Generación automática de arquitecturas de líneas de productos de software usando MDE

Gentileza Johan Fabry

Cecilia Bastarrica
Profesora Asistente, DCC, Universidad de Chile. PhD Computer Science and Engineering, University of Connecticut (2000); Magíster en Ciencias de la Ingeniería, Universidad Católica de Chile (1994); Ingeniera en Informática, Universidad Católica del Uruguay (1991). cecilia@dcc.uchile.cl

DISEÑO DE ARQUITECTURAS DE SOFTWARE

La arquitectura del software es la estructura del software entendida como los elementos que actúan entre sí, como también sus propiedades externamente visibles. O sea, la arquitectura del software no tiene relación con algoritmos ni estructuras de datos, que son relevantes para la construcción de cada uno de los elementos constitutivos del software, sino con la composición de dichos elementos de software y eventualmente del hardware.

La arquitectura del software es la que determina en gran medida los atributos de calidad que va a exhibir éste, tales como mantenibilidad, portabildad, seguridad, interoperabilidad, performance, etc. Su diseño, por lo tanto, es reconocido como uno de los factores que más influye en el éxito del software, sobre todo en el caso de aquellos de gran tamaño y/o complejidad. Tener una arquitectura definida antes de construir un software, permite hacer un análisis acerca de sus propiedades y tomar decisiones correctivas, si es que no se están satisfaciendo los requisitos de calidad; lo que es en general prácticamente imposible después de construido el sistema.

Existe consenso acerca de la conveniencia de especificar la arquitectura del software usando múltiples vistas donde se aborde el diseño desde distintos puntos de vista: módulos de código, componentes de software ejecutable, nodos de hardware donde se instalará el software, etc. Cada vista permite analizar diversos tipos de propiedades. Por ejemplo, una vista de módulos de código

permite analizar la reusabilidad de los módulos o la mantenibilidad de los mismos, pero no analizar performance. Asimismo, una vista de componentes y conectores admite analizar performance, escalabilidad o seguridad pero nada nos dice acerca de la portabilidad, ya que no incluye elementos de hardware. Esto puede analizarse en una vista de emplazamiento (deployment), donde se especifica en qué nodo de hardware se instalará cada componente de software.

A pesar de que existe una gran cantidad de conocimiento acerca de cómo diseñar una buena arquitectura, se la sigue identificando como una actividad difícil y desafiante, en la que en general solamente los más experimentados tienen éxito.

La reutilización de software es una estrategia positiva para su desarrollo, tanto por el aumento de la productividad que conlleva como por la calidad que componentes ya probados proporcionan al nuevo software. Reutilizar el conocimiento de diseño arquitectónico es por lo tanto una estrategia promisoria en el desarrollo de software, sobre todo teniendo en cuenta que la arquitectura se construye como un conjunto de decisiones de diseño esenciales que se toman respecto del sistema [1]. Pero reconocer que la reutilización, tanto de componentes como de conocimiento, es una buena estrategia que no resuelve de por sí la mecánica de llevarlo a cabo. En este artículo mostramos cómo podemos codificar las decisiones arquitectónicas de tal modo de poder aplicar distintas combinaciones de ellas, dependiendo de las necesidades de cada caso.

