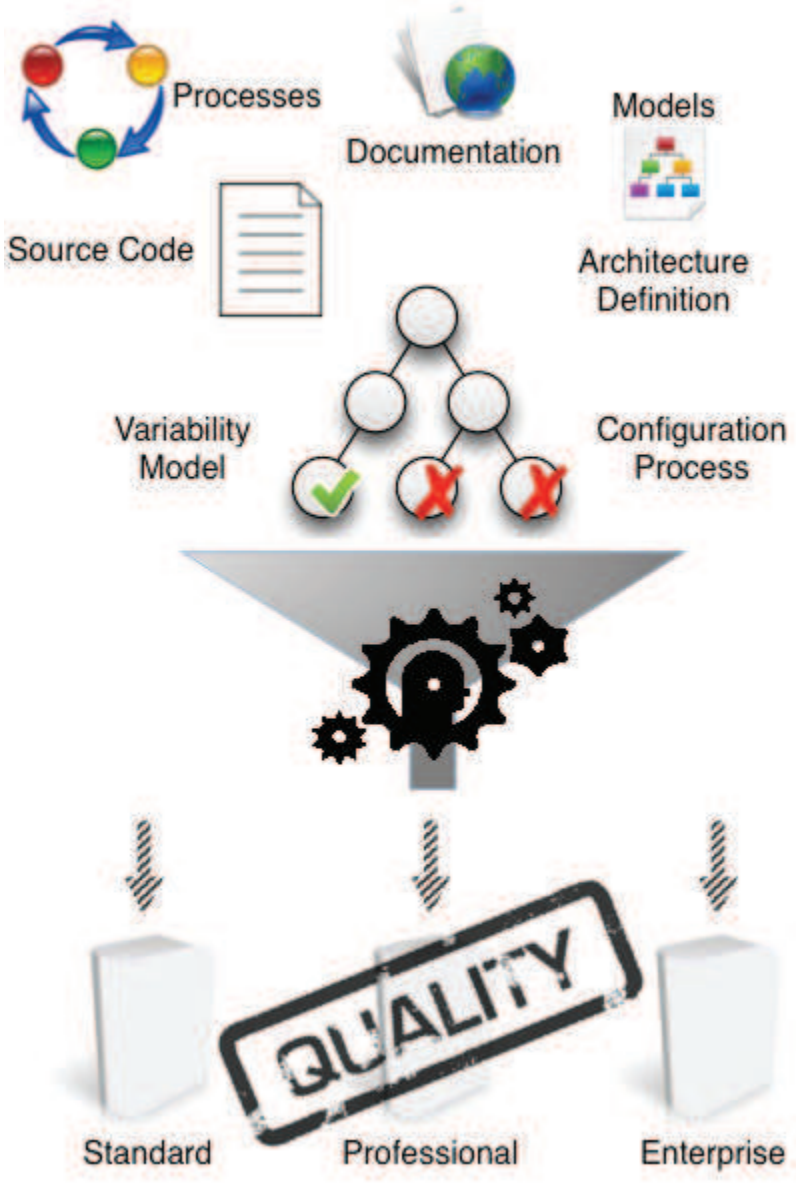
Manejo de variabilidad positiva y negativa en modelos de decisión



Andrés Felipe Paz[1]
Andrés Felipe Esguerra[2]
Diego Armando Rojas[3]
Fabio Andrés García[4]
Hugo Arboleda[5]

Resumen

Las líneas de producto de software dirigidas por modelos (MD-SPLs) son un enfoque de desarrollo generativo de software que ofrece acelerar el desarrollo mientras se alcanzan objetivos de costos y calidad. Su efectividad para derivar productos de software radica en el orden preciso de ejecución de las transformaciones de modelos que incrementalmente introducen los detalles de implementación, y en una óptima administración de la variabilidad. En este artículo presentamos un enfoque, basado en modelos, para expresar las decisiones del proceso de derivación de productos de software que incluye técnicas de variabilidad positiva y negativa. Nuestra estrategia se basa en el uso de un DSML para

[1] Grupo de Investigación en Informática y Telecomunicaciones (i2t), Universidad Icesi, Cali, Colombia. andres.paz@correo.icesi.edu.co
[2] Estudiante Ingeniería de Sistemas, Universidad Icesi, Cali, Colombia. andres.esguerra1@correo.icesi.edu.co
[3] Estudiante Ingeniería de Sistemas, Universidad Icesi, Cali, Colombia. diego.rojas1@correo.icesi.edu.co
[4] Estudiante Ingeniería de Sistemas y Administración de Empresas, Universidad Icesi, Cali, Colombia. fabio.garcia@correo.icesi.edu.co
[5] Grupo de investigación en Informática y Telecomunicaciones (i2t), Universidad Icesi, Cali, Colombia. hfarboleda@icesi.edu.co
Septiembre 16/11 • Fecha de aceptación: Octubre 14 /11

crear modelos de decisión que permiten enlazar las características de una línea de productos con las reglas de transformación requeridas para derivarlos.

Palabras clave: Desarrollo dirigido por modelos, líneas de producto de software, administración de variabilidad, modelo de decisión.

Abstract

Model-Driven Software Product Lines (MD-SPLs) is a generative approach to software development that offers speed while achieving cost and quality goals. The effectiveness of them to derive software products lies in the precise order of execution of the model transformations that incrementally introduce implementation details, and an optimal variability management. In this article we present a model-based approach to express decisions in the software product derivation process that include variability management using different variability techniques. We use a DSML to create decision models that allow links between the characteristics of a product line and the transformation rules required to derive them.

Key words: Model-Driven Software Development, Software Product-Lines, variability management, decision model.

