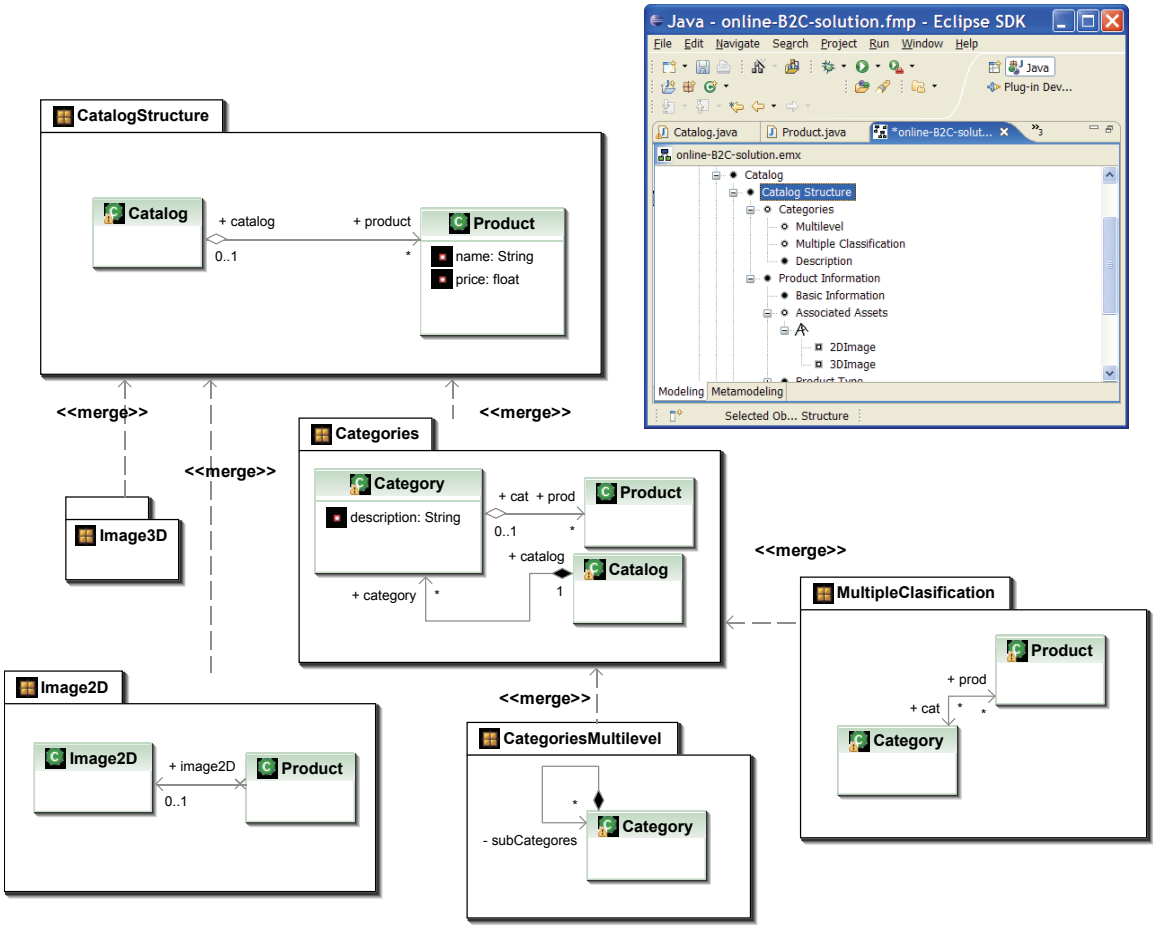


Figura 4. Modelo arquitectónico de una línea de productos de comercio electrónico que refleja un modelo parcial de características