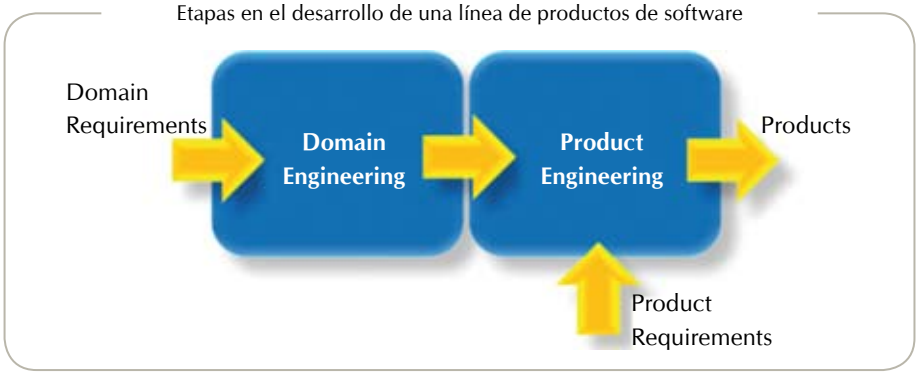
LÍNEAS DE PRODUCTOS DE SOFTWARE

Las líneas de productos de software (SPL por su acrónimo en inglés) son conjuntos de productos dirigidos a un mismo nicho de mercado, construidos a partir de un conjunto administrado de activos de software [2]. Dichos activos son elementos reutilizables y recombinables para dar lugar a distintos productos de la línea. Es así, por ejemplo, como se pueden construir diferentes productos ya sea con funcionalidad extendida o limitada, o bien productos similares que se pueden ejecutar en distintas plataformas. Los activos de software más comunes son los componentes de software. Pero también podemos encontrar otros, tales como conjuntos de prueba, manuales de usuario, documentos de requisitos o diseños de interfaces gráficas.

Desde un punto de vista metodológico, una línea de productos se desarrolla esencialmente en dos etapas: en la Ingeniería del Dominio, donde se construyen todos los activos reutilizables. Y en la Ingeniería del Producto, o de la Aplicación, donde se construyen los distintos productos de la SPL combinando los elementos construidos en la otra etapa. Esto se describe en la Figura 1.

Figura 1

Etapas en el desarrollo de una línea de productos de software



Tanto la Ingeniería del Dominio como la del Producto se dividen a su vez en Análisis, Diseño e Implementación del Dominio y del Producto, respectivamente. Cada una de estas etapas tiene un objetivo y una serie de productos para ser desarrollados. La idea central de las SPL es construir los elementos de la Ingeniería del Dominio de tal forma que después construir cada producto resulte ser sencillo, rápido y con una muy baja probabilidad de cometer errores.

ANÁLISIS DE DOMINIO

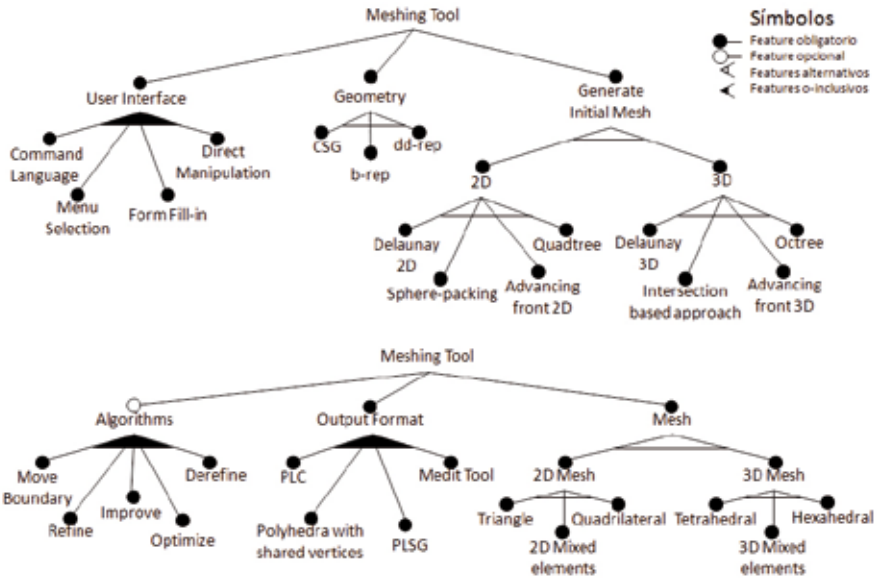
En el Análisis del Dominio se identifican los requisitos de toda la SPL. O sea, es una generalización del análisis de requisitos que se da en un desarrollo tradicional de software. Aquí se determinan cuáles requisitos son comunes a todos los productos, y cuáles son variables, y se los documenta como un modelo del dominio. En esta etapa también se define el alcance de la SPL, es decir, qué productos serán desarrollados y cuáles no.