Introducción

Acelerar el desarrollo de software mientras se reducen los costos y se aumenta la calidad se ha convertido en una creciente preocupación para los fabricantes de software. Cuando el tiempo se empieza a agotar, las empresas abandonan los modelos y la documentación. El desarrollo de software dirigido por modelos (MDSD) surge como un enfoque mucho más efectivo que modelar solo para documentar el software. En MDSD los modelos constituyen un factor clave en aumentar tanto la velocidad como la calidad de los desarrollos en el sentido que la mayor parte de la aplicación final puede ser generada a partir de ellos. En este caso se concibe el ciclo de desarrollo de software como un proceso de creación, refinación iterativa e integración de modelos.[1]

Con el enfoque de desarrollo de software dirigido por modelos (MDSD), el código de una aplicación se estructura desde modelos orientados hacia el dominio del espacio del problema, y que permiten la abstracción de la implementación. Las aplicaciones se derivan a partir de la transformación de estos modelos. Pueden ocurrir varias etapas de transformación intermedias durante las cuales se incluyen más detalles de implementación. Dicho de otra forma, los modelos son transformados incrementalmente desde el espacio del problema hasta el espacio de la solución con el uso de reglas de transformación. Dicha regla introduce detalles de implementación para una parte del producto final. El último paso de la transformación lleva todos los detalles de implementación al código fuente.

Nuestro trabajo se basa en los principios de MDSD para la creación de Líneas de Producto de Software (SPLs). Una línea de producto de software es un conjunto de productos similares creados a partir de una arquitectura común, activos reutilizables y un proceso para derivar productos.[2] La descripción del conjunto de productos determina las características comunes, o compartidas, entre todos ellos y la forma en la que estos se pueden variar. Los diferentes puntos de variación definidos y los múltiples valores o

variantes disponibles para escoger se incluyen en un modelo de variabilidad. A la selección de estas variantes con base en las necesidades del cliente se le conoce como una configuración. En particular usamos modelos de características para representar los puntos de variabilidad y modelos de configuración para representar las configuraciones.

Las líneas de producto de software dirigidas por modelos (MD-SPL) derivan sus productos a partir de un orden preciso de las transformaciones de los modelos de acuerdo con una configuración. La mayoría de los enfoques MD-SPL [3, 4] no ofrecen la posibilidad de modelar y adaptar la ejecución de las transformaciones de modelos para derivar productos. En el trabajo realizado por Arboleda, Romero, Casallas y Royer [5] se definió una estrategia para (semi-)automatizar el proceso de la transformación de modelos. Ellos sugieren utilizar un lenguaje de modelado de dominio específico (DSML) para la creación de modelos de decisión que permiten la selección y ejecución automática de las transformaciones en un orden específico para derivar el producto deseado. Aunque esta propuesta es capaz de derivar un producto de acuerdo con las opciones deseadas por el usuario, no cuenta con una completa administración de la variabilidad. Nuestro enfoque parte del mecanismo para modelar y procesar las relaciones entre los variantes y las transformaciones de Arboleda *et al*. [5], pero mejora este mecanismo teniendo en cuenta las diferentes técnicas de variabilidad identificadas por Groher y Voelter [6]. Algunos autores [7] han señalado los modelos de decisión como la introducción a lo que se conoce como megamodelos. Un megamodelo se define como un modelo donde algunos de sus elementos representan o referencian modelos, metamodelos y las relaciones entre estos. Bézivin argumenta [7] acerca de la necesidad de los megamodelos en una variedad de situaciones, entre las que se encuentra el diseño de plataformas de desarrollo generativo dirigidas por modelos. Actualmente, hay investigaciones acerca de los potenciales usos de megamodelos en MDSD.

El resto de este artículo está organizado de la siguiente manera. La Sección 1 presenta los trabajos relacionados. La Sección 2 detalla el trabajo previo realizado que tomamos como base para nuestro enfoque. La Sección 3 introduce la necesidad de capturar la variabilidad y las diferentes técnicas identificadas para hacerlo. La Sección 4 presenta y explica nuestra propuesta de DSML para derivar productos usando modelos de decisión con administración de variabilidad. Finalmente, la Sección 5 hace un recuento de las principales características de nuestro enfoque, y muestra un resumen comparativo con los enfoques presentados en la Sección 1, y expone el trabajo futuro.

Trabajos relacionados

Nuestro trabajo está relacionado con la configuración y derivación (semi-)automática de productos de software en SPL. Czarnecki y Antkiewicz [3] proponen la utilización de plantillas de modelos para estructurar SPLs. Las plantillas de modelos contienen las reglas de transformación que se anotan con metadata en términos de las características definidas en un modelo de características para representar la variabilidad. Para crear un producto específico se crea una configuración seleccionando las características disponibles en el modelo de características. Las anotaciones crean un acoplamiento entre el modelo de variabilidad y las reglas de transformación, lo cual no solo dificulta la reutilización de las

reglas de transformación y también del modelo de variabilidad sino el mantenimiento de la SPL.

Por otro lado, Tessier *et al*. [4] emplean también plantillas de modelos para estructurar SPLs y modelos de características para capturar la variabilidad. Sin embargo, proponen un modelo adicional e independiente, un modelo de decisión, para relacionar las características con las reglas de transformación. A pesar de que es posible reutilizar las reglas de transformación ya creadas, no es posible adaptar las transformaciones de acuerdo con las configuraciones. Voelter y Groher [8], en cambio, estructuran las SPL basándose en MDSD y emplean modelos de dominio del espacio del problema en conjunto con modelos de características para capturar la variabilidad. A pesar de esto, las decisiones del proceso de derivación deben escribirse manualmente en archivos de texto.

