Un Modelo de Metadatos para la Gestión de la Variabilidad en Líneas de Productos de Software

Matías Pol'la[1,2], Agustina Buccella[1,2], Alejandra Cechich[1], and Maximiliano Arias[1,2]

[1] Giisco, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina
[2] Concejo Nacional de Investigaciones Científicas y Técnicas, Universidad Nacional del Comahue, Neuquén, Argentina
matias.polla@fi.uncoma.edu.ar,agustina.buccella@fi.uncoma.edu.ar, alejandra.cechich@fi.uncoma.edu.ar,maximiliano.arias@fi.uncoma.edu.ar

Abstract. La gestión de la variabilidad conforma un área de estudio altamente investigada en la actualidad. La misma, aplicada al paradigma de la ingeniería de líneas de productos, provee un conjunto de técnicas y métodos que permiten configurar, adaptar y/o extender los servicios provistos por una línea para que formen parte de los nuevos productos a ser derivados. Dentro de las nuevas propuestas en la literatura, existen aquellas que proveen soluciones para diferentes aspectos involucrados en dicha gestión. En este trabajo, presentamos un modelo para la gestión de la variabilidad basado en metadatos que son implementados mediante un sistema de anotaciones. La definición de este modelo permite definir diferentes configuraciones ya sea a través de composición de componentes o simplemente realizando pequeños ajustes en el código. A su vez, hemos ilustrado la aplicación del modelo mediante el uso de una SPL previamente desarrollada en trabajos previos sobre el dominio de ecología marina.

Keywords: Gestión de la Variabilidad, Línea de Productos de Software, Metadatos, Anotaciones

1 Introducción

Dentro de la ingeniería de desarrollo de software, el reuso es una característica importante que busca ser maximizada a la hora de encarar tanto el diseño como la implementación de nuevos sistemas; en especial para aquellos cuyo conjunto de funcionalidades que ofrece y tipos de datos utilizados provocan un gran esfuerzo al momento de ser implementados. Existe, en este sentido, un gran numero de investigaciones y tecnologías que apuntan a reducir el esfuerzo y tiempo en el desarrollo de software, centrándose en el reuso de artefactos de software ya implementados. Tal es el caso, del Desarrollo de Software Basado en Componentes [1, 2] (DSBC) y la Ingeniería de Lineas de Producto de Software [3, 4].

La Ingeniería de Lineas de Productos de Software (ILPS) es un paradigma de desarrollo de software compuesto por un grupo de sistemas similares pertenecientes a un dominio, que poseen un conjunto de funcionalidades y características comunes. Una Línea de Productos de Software (SPL por sus siglas en ingles) tiene como principal objetivo proveer una plataforma común que permita la derivación de sistemas particulares. Para lograr esto, la misma debe ser lo suficientemente flexible para poder adaptarse a las diferentes necesidades requeridas.

La flexibilidad de una SPL se logra a través de la configuración de la variabilidad, que es la capacidad de un artefacto de software de ser configurado, adaptado, extendido o cambiado en un contexto especifico [5]. Éste es un concepto clave en las Líneas de Producto de Software ya que permite adaptar la plataforma a situaciones particulares de acuerdo a las necesidades de los clientes al momento de derivar un producto. Una SPL está formada por un conjunto de puntos variantes, definidos en [4] como la representación de la variabilidad dentro de un artefacto o componente, junto con la información para determinar el contexto. Estos puntos están compuestos por un conjunto de variantes que representan las posibles configuraciones asociadas a la variabilidad. La gestión de variabilidad [4] involucra todas las actividades relacionadas con la variabilidad a lo largo de todo el ciclo de vida de una SPL, desde la especificación de requerimientos, modelado e implementación, hasta el testing de los productos derivados. Entre las mismas podemos incluir a la definición de la variabilidad junto con los puntos variantes, la gestión de los componentes variables y la resolución de la misma al momento de realizar el proceso de derivación.