4. Una visión global del desarrollo de líneas de productos

En esta Sección se presenta una visión global de la extensión del mecanismo de combinación de paquetes a todo el ciclo de vida de una línea de productos. Las experiencias se han llevado a cabo inicialmente utilizando un caso de estudio sobre comercio electrónico, descrito en la literatura desde distintos puntos de vista. En particular, sus características están descritas en [17], donde se realiza un estudio de dominio completo utilizando modelos de características y diagramas de clases y de actividades. Para extender la trazabilidad hasta los modelos de implementación utilizamos el concepto de clases parciales. La utilización de *"mixins"* o clases parciales fue propuesta originalmente en el lenguaje Flavors [18] y representa una alternativa a la herencia múltiple y una manera de manejar la variabilidad relacionada con aspectos. La intención es mantener la correspondencia uno a uno no sólo entre características y paquetes de diseño sino también con la estructura del código.

En nuestro modelo de desarrollo, utilizamos el mecanismo de clases parciales de C# para implementar la línea de productos dentro de la estructura de soluciones y paquetes de la plataforma MS Visual Studio 2005. Si el *framework* que implementa la arquitectura de la línea de productos está organizado en paquetes de clases parciales (un paquete base y tantos paquetes auxiliares como variaciones existen), para derivar una aplicación concreta basta con importar o referenciar los paquetes que se correspondan directamente con la configuración elegida en el modelo de características. La Figura 5 muestra una parte de la línea de producto de la Figura 4, implementada utilizando paquetes de C#. Estos paquetes opcionales pueden añadirse o no al proyecto que representa una aplicación concreta de la línea de producto utilizando los ficheros de configuración del compilador. En la misma figura se puede apreciar parte del fichero de configuración. De esta manera se cubre el objetivo 4 de la Sección 2, estableciendo la trazabilidad uno-a-uno desde las metas/características hasta el código. La escalabilidad está bajo estudio en líneas de productos piloto de PYMES con las que colaboramos.