Arboleda *et al*. [5] proponen, al igual que Tessier et al., relacionar las características con las reglas de transformación en un modelo de decisión. El modelo de decisión, sin embargo, permite capturar el orden de ejecución de las reglas de transformación y cuáles de estas reglas deberán interceptarse ante la presencia de un conjunto de características en una configuración.

Antecedentes

En este artículo proponemos un enriquecimiento a la propuesta de Arboleda *et al*. [5]. Allí los autores plantean hacer uso de modelos de decisión en un contexto de MDSD para capturar la variabilidad de una SPL en términos de decisiones abiertas de los usuarios y sus posibles resoluciones. Particularmente, los modelos de decisión propuestos en este enfoque permiten la representación de *i)* las relaciones entre las características de una línea de productos y las reglas de transformación específicas para introducirlas, *ii)* el orden de ejecución requerido para la derivación (semi-)automática de un producto con base en una configuración de características, y *iii)* las diferentes etapas de transformación que deben llevarse a cabo para el proceso de derivación. La idea fundamental es que el modelo de decisión contenga el ordenamiento básico, es decir, la secuencia de llamados a las reglas de transformación base que derivan un producto genérico. Este ordenamiento básico será modificado de acuerdo con una configuración de características para un producto particular a través de programación orientada a aspectos (AOP). Los aspectos en este caso especifican la regla de transformación base que deberá ser interceptada y la regla de transformación específica que será introducida y ejecutada en su lugar.

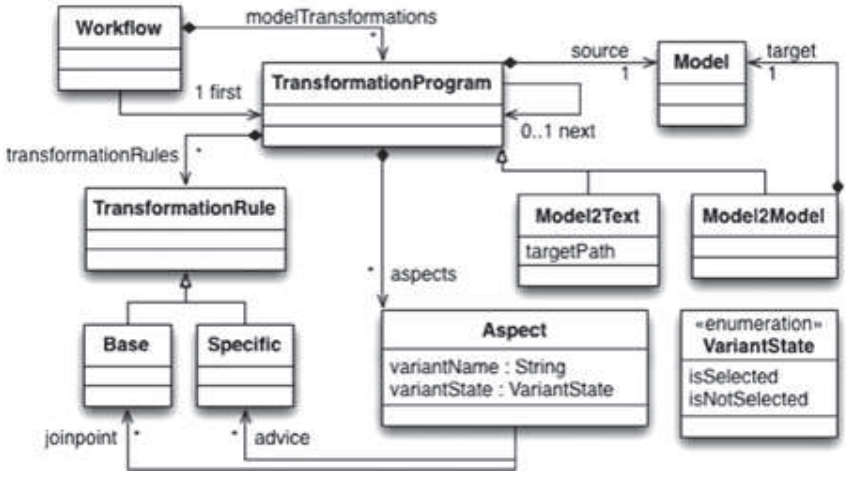


Figura 1. Metamodelo de decisión planteado por Arboleda et al. [5].

Los modelos de decisión involucran varios conceptos, que se presentan en la Figura 1. A los conjuntos de reglas de transformación base con un orden de ejecución específico, reglas de transformación específicas y aspectos le han dado el nombre de *TransformationProgram*, o programa de transformación. Un programa

de transformación puede transformar un modelo en otro modelo o traducirlo a código fuente. A la secuencia de programas de transformación se le llama Workflow, o flujo de trabajo. Aunque este enfoque puede derivar un producto con base en una configuración de características, solo es posible representar puntos de variación usando estrategias basadas en AOP. Esto limita el alcance del enfoque ya que no tiene en cuenta la posibilidad de i) entretejer modelos creados previamente y disponibles en un repositorio de artefactos reusables y ii) remover estructuras de modelos que no son necesarias de acuerdo con una configuración dada.

Técnicas de variabilidad

Una SPL define un conjunto de productos que varían en características bien definidas. Un producto se adapta durante el proceso de generación con base en una configuración usando los mecanismos que provee el lenguaje de implementación de la SPL o herramientas de configuración y construcción. Sin embargo, la variabilidad tiene un impacto a lo largo del ciclo de vida y en múltiples artefactos de la SPL. En consecuencia, los modelos y sus transformaciones deben poder variar según la configuración de un producto particular. Es necesario poder expresar esa variabilidad de diferentes maneras. Groher y Voelter [6] distinguen dos formas fundamentales de implementar la variabilidad en SPL: variabilidad positiva y variabilidad negativa.

Variabilidad positiva

En la variabilidad positiva se parte de un conjunto de elementos con un mínimo común a todos los productos al cual se le agregan elementos adicionales de acuerdo con una configuración de un producto particular. La Figura 2 muestra una vista esquemática del resultado.

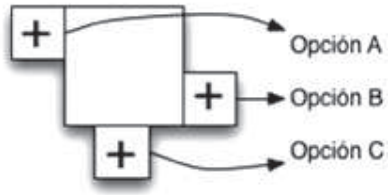


Figura 2. Variabilidad positiva, adaptada de [6].

La variabilidad positiva se basa en técnicas orientadas a aspectos (AO) como el modelado orientado a aspectos (AOM) [8, 10] y AOP [11, 12]. Decimos entonces que la variabilidad positiva puede estar orientada a modelos o a programa. La variabilidad positiva orientada a modelos se apoya en la estrategia de entretejido de modelos, es decir, la composición de modelos separados en uno solo. Se tiene un modelo base que define los elementos comunes. En otro modelo, conocido como el modelo de aspectos, se define la variabilidad. El modelo de aspectos también incluye los puntos en donde se deben adicionar los nuevos elementos al modelo base teniendo en cuenta una configuración. El modelo base y el modelo de aspectos se entretejen en un nuevo modelo, el modelo resultado. Por ejemplo, en la Figura 3 el modelo base define los elementos A, AB y C, sus relaciones y sus atributos. El modelo de aspectos agrega un elemento D, subtipo de C, y un atributo al elemento A. El modelo que resulta incluye tanto los elementos del modelo base como los elementos del modelo de aspectos.