Todos los productos de una misma línea comparten una serie de características y/o funcionalidades, que se denominan habitualmente *commonalities*. También existen elementos variables denominados *variabilities*. Los elementos variables pueden a su vez ser opcionales (es posible que estén presentes o no en un producto). O bien, ofrecer una serie de alternativas entre las cuales es posible elegir la característica que se desea que tenga el producto. Una de las técnicas más usadas para la especificación de *commonalities* y *variabilities* son los diagramas de características *(features)* [6]. En la Figura 2 mostramos un diagrama de *features*, donde identificamos las características comunes y variables de una línea de productos para el manejo de mallas geométricas.

El diagrama indica, por ejemplo, que todo producto de la SPL de mallas geométricas tendrá necesariamente una *UserInterface* (elemento obligatorio), que podrá tomar uno o cualquier subconjunto de sus subcaracterísticas, es decir, mediante menús, líneas de comandos, manipulación directa, etc. Así también se ve que una herramienta de mallas puede tener o no *Algorithms* (elemento opcional). Pero si existen algoritmos, estos deberán ser elegidos de entre el conjunto de subcaracterísticas indicadas. Por otra parte, todo producto de mallas tendrá obligatoriamente una forma de

Figura 2

Diagrama de features de una SPL de mallas geométricas



representar la geometría de los elementos modelados *(Geometry)*. Y esta será una y solamente una de las subcaracterísticas, o sea CSG, b-rep ó dd-rep en este caso.

DISEÑO DEL DOMINIO

Uno de los activos esenciales de una SPL es lo que se llama la arquitectura de la línea de productos (PLA, por su acrónimo en inglés). Su importancia radica en que todos los productos de la línea comparten la misma arquitectura y, por lo tanto, el impacto de tener una buena o mala arquitectura no se limita a un producto, sino a toda la línea.

El objetivo del Diseño del Dominio es construir la arquitectura de la línea de productos o PLA. Para su diseño se debe tener en cuenta el modelo de dominio como el súper conjunto de requisitos que esta arquitectura debe satisfacer. Para ello se aplican los patrones habituales usados en el diseño de arquitecturas de software [7]. Pero también se deberá indicar cuáles componentes son comunes a todos los productos y cuáles son variables. Esto hace que los lenguajes más habituales para descripción de arquitecturas no sean directamente aplicables. Existen algunos lenguajes específicos para individualizar PLAs tales como xADL, pero son complejos y su uso no está muy difundido. También es posible usar UML definiendo estereotipos para indicar los componentes variables.

Es evidente que, si es difícil diseñar una buena arquitectura para un producto de software, lo es más aún diseñar una que satisfaga las necesidades de toda una serie de productos con requisitos variables. Y hasta algunos productos que aún no han sido imaginados. Este problema es todavía más radical cuando los distintos productos varían en sus atributos de calidad. Por ejemplo, cuando un producto está dirigido a ser ejecutado en un ambiente local y otro en una red de computadores, claramente sus arquitecturas de software serán diferentes.

En el contexto de las líneas de productos de software es donde nuestra propuesta de construir la arquitectura del software de manera incremental adquiere real relevancia, ya que no es factible construir una arquitectura para cada producto por el costo que esto conllevaría. Y no es apropiado usar una única arquitectura para productos con requisitos diferentes. La idea entonces es asociar los incrementos de construcción de la arquitectura a los requisitos definidos para cada producto. Estas definiciones podrán ser reutilizadas cada vez que este requisito aparezca en un producto de la línea.

INGENIERÍA DIRIGIDA POR MODELOS

La ingeniería dirigida por modelos es un nuevo paradigma de desarrollo de software, donde el principal concepto es el de modelo y la principal operación es la transformación de modelos [3]. Un modelo es una abstracción o representación formal de un sistema. Y una transformación es una manipulación del modelo para obtener un nuevo modelo. Es necesario, por lo tanto, contar con lenguajes formales para la definición y transformación de modelos. UML es uno de los lenguajes más usados para la definición de modelos, aunque también es común definirlos en XML. Todo modelo deberá definirse siguiendo reglas de buena formación para ese modelo, que indican qué elementos pueden incluirse y las relaciones posibles entre los mismos. Estas reglas se definen usando metamodelos. En el caso de UML, esto está determinado por el metamodelo de UML y en el caso de XML por el esquema que se esté usando.