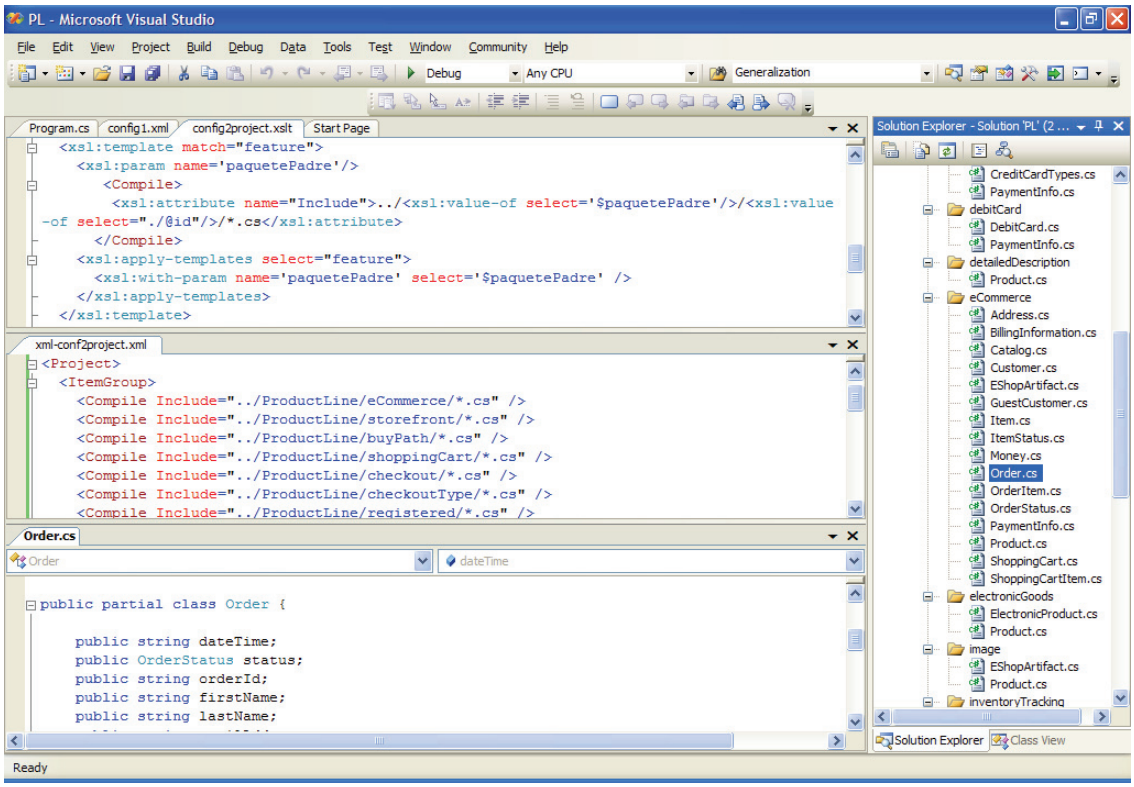


Figura 5. Parte de una línea de productos de comercio electrónico implementada en C# mediante clases parciales y hoja de estilo que genera la combinación de paquetes para una configuración concreta

Desde el punto de vista práctico, utilizamos una combinación de herramientas, algunas basadas en Eclipse como el plug-in fmp del grupo de Czarnecki [7], y otras de Microsoft. La conexión entre ellas es posible utilizando ficheros XML y transformaciones con hojas de estilo XSTL (esquema de la figura 6). Los modelos de características se pueden convertir en la estructura de paquetes de una línea de productos (obtenida como un fichero XMI, que se puede importar a una herramienta CASE convencional, Figura 7). Por su parte, una configuración concreta de un producto pude generar el fichero de configuración necesario para el compilador (parte inferior de la figura 7). Entre el trabajo pendiente se plantea la creación de un lenguaje específico de dominio que permita describir y configurar los modelos de características y conecte con los modelos de metas de la herramienta descrita en [10], los modelos de UML y la configuración de paquetes C# en MS Visual Studio.

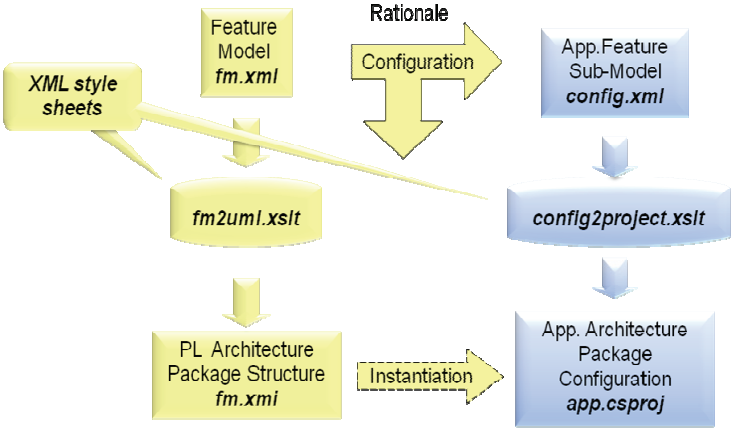
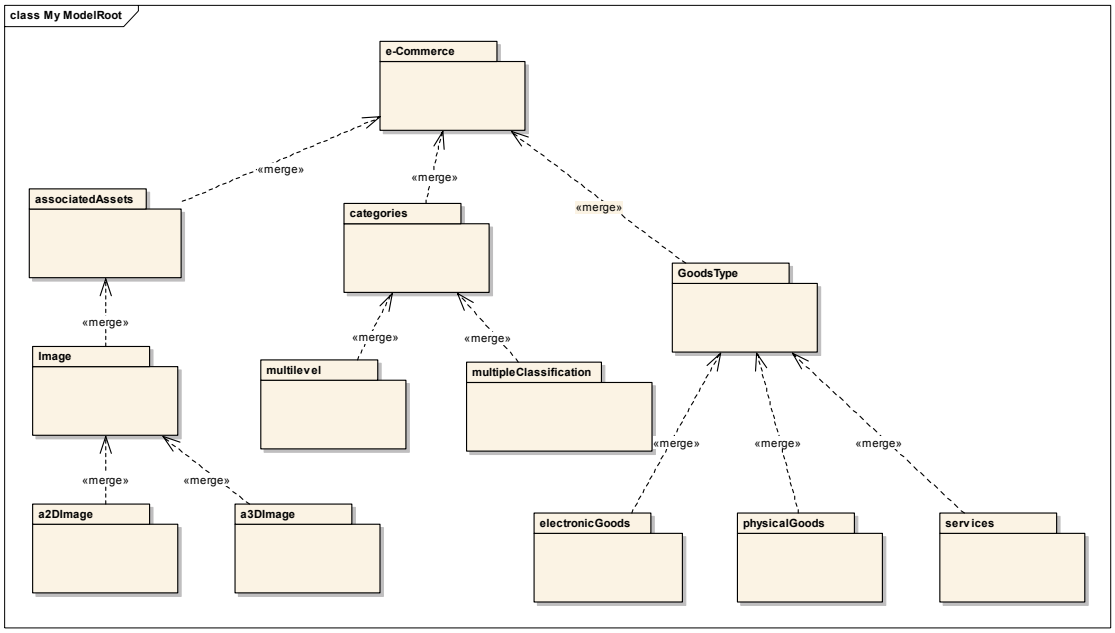