La variabilidad positiva orientada a programa se apoya en la interceptación de ciertas reglas de transformación para introducir la variabilidad en los productos. En este caso se crean dos tipos de reglas de transformación. El primer tipo se conoce como las reglas base, que contienen las instrucciones para derivar los elementos comunes en todos los productos. El segundo

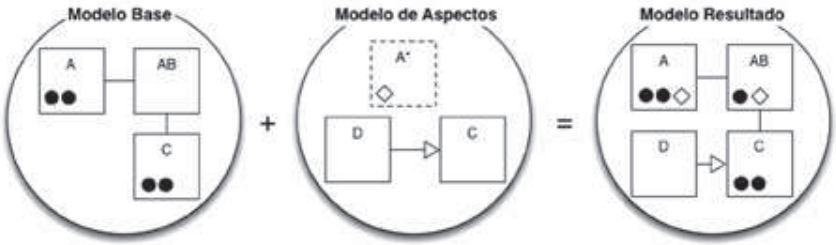


Figura 3. Entretejido de modelos, adaptada de [6].

tipo son las reglas específicas que contienen las instrucciones para introducir los elementos específicos configurados. La información acerca de cuáles reglas base se deben interceptar y cuándo se debe hacerlo (joinpoint), y qué reglas específicas ejecutar (advice) se captura en lo que se conoce como aspectos. Por ejemplo, en la Figura 4 se describen tres reglas base que se ejecutan en el orden presentado, tres reglas específicas que crean elementos particulares, y dos aspectos que introducen la variabilidad. El aspecto Crear Grilla intercepta la regla base crear vistas comunes y ejecuta la regla específica crear componente grilla en su lugar. Lo mismo sucede con el aspecto Crear Comparable.

Variabilidad negativa

La variabilidad negativa es inversa a la variabilidad positiva. En esta técnica se crea un modelo "global" que incluye todas las características disponibles. Los elementos del modelo de características se conectan a partes del modelo global. Las partes del modelo "global" se eliminan de acuerdo con la ausencia de características en la configuración de un producto particular, como se observa en la Figura 5.a. Por ejemplo, en la Figura 5.b se muestra un modelo de características cuyos elementos se encuentran enlazados con los elementos de un modelo. El modelo de características presenta dos características obligatorias (F2 y F3) y dos opcionales (F1 y F4). En el modelo resultante el elemento A y la relación entre B y C sólo estarán presentes si las características F1 y F4 hacen parte de la configuración del producto.

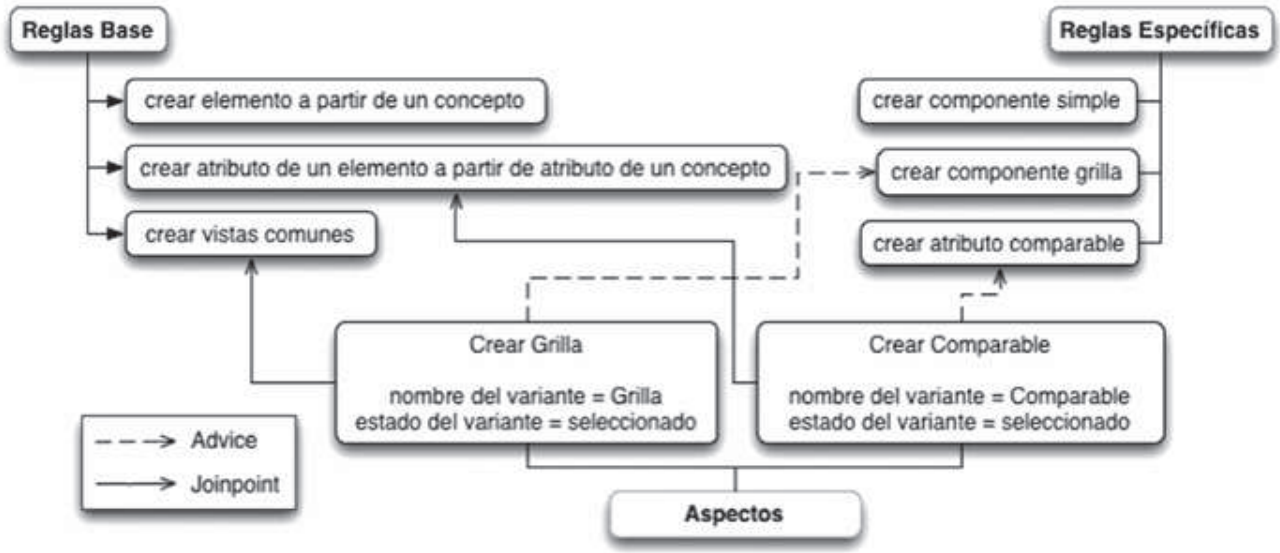


Figura 4. Derivación de un producto, adaptada de [5].