Además de estos lenguajes de propósito general es posible definir lenguajes específicos de dominio, donde los elementos de modelamiento tengan un alto nivel de abstracción y muy rico significado. Por ejemplo, el modelo de dominio en el contexto de RUP está orientado a objetos de alto nivel de abstracción, de un sistema donde los elementos son clases y las relaciones entre estos son herencia, dependencias y agregaciones. De forma similar, el modelo relacional es un modelo abstracto de los datos persistentes de un sistema, donde los elementos de modelamiento son tablas y sus columnas y las relaciones son las claves foráneas.

Los lenguajes de transformaciones más populares son declarativos basados en reglas tales como QVT y ATL. Pero las transformaciones también pueden escribirse usando lenguajes imperativos tales como C o Java. Estos lenguajes determinan qué

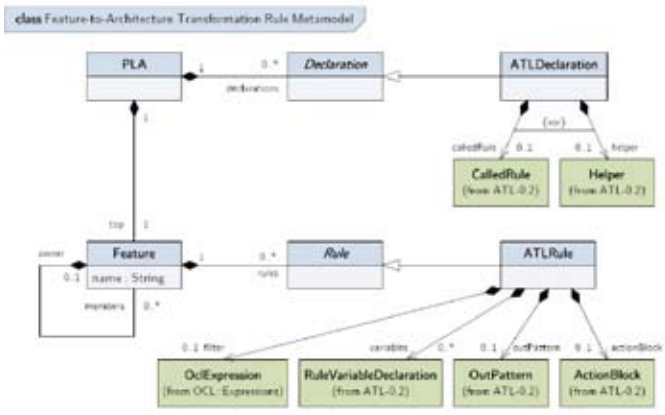
elementos del modelo de entrada se van a transformar en cuáles elementos del o los modelos de salida.

En el caso de la SPL de mallas geométricas hemos usado técnicas de MDE para construir un modelo formal del dominio mediante un modelo de *features* y asociar transformaciones a cada *feature*, de tal forma de ser capaz de construir automáticamente la PLA. En concreto asociamos una regla o un conjunto de reglas ATL a cada *feature*, de modo que, al configurar un producto particular de la línea de productos, la transformación resultante sea la conjunción de todas las reglas asociadas a todos los *features* seleccionados. Esta transformación construye un modelo de la arquitectura del producto a partir de un modelo de configuración de *features*, o sea, un modelo de *features* donde las variabilidades ya han sido resueltas.

La Figura 3 muestra el metamodelo que describe la transformación que construye incrementalmente la PLA, es decir, el producto del Diseño del Dominio en nuestra propuesta. La PLA está formada por una serie de declaraciones y un *feature* raíz. Cada declaración general puede ser usada por las reglas asociadas a cada *feature*. Los *features* tienen nombres que los diferencian y los organizamos en una estructura de árbol inspirada en el modelo de *features*. El nombre de cada uno se usa para asociarlo con el *feature* del modelo de configuración que será la entrada para seleccionar los *features* presentes en un producto particular. El conjunto de reglas asociadas a cada *feature* indica cómo se afectará la arquitectura del producto particular que incluya ese *feature*. Las clases Declaración y Rule son abstractas para lograr portabilidad; el metamodelo puede especializarse para distintos lenguajes de transformaciones; en la ilustración se muestran las clases necesarias para una implementación en ATL.