Figura 6. Esquema de transformaciones que permiten generar la estructura de paquetes de una línea de productos y la configuración de paquetes incluidos en un proyecto concreto.




```
<ItemGroup>
<Compile Include="../ModelRoot/eCommerce/*.cs" />
    <Compile Include="../ModelRoot/electronicGoods/*.cs" />
    <Compile Include="../ModelRoot/categories/*.cs" />
    <Compile Include="../ModelRoot/multipleClassification/*.cs" />
    <Compile Include="../ModelRoot/multilevel/*.cs" />
    <Compile Include="../ModelRoot/physicalGoods/*.cs" />
</ItemGroup>
```

Figura 7. Resultado parcial de la transformación del modelo de características de una línea de productos (obtenido como un fichero XMI, importado mediante una herramienta CASE convencional) y fichero de configuración de un producto determinado

5. Trabajos relacionados

En la Sección 2 se mencionó la propuesta original de Jacobson para mostrar la variabilidad en una línea de productos con modelos UML y sus debilidades [12]. Las soluciones más recientes propuestas pasan por modificar o anotar dichos modelos, tanto en modelos estructurales como funcionales o incluso dinámicos.

Distintos autores han propuesto representar explícitamente los puntos de variación añadiendo anotaciones o cambiando la esencia de los diagramas de casos de uso. Por ejemplo, Von der Maßen [21] propone utilizar una notación gráfica con nuevas relaciones ("option" y "alternative"), con la consiguiente extensión del meta-modelo de UML. John y Muthig [13] sugieren la aplicación de plantillas de casos de uso, utilizando estereotipos, aunque no distinguen entre variantes opcionales, alternativas u obligatorias. En cambio Halman y Pohl [11] defienden la modificación de los modelos de casos de uso para representar de forma gráfica (mediante un triángulo con anotaciones) los puntos de variación. En todos los casos se trata de modificar el modelo original UML para conseguir el propósito deseado. Aunque la solución es válida creemos que el coste de entrada, tanto en aprendizaje de nuevas técnicas de modelado, necesidad de herramientas CASE ad hoc, etc., representa una barrera a menudo insalvable para la adopción del enfoque de desarrollo de líneas de productos en muchas organizaciones.