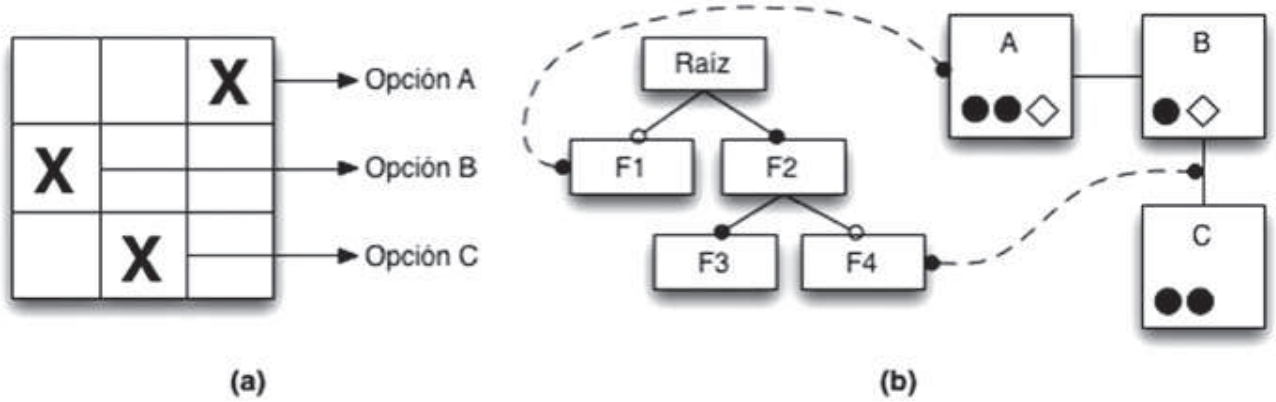


Figura 5. (a) Variabilidad negativa, adaptada de [6] y **(b)** Ejemplo de variabilidad negativa.

Captura de variabilidad positiva y negativa en modelos de decisión

El enfoque general que proponemos consiste en expresar, usando un DSML, *i)* el proceso para la transformación de modelos, y *ii)* los puntos de variación a través de las técnicas de variabilidad de la Sección 3. Para definir el DSML hemos extendido el metamodelo de decisión de la Figura 1 para que incluya conceptos que permitan hacer uso de las técnicas de variabilidad positiva y negativa, como se explicó anteriormente. En el DSML se identifican dos conceptos clave que permiten la administración de la variabilidad: el modelo de características y los puntos de variación. El modelo de características define el modelo de variabilidad que representa las diferentes características que hacen parte de los productos en la SPL. Un punto de variación define la configuración de la variabilidad para cada uno de los programas de transformación, que puede ser positiva o negativa. El metamodelo del DSML se explica con más detalle en la Subsección 4.2.

Caso de estudio

Hemos tomado la SPL creada por Arboleda *et al*. [6], que permite generar productos para el manejo de colecciones de datos, y la hemos adaptado para el caso de estudio de una tienda de música. La aplicación generada para la tienda de música muestra el listado de canciones registradas y permite agregar nuevas canciones. Además, muestra la información de la canción seleccionada en la lista de canciones (e.g. nombre, duración) y permite ordenar la lista de canciones por cualquier atributo. La SPL estructura los productos a partir de un modelo de dominio del espacio del problema en un componente de kernel, que implementa la funcionalidad del manejo de la colección de datos, y en un componente gráfico, que implementa la interfaz gráfica de usuario (GUI). Para el alcance de este artículo sólo tenemos en cuenta el componente de kernel.

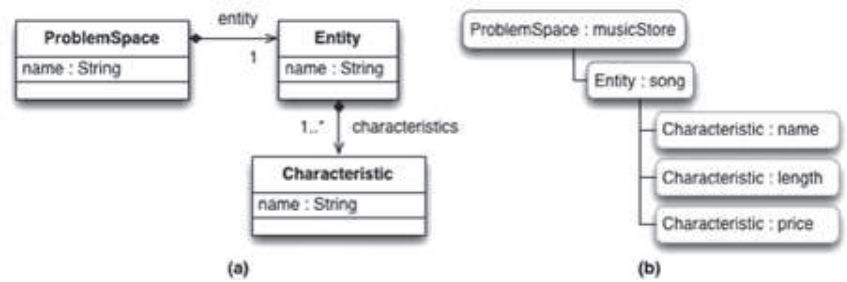


Figura 6. (a) Metamodelo y **(b)** modelo del espacio del problema.

El metamodelo del espacio del problema abstrae los elementos del mundo real. De esta forma se tiene una manera sencilla de representar la estructura del negocio. La Figura 6.a representa el metamodelo mencionado; la Figura 6.b presenta el modelo del espacio del problema para la tienda de música. El concepto ProblemSpace encapsula todo el contexto. El concepto Entity representa el elemento que se administra en la colección. El concepto Characteristic representa los distintos atributos que describen a la entidad.

El componente de kernel de la línea introduce los detalles de la implementación de los requerimientos funcionales de la aplicación. A partir de un programa de transformación se traduce el modelo del espacio del problema en un modelo conforme al metamodelo de kernel. A este programa de transformación es posible introducirle puntos de variación de acuerdo con la funcionalidad deseada por el usuario. Para este caso de estudio la fuente de variación está en los algoritmos de ordenamiento. El componente de GUI introduce los detalles de la implementación visual y de las interacciones. Usando un programa de transformación se transforma el modelo del espacio del problema en un modelo conforme al metamodelo de GUI. Aunque el caso de estudio de la tienda de música no introduce puntos de variación en la GUI, a este programa de transformación también es posible introducirle puntos de variación de acuerdo con las funcionalidades deseadas por el usuario.

Configuración y derivación de productos con el metamodelo de decisión

Introducir puntos de variación en el caso de estudio de la tienda de música brinda la posibilidad de cambiar sus características de acuerdo con una configuración particular. Por ejemplo, en una configuración puede seleccionarse sólo el ordenamiento con el algoritmo de burbuja. A partir de esta y usando las técnicas de variabilidad identificadas se pueden agregar o remover las características de la aplicación.