Figura 3

Metamodelo de la transformación de features a arquitectura



La Figura 4 muestra un ejemplo de regla ATL asociada al *feature* raíz del diagrama de *features* de la SPL de mallas geométricas. En la línea 2 se indica que f corresponderá a cada elemento de *Algorithms* elegido como parte del modelo de configuración de *features*. Aquí se indican los patrones de arquitectura que regirán el diseño arquitectónico de todos los productos: una combinación de tres capas donde las dos inferiores siguen a su vez el patrón de datos compartidos [7]. La regla indica que habrá un componente c en la arquitectura del producto (líneas 6 a 10), con el mismo nombre que el *feature* y cuyos subcomponentes serán aquellos generados por las reglas correspondientes a los *subfeatures* de f (línea 7). Los conectores de c son los definidos en esta regla. En la Figura 4 se incluyen dos ejemplos de conectores: uno que conecta los subcomponentes *UserInterface* y *Geometry* (líneas 11 a 16) y varios conectores entre *UserInterface* y cada interfaz provista por el componente *Algorithms* (líneas 17 a 23).

ARQUITECTURA COMO COMPOSICIÓN DE TRANSFORMACIONES

Tal como dijimos anteriormente, el objetivo de las SPL consiste en tener una etapa de Ingeniería del Dominio donde se desarrollan los activos, de tal modo que la Ingeniería del Producto sea un proceso rápido y sencillo que consista simplemente en la combinación de los activos ya desarrollados. Tal como lo expusimos, los activos desarrollados como parte de nuestra propuesta son: el modelo de *features*, desarrollado durante el Análisis del Dominio, y el conjunto de reglas que construyen la PLA, desarrollado durante el Diseño del Dominio.

Mostraremos ahora cómo se puede construir la arquitectura de un producto particular de manera automática, una vez que se seleccionan los *features* que deseamos que formen parte de ese producto. El cuadro explica los principios generales que inspiran nuestra propuesta [4].

Figura 4

Regla ATL asociada al feature Meshing Tool

```
1  ATLRule for 'Meshing Tool' {
2    filter f.members->select (fi | fi.name = 'Algorithms')->notEmpty();
3    variable lnames : Sequence (String) = thisModule.getAlgorithmFeatures(f)
4                                          ->collect(fa | fa.name)->asSequence();
5    out {
6      c : PAMMIComponent {
7        name <- f.name, components <- f.members,
8        connectors <- Set{xgeometry, xgenerate, xgeneratemesh, xoutput, xoutputmesh, xalg, xalgmesh},
9      },
10     xgeometry : PAMMIConnector {
11       name <- 'Geometry', kind <- #Assembly,
12       source <- c.components->any(ci | ci.name = 'User Interface').required->any(ii | ii.name = 'IGeometry'),
13       target <- c.components->any(ci | ci.name = 'Geometry').provided->first()
14     },
15     xalg : distinct PAMMIConnector foreach (lname in lnames) {
16       name <- lname, kind <- #Assembly,
17       source <- c.components->any (ci | ci.name = 'User Interface').required->any(ii | ii.name = 'I' + lname),
18       target <- c.components->any (ci | ci.name = 'Algorithms').provided->any(ii | ii.name = 'I' + lname)
19     },
20     --
21   }
22 }
```