En cuanto a los modelos estructurales, o bien se utilizan directamente los mecanismos de UML (mediante la relación de especialización, la multiplicidad de las asociaciones, etc.) como en el caso mencionado de Jacobson, o bien se anotan de forma explícita mediante estereotipos. El trabajo de Gomaa [8] es un ejemplo de este último enfoque, ya que utiliza los estereotipos <<kernel>>, <<optional>> y <<variant>> (que corresponden a clases obligatorias, opcionales, y variantes). De un modo similar Clauß [3] propone un conjunto de estereotipos en la misma línea para expresar la variabilidad: <<optional>>, <<variationPoint>> y <<variant>> (para designar respectivamente clases opcionales, puntos de variación y sus subclases variantes). El problema con este tipo de aproximaciones es que aunque abordan el requisito 2

de la Sección 2, requieren el cambio del meta-modelo de UML (incumplen el requisito 3), y en cualquier caso no resuelven el requisito 1 (correspondencia uno-a-uno entre elementos).

Otra solución propuesta por Czarnecki en [5] y [7] consiste en anotar los diagramas de UML con condiciones de presencia expresados por ejemplo mediante códigos de colores (Figura 8), de modo que cada característica opcional se refleja en una o, de forma más realista, en varias partes de un diagrama (puede ser una clase, una asociación, un atributo, etc. o una combinación de elementos). Aunque en este trabajo se muestra un diagrama de clases, la técnica de representación se puede aplicar a otros diagramas de UML, en particular los diagramas de actividades. En este caso, la ventaja es que no se limita de forma artificial la representación de una variante a un único elemento e incluso el código de colores ayuda a destacar de un solo vistazo las implicaciones de elegir una cierta variante. Sin embargo esta ayuda visual es muy poco escalable en cuanto se manejen más de una docena de variantes. Por otro lado, de nuevo es necesario introducir elementos auxiliares en el meta-modelo de UML. Por esta razón, creemos que nuestra solución es mejor que las propuestas analizadas.

La validez de la operación de combinación de paquetes en el modelado se sistemas en general ha sido analizada en [23] llegando a la conclusión de que respeta las propiedades de unicidad y asociatividad, pero no la de conmutatividad. Por esta razón no se recomienda el uso del mecanismo en general para el modelado dado que si un paquete receptor se combina con dos o más paquetes el orden de combinación influye en el resultado. Sin embargo, en nuestro caso concreto, la construcción de modelos de paquetes a partir del modelo de características es jerárquica en todo momento por lo que se evita el problema (un paquete receptor siempre se relaciona con un único paquete base).

6. Conclusiones y trabajo futuro

En este trabajo se ha mostrado la viabilidad de una nueva propuesta para mostrar de forma explícita la variabilidad de una línea de productos utilizando únicamente modelos estándar de UML y modelos de características. Esta posibilidad simplifica la adopción de este paradigma de desarrollo, evitando la necesidad de herramientas o técnicas específicas como en otros trabajos previos.

Además, la combinación de paquetes permite derivar la implementación de cada aplicación concreta a partir de la configuración de características deseadas. La técnica se ha aplicado a un caso de estudio publicado en la literatura y se están desarrollando varias líneas de productos de aplicación industrial para evaluar la escalabilidad de la propuesta.

Como trabajo pendiente se encuentra en fase de diseño (utilizando las *DSL* tools de Microsoft) una herramienta que permita definir y configurar los modelos de características y enlace con las herramientas ya disponibles. Para ello, hay que definir un lenguaje específico de dominio para describir los modelos de características y capaz de generar la estructura de paquetes y los ficheros de configuración para el compilador de C#.

Agradecimientos

Este artículo ha sido parcialmente financiado por el MEC/FEDER y la Junta de Castilla y León (proyectos TIN-2004-03145 y VA-018A07 respectivamente). El segundo autor disfruta de una beca MEC-FPU.