La Figura 7 muestra el metamodelo de decisión creado para el DSML. Este consiste principalmente en un conjunto de conceptos que contienen la información necesaria para la generación de un descriptor de código consistente con la aplicación que se desea generar. El concepto Workflow realiza el papel de contenedor de diferentes elementos, como son los modelos, metamodelos, el modelo de variación, los programas de transformación y el modelo de configuración del producto. Los modelos y metamodelos a transformar se representan en M1 y M2, respectivamente. Un TransformationProgram describe el conjunto de transformaciones que llevan el modelo de dominio del espacio del problema a la solución de software deseada. Una TransformationRule define la correspondencia de ciertos elementos de unos modelos de entrada con un modelo de salida o con código fuente. El FeatureModel define las características de la familia de productos. Una Configuration, creada por el diseñador de productos, contiene el conjunto de variantes requeridos para aplicar puntos de variación. Un Variant define si una característica está o no seleccionada para una configuración, esto es un manejo grueso de la selección de las características. Los VariationPoint pueden emplear técnicas tanto positivas como negativas. Si el punto de variación emplea una técnica positiva, puede ser ModelOriented o ProgramOriented. Si es orientada a modelos se debe definir el modelo base y el modelo de aspectos que será entretejido.

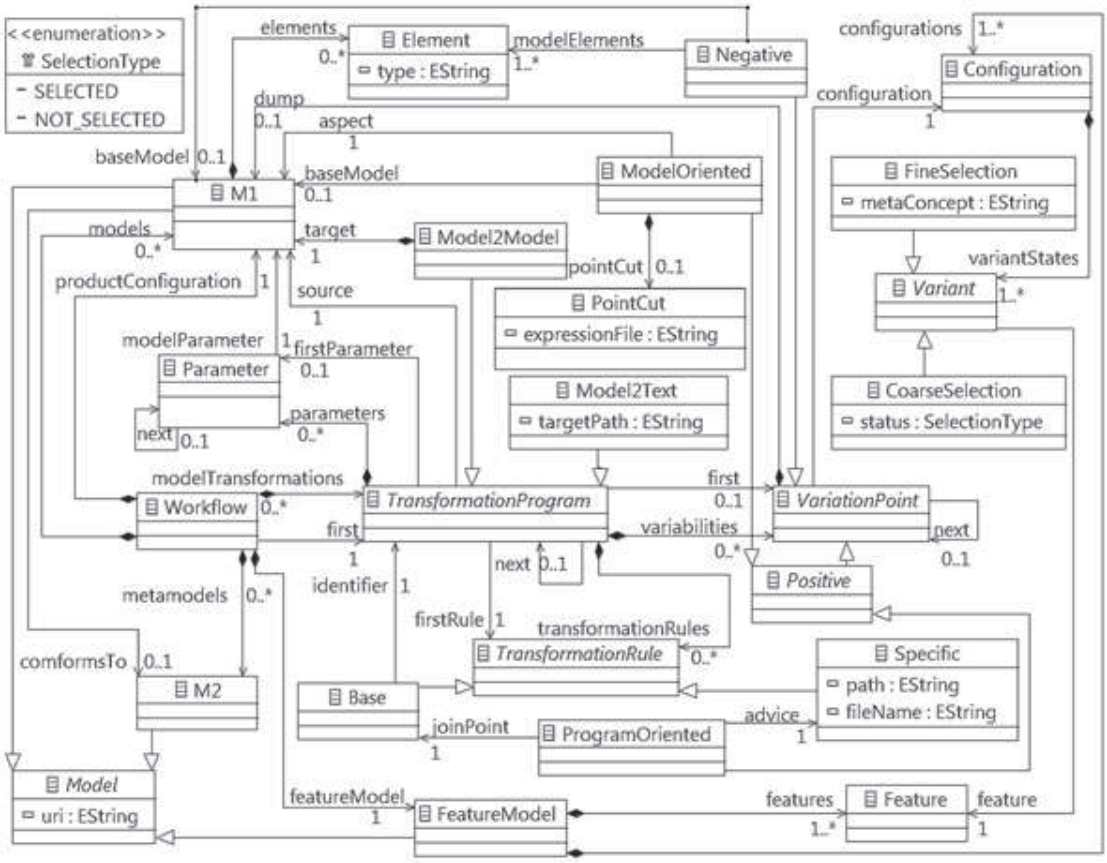


Figura 7. Metamodelo del DSML.

Si es orientada a programa se debe especificar el joinpoint, o la regla de transformación base que será interceptada, y el advice, es decir, la regla de transformación específica que será ejecutada en su lugar. Si el punto de variación emplea una técnica negativa, se seleccionan los elementos a eliminar del modelo. De esta forma, al crear el modelo de decisión se puede tener una representación global de los modelos y metamodelos que se requieran transformar, los programas de transformación que se deben llevar a cabo en la ejecución del flujo de trabajo para generar el producto deseado y las reglas de transformación en las que se han incluido los puntos de variación, positivos y negativos.

El modelo de decisión de la Figura 8 define un programa de transformación de modelo a modelo para transformar el modelo de dominio del espacio del problema a un modelo del kernel. Primero se especifican las reglas de transformación base que crean los componentes genéricos del kernel a partir del modelo del espacio del problema. El programa de transformación define un punto de variación positivo orientado a programa, el cual contiene el conjunto de reglas específicas y los aspectos con la información acerca de la variación a introducir de acuerdo con la configuración. Esto es, si el variante Bubble está seleccionado en la configuración se intercepta la regla base createOrdering y se ejecuta la regla específica, createBubbleOrdering en su lugar. Algo similar sucede si el variante Insertion está seleccionado. En este caso se intercepta la regla base createOrdering y se ejecuta la regla específica createInsertionOrdering en su lugar.