- Los *features* representan funcionalidad – los modelos de *features* son una forma estándar de documentar modelos de dominio en líneas de productos de software, como una forma clara de capturar *commonalities* y *variabilities*. Restringimos los diagramas de *features* para que incluyan solamente funcionalidad, servicios, parámetros o almacenamientos de datos. Suponemos que los requisitos de calidad se documentan separadamente en otro artefacto. Las decisiones de diseño que el arquitecto tome durante el Diseño del Dominio se asocian a los *features* (funcionales) pero tomando en cuenta los requisitos de calidad.
- Los *features* guían la construcción de la arquitectura – la principal tarea que se aborda en el Diseño del Dominio es la construcción de la PLA, que encarna las decisiones críticas que resuelven tanto los requisitos funcionales como de calidad. Así como identifica *commonalities* y *variabilities* a nivel de arquitectura. En nuestra propuesta organizamos estas decisiones de acuerdo con los *features* del modelo de *features*, los cuales a su vez guían la estructura composicional de los componentes arquitectónicos. Cada *feature* inspira una serie de decisiones de diseño que guían la construcción de la parte de la arquitectura que incluye ese *feature*. Las decisiones solamente afectan un *feature* y sus subordinados. De esta forma, las decisiones relativas a los atributos de calidad se encuentran en general asociadas a los *features* cercanos a la raíz, mientras que las decisiones relativas a la funcionalidad se encuentran más cercanas a las hojas del modelo de *features*. No consideramos relaciones tales como *requires* o *excludes* en este trabajo, pues son parte de trabajos futuros.
- No se documenta la arquitectura sino el proceso de diseño arquitectónico – en un enfoque tradicional, en el Diseño del Dominio se desarrolla la PLA obteniendo una estructura sofisticada especificada con lenguajes de descripción de arquitecturas no estándar. En el Diseño del Producto todas las *variabilities* de la PLA se deben resolver de modo de obtener la arquitectura del producto particular (PA), mientras que el Análisis del Producto resuelve las *variabilities* a nivel de *features*. El trabajo está no sólo duplicado, sino que puede prestarse a inconsistencias si no se controla una estricta trazabilidad entre *features* y componentes. Si se parte de una PLA, necesariamente no se tiene un registro explícito de la motivación (*rationale*) detrás de cada decisión de diseño, haciendo más difícil la posterior modificación de esta PLA si fuese necesario. En nuestro enfoque cada decisión está explícitamente encapsulada como una transformación. La arquitectura de un producto será entonces la conjunción de decisiones de diseño asociadas a cada *feature* escogido.
- Desarrollo incremental de la SPL – durante el Análisis de Dominio se construye un diagrama de *features*, y durante el Diseño de Dominio se suele construir una PLA que tenga en cuenta todos los requisitos en términos de *features* comunes y variables, así como requisitos de calidad. Esto hace que la construcción de la PLA sea especialmente compleja por la cantidad y diversidad de factores a tener en cuenta. En nuestro enfoque es posible diseñar solamente las transformaciones asociadas a los *features* que son incluidos en el producto en desarrollo (todos los *commonalities* y solamente las *variabilities* que han sido escogidas). Y solamente cuando se incluyan otros *features*, en desarrollos futuros será necesario sumar el diseño arquitectónico asociado a esos *features* en forma de nuevas transformaciones. Esto no sólo favorece la incrementalidad sino también la evolucionabilidad.

ANÁLISIS DE PRODUCTO

Como parte del Análisis del Producto es necesario tomar una decisión acerca de los *features* que fueron definidos como variables durante el Análisis del Dominio. La Figura 4 muestra los elementos escogidos para un producto particular configurados en FeaturePlugin [8]. Tendremos una herramienta cuya UserInterface será un lenguaje de comandos, la geometría se representará como CSG, habrá un componente para generar la malla inicial 2D de quadtrees. Los algoritmos disponibles serán de refinamiento, mejora y optimización, y los formatos de salida serán PLC y PLSG y la malla internamente será de triángulos en 2D.

Nótese que no es necesario tomar decisiones acerca de los *features* que fueron identificados como comunes a todos los productos, ya que siempre aparecerán. En consecuencia, la tarea a llevar a cabo durante el Análisis del Producto se reduce a resolver las variabilidades.

DISEÑO DEL PRODUCTO

El objetivo del Diseño del Producto es generar la arquitectura del producto particular que se desea construir. Para ello contamos con el modelo de configuración de *features* ya definido durante el Análisis del Producto y con las transformaciones asociadas a cada uno de los *features* que fueron construidas durante el Diseño del Dominio. La tarea en esta etapa será entonces sólo ejecutar la conjunción de todas las transformaciones que darán como resultado la arquitectura del producto de manera completamente automática.

La Figura 6 muestra la arquitectura resultante de aplicar las transformaciones asociadas a los *features* escogidos y mostrados en la Figura 5. Nótese, por ejemplo, que la componente de *Algorithms* tiene tantas interfaces como *subfeatures* hayan sido elegidos, lo cual hace que la arquitectura del producto sea distinta si la configuración de *features* realizada durante el Análisis del Producto hubiese sido otra.