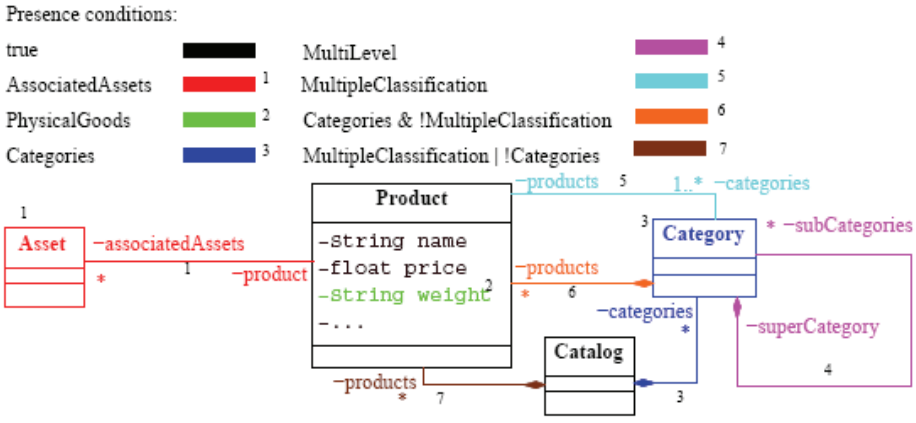


Figura 8. Un ejemplo de la solución propuesta por Czarnecki en [5] y [7]

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. 2000.

[2] Bosch, J. "*Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach*". Addison-Wesley. 2000. Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. "Non-Functional Requirements in Software Engineering" Kluwer Academic Publishers 2000.

[3] Clauß, M. *Generic modeling using Uml extensions for variability*. In Workshop on Domain Specific Visual Languages at OOPSLA 2001, 2001.

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

[5] Czarnecki, K., M. Antkiewicz, Mapping Features to models: a template approach based on superimposed variants, In proc. International Conference on Generative Programming and Component Engineering (GPCE'05), LNCS 3676, Springer, pp. 422-437.

[6] Czarnecki, K., and U. W. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston, MA, 2000.

[7] Czarnecki, K., and K. Pietroszek. Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints. In Proceedings of ACM SIGSOFT/SIGPLAN International Conference on Generative Programming and Component Engineering (GPCE'06), ACM Press, 2006

[8] Gomaa, H. Object Oriented Analysis and Modeling for Families of Systems with UML. InW. B. Frakes, editor, IEEE International Conference for Software Reuse (ICSR6), June 2000, pages 89–99.

[9] González-Baixauli, Bruno, 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, 2004, pp 77-87.

[10] 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.

[11] Halmans, G., and Pohl, K., "Communicating the Variability of a Software-Product Family to Customers". Journal of Software and Systems Modeling 2, 1 2003, 15--36.

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

[13] John, I., Muthig, D.: Tailoring Use Cases for Product Line Modeling. Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 (REPL'02). Technical Report: ALR-2002-033, AVAYA labs, 2002

[14] 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, 1998. 5:143-168.

[15] Kang, K. C., 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

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

[17] Lau, S., "Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates", MASc Thesis, ECE Department, University of Waterloo, Canada, 2006.

[18] David A. Moon, Object-oriented programming with flavors, Conference proceedings on Object-oriented programming systems, languages and applications,, 1986, pp. 1-8.

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

[20] Object Management OMG. Unified modeling language specification version 2.0 : Infrastructure. Technical Report ptc/03-09-15, OMG, 2003.

[21] von der Maßen, T., Lichter, H.: Modeling Variability by UML Use Case Diagrams. Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 (REPL'02). Technical Report: ALR-2002-033, AVAYA labs, 2002.

[22] Waldén, K. & Nerson, J-M. Seamless Object-Oriented Software Architecture, analysis and design of reliable systems. Object-Oriented Series, Prentice Hall International, 1995.

[23] Zito, A., Z. Diskin, and J. Dingel. Package merge in UML 2: Practice vs. theory? In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, MoDELS/UML 2006, October 2006.