Para el modelo de decisión presentado en la Figura 9 se ha incluido

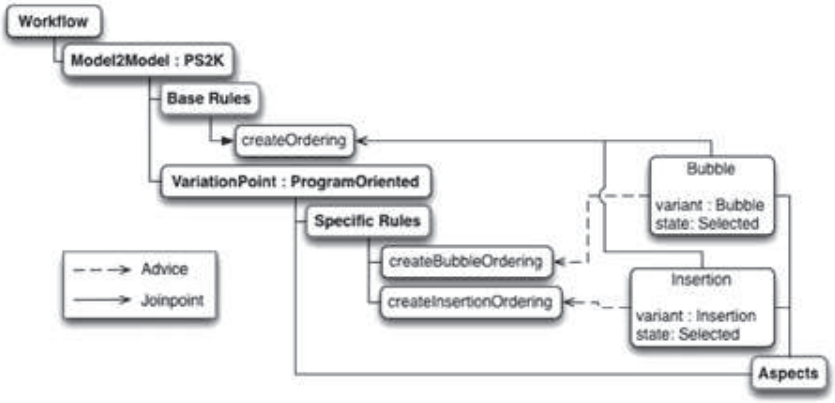


Figura 8. Fragmento de un modelo de decisión del caso de estudio usando variabilidad positiva AOP.

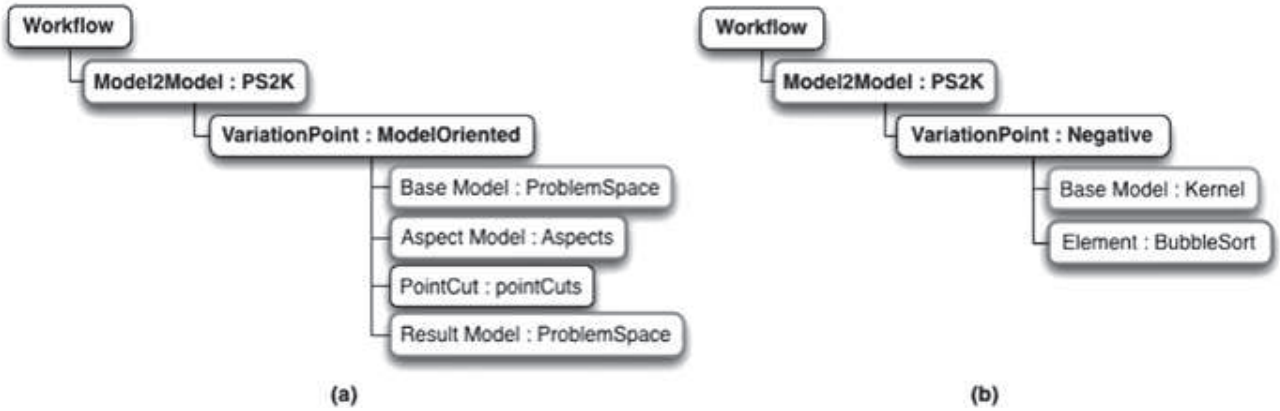


Figura 9. Fragmento de un modelo de decisión del caso de estudio usando **(a)** variabilidad positiva AOM y **(b)** variabilidad negativa.

un punto de variación positivo orientado a modelos en un programa de transformación de modelo a modelo. Aquí se va a entretejer el modelo del espacio del problema con un modelo de aspectos, que define las características a introducir, de acuerdo con las expresiones descritas en el archivo de expresiones pointCuts. El modelo resultante es conforme al metamodelo del espacio del problema y será escrito sobre el modelo del espacio del problema original.

En el modelo de decisión de la Figura 10 se define un punto de variación negativo, donde al modelo del kernel se le eliminará el elemento BubbleSort si en la configuración esta característica no se encuentra seleccionada.

Implementación

Hemos desarrollado este enfoque sobre el Eclipse Modeling Framework (EMF) [13] y openArchitectureWare (oAW) [14], y se han usado particularmente las herramientas XWeave y XVar para implementar las técnicas de variabilidad positiva y negativa, respectivamente. La variabilidad de los productos de software y las configuraciones de productos se capturan con el empleo del modelo de características y de configuración de pure::variants [15], respectivamente.

También hemos creado un conjunto de reglas de transformación para traducir los modelos de decisión a texto, particularmente workflows

de oAW. En el Listado 1 se presenta un fragmento del workflow generado para el caso de estudio. Inicialmente la configuración es cargada de un modelo de configuración pure::variants (líneas 2-4). Luego se cargan el modelo del dominio del espacio del problema (líneas 5-7) y el modelo de aspectos (líneas 8-10). Cuando los variantes de la característica Sort, es decir, insertion y bubble aparezcan seleccionados en la configuración (líneas 11-12) se ejecutará el componente para entretejer el modelo de dominio del espacio del problema con el modelo de aspectos basándose en un archivo de expresiones (líneas 13-15). Luego de entretejer los modelos, el modelo resultante es transformado para generar el componente kernel de la aplicación usando el programa de transformación ProblemSpace-2Kernel (líneas 18-27).

```
1.  <workflow>
2.    <component class=”org.openarchitectureware.util.featureconfig.
      pv.PVVariantModelReader”>
3.     <uri value=”ConfigurationPVModel”/>
4.   </component>
5.   <component id=”readProblemSpaceModel” class=”org.eclipse.mwe.
      emf.Reader”>
6.     ...
7.   </component>
8.   <component id=”readAspect” class=”org.eclipse.mwe.emf.Reader”>
9.     ...
10.  </component>
11.  <feature isSelected=”insertion”>
12.    <feature isSelected=”bubble”>
13.       <cartridge file=”org/openarchitectureware/util/xweave/wf-
      weave-expr”
14.       baseModelSlot=”problemSpaceModel” aspectFile=”aspect”
15.       expressionFile=”pointcuts” overwrite=”false” />
16.    </feature>
17.  </feature>
18.  <transform id=”ProblemSpace2Kernel”>
19.    <metaModel class=”org.openarchitectureware.type.emf.EmfMe-
      taModel”>
20.     <metaModelFile value=”metamodels/problemSpaceMetamodel.
      ecore”/>
21.    </metaModel>
22.    <metaModel class=”org.openarchitectureware.type.emf.EmfMe-
      taModel”>
23.     <metaModelFile value=”metamodels/kernelMetamodel.ecore”/>
24.    </metaModel>
25.    <invoke value=”transformations::kernel::problemSpace2kernel::
      problemSpace2kernel(problemSpaceModel)”/>
26.    <outputSlot value=”kernelModel”/>
27.  </transform>
28. </workflow>
```

Listado 1. Workflow oAW para el caso de estudio.