CONCLUSIONES

Las líneas de productos de software son una estrategia apropiada para el desarrollo de software de calidad en nichos de mercado definidos, así como las técnicas de MDE se muestran muy promisorias para hacer operativas las tareas que deben llevarse a cabo. Es así como, definiendo lenguajes

específicos para cada uno de los modelos que representan los artefactos que se producen en cada etapa, somos capaces de sistematizar la Ingeniería del Dominio de tal modo que la Ingeniería del Producto pueda realizarse en forma automática, obteniendo así aún más beneficios del enfoque de líneas de productos.

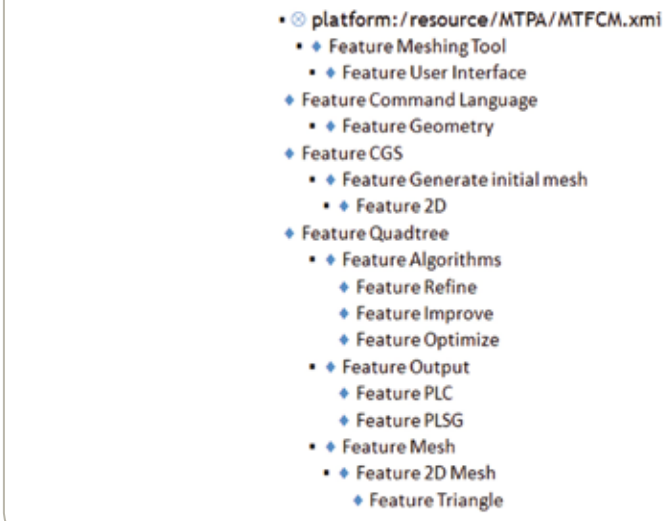
Si bien en el trabajo presentado los requisitos de calidad se supone que se especifican como parte de otros documentos, estos tienen una incidencia decisiva en el diseño de la arquitectura y como tales deben ser tenidos en cuenta. Hemos experimentado también aplicando esta misma estrategia en contextos donde la variabilidad se da a nivel de atributos de calidad [5]. Aquí los resultados obtenidos son aún más relevantes, dado que una única PLA prediseñada es definitivamente no factible para la instanciación de productos, ya que productos con distintos requisitos de calidad requieren generalmente arquitecturas que sigan distintos patrones.

AGRADECIMIENTOS

Quisiera agradecer fundamentalmente a Pedro Rossel y Daniel Perovich, coautores de los artículos [4] y [5] y estudiantes de doctorado del DCC. También a la profesora del DCC Nancy Hitschfeld por proporcionar el ejemplo ilustrativo de la línea de productos de mallas geométricas.BITS

Figura 5

Modelo de configuración de features de la SPL de mallas geométricas



REFERENCIAS

[1] Richard N. Taylor, Nenad Medvidovic and Eric Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, January 2009.

[2] Paul Clements, Linda Northrop. Software Product Lines: Practices and Patterns, Addison-Wesley Professional; 3rd edition, August 30, 2001.

[3] Douglas C. Schmidt. Model-Driven Engineering. IEEE Computer, 39(2):25–31, February 2006.

[4] Daniel Perovich, Pedro O. Rossel and María Cecilia Bastarrica. Feature Model to Product Architectures: Applying MDE to Software Product Lines. Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture 2009, WICSA/ECSA'2009, pages 201 – 210, Cambridge, UK, September 2009.

[5] Pedro O. Rossel, Daniel Perovich and María Cecilia Bastarrica. Architectural Knowledge Evolution and Reuse in SPL Development. 11th International Conference on Software Reuse, ICSR 2009, pages 191 – 200, September 27 - 30, 2009, Falls Church, Virginia, USA.

[6] Kyo C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA). Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture. A System of Patterns. John Wiley & Sons, 1996.

[8] M. Antkiewicz and K. Czarnecki. FeaturePlugin: Feature modeling plug-in for Eclipse. In OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, 2004.

Figura 6

Arquitectura de un producto de la SPL de mallas geométricas