Conclusiones

Hemos presentado un DSML para crear modelos de decisión con el objetivo de tener un mecanismo (semi-)automático para derivar productos en una SPL. Los modelos de decisión permiten expresar *i)* el orden de las reglas de transformación base para la generación de los elementos comunes en los productos, *ii)* los puntos de variación a través de las técnicas de variabilidad positiva y negativa, y *iii)* cuando debe introducirse la variabilidad en los productos que se generan.

Nuestro enfoque introduce las técnicas de variabilidad en la propuesta de Arboleda *et al*. [5]. La variabilidad positiva permite agregar características, de acuerdo con una configuración, a un producto genérico a un nivel de modelos mediante AOM o a un nivel de programación mediante AOP. En modelos, un modelo base es entretejido con un modelo de aspectos. En programación se adaptan los programas de transformación de modelos interceptando las reglas de transformación base y ejecutando en su lugar reglas específicas de acuerdo con una configuración. Por otro lado, la variabilidad negativa adapta los productos y elimina las características que no hagan parte de la configuración.

La Tabla 1 presenta un resumen comparativo de las características de nuestro enfoque con las de los otros enfoques que hemos mencionado.

Como trabajo futuro realizaremos la implementación de nuestro enfoque usando el ATLAS Transformation Language (ATL) y el framework Acceleo, e implementaremos un editor gráfico. Adicionalmente, crearemos la definición de un DSL para el enfoque, e integraremos todo junto con la estrategia FieSta desarrollada por Arboleda

Tabla 1. Comparación de enfoques.

Característica	Nuestro enfoque	Czarnecki y Antkiewicz	Tessier et al.	Voelter y Groher	Arboleda et al.
Representación de reglas de transformación	X	X	X	X	X
Orden de ejecución de reglas de transformación	X				X
Representación de variabilidad con modelo de variabilidad	X	X	X	X	X
Relacionar variantes con reglas de transformación	X	X	X	X	X
Variabilidad positiva AOM	X				
Variabilidad positiva AOP	X	X		X	X
Variabilidad negativa	X				

[1] y el framework FAMA [16] en un framework para la configuración y derivación de productos de software en MD-SPLs.

Bibliografía

[1] H. F. Arboleda, "FieSta: An approach for Fine-Grained Scope Definition, Configuration and Derivation of Model-Driven Software Product Lines". Trabajo de grado Doctorado en Ingeniería Universidad de los Andes y Université de Nantes. (2009)

[2] BOSCH, Jan. Design and Use of Software Architectures: Adapting and Evolving a Product-Line Approach. First Edition. Boston, MA, USA: Addison-Wesley, 2000. 368 p.

[3] K. Czarnecki, M. Antkiewicz, "Mapping Features to Models: A Template Approach Based on Superimposed Variants". GPCE, LNCS, pp 422-437. (2005)

[4] P. Tessier, S. Gérard, F. Terrier y J. M. Geib, "Using Variation Propagation for Model-Driven Management of a System Family". LNCS, pp 222-233. (2005)

[5] H. Arboleda, A. Romero, R. Casallas, y J.-C. Royer, "Product Derivation in a Model-Driven Software Product Line using Decision Models", Proceedings of the 12th Iberoamerican Conference on Requirements Engineering and Software Environments (IDEAS'09), pp 59-72. (2009)

[6] I. Groher y M. Voelter, "Expressing Feature-Based Variability in Structural Models", Workshop on Managing Variability for Software Product Lines. (2007)

[7] J. Bézivin, F. Jouault, y P. Valduriez, "On the need for Megamodels", Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA). (2004)

[8] M. Voelter y I. Groher, "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development". Proceedings of the 11th SPLC, pp 233-242 (2007)

[9] Aspect-Oriented Modeling Workshops website [actualizado en 27 mayo 2009] [consultado en 17 agosto 2011]. Disponible en: <http://www.aspect-modeling.org/>

[10] CLARKE, Siobhán, y Baniassad, Elisa. Aspect-Oriented Analysis and Design: The Theme Approach. First Edition. Amsterdam: Addison-Wesley Longman, 2005. 400 p.

[11] FILMAN, Robert E., Elrad Tzilla, Clarke, Siobhán, Aksit, Mehmet. Aspect-Oriented Software Development. First Edition. Boston: Addison-Wesley, 2005. 800 p.

[12] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J. M. Loingtier, *et al*. "Aspect-Oriented Programming". Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. (1997)

[13] Eclipse Modeling EMF [consultado en 12 agosto 2011]. Disponible en <http://www.eclipse.org/modeling/emf/>

[14] openArchitectureWare (oAW) [consultado 11 agosto 2011]. Disponible en <http://www.openarchitectureware.org/>

[15] pure::variants Community [consultado en 11 agosto 2011]. Disponible en <http://www.pure-systems.com/pure_variants.49.0.html>

[16] FAMA Framework [consultado 11 agosto 2011]. Disponible en http://www.isa.us.es/fama/?Community