En la literatura existen varias investigaciones y desarrollos que intentan abordar la problemática de la gestión de la variabilidad utilizando diversos enfoques de acuerdo a la etapa del ciclo de vida [6–9]. Esta cantidad y variedad de propuestas indica que todavía no existe un modelo estándar y único para la gestión de la variabilidad ya que los enfoques presentan un abanico extenso de soluciones. Entre ellos podemos diferenciar dos grupos. Por un lado, están las propuestas que defienden la representación jerárquica de la variabilidad [10, 11] a través del uso de componentes, intentando obtener un sistema escalable, reusable y flexible. Por otro lado, tenemos las propuestas que critican estos enfoques justificando que no son lo suficientemente flexibles como los sistemas que representan la variabilidad a nivel de modificaciones directamente sobre el código [12, 13].

Teniendo en cuenta estos diferentes enfoques, en este trabajo proponemos un enfoque mixto que busca potenciar los beneficios del desarrollo basado en componentes utilizado en el modelo jerárquico, y permitiendo a su vez, generar adaptaciones simples en el código. Así, hemos desarrollado un modelo de metadatos que permite gestionar la variabilidad a nivel de código con el fin de implementar las variabilidades y puntos variantes definidos. De esta forma, podemos proveer un soporte adecuado para el proceso de derivación de productos. Tanto la defincón como su aplicación y uso son descriptos en este artículo y aplicados a una SPL desarrollada en trabajos previos [14–16]. Además, presentamos una herramienta prototipo implementada para llevar adelante el proceso de configuración previo a la derivación de productos de la SPL previa. Ésta ha sido construída mediante

un enfoque orientado a dominios, y aplicada específicamante al subdominio de ecología marina. La SPL ha sido desarrollada utilizando componentes implementados en Java, lo que permite contar con una estructura jerárquica que posibilita tanto el reuso efectivo como la creación e integración de nuevos componentes. La misma cuenta con una plataforma de servicios comunes de la cual ya han sido derivados dos productos particulares [14–17].

Este artículo está organizado de la siguiente manera. A continuación se analizaran diferentes enfoques que abordan la gestión de variabilidad a nivel de implementación. En la Sección 3 se presentan nuestros antecedentes en cuanto a la SPL previamente desarrollada y la motiviación para gestionar la variabilidad de una manera eficiente. En la Sección 4 se detalla el modelo de metadatos desarrollado, el cual permite una representación de la variabilidad en una SPL, junto con una herramienta que soporta el proceso de derivación de productos. Por último se enumeran las conclusiones obtenidas y los trabajos futuros.

2 Trabajos Relacionados

La gestión de la variabilidad es una problemática de gran interés en la actualidad. Esto se ve reflejado en la gran cantidad de investigaciones existentes en el tema. Existen así varias revisiones sistemáticas en la literatura [6, 7] donde se pueden observar diferentes enfoques utilizados en el desarrollo de software en general, ya que la variabilidad no sólo forma parte de las Líneas de Producto de Software. Por ejemplo, se utiliza en paradigmas tales como lo son los Sistemas Auto Adaptativos [18], los Sistemas Basados tanto en Servicios como en Servicios Web [8, 9], etc. En particular, en [7] se realiza una revisión sistemática centrándose en la gestión de variabilidad utilizada en el contexto de la ingeniería de líneas de productos. En la misma se pueden observar diferentes enfoques que abordan la variabilidad en diferentes momentos del ciclo de desarrollo de software como por ejemplo el modelado de la variabilidad, el soporte para la identificación de variabilidades y partes comunes de una SPL, el diseño de la arquitectura, la derivación de un producto, herramientas de soporte, entre otras. Dentro de los trabajos pioneros en variabilidad, podemos citar a FODA (Análisis de Dominio Orientado a Funcionalidades) [19], el cual intenta capturar las características principales de un conjunto de sistemas similares pertenecientes a un dominio específico, diferenciando entre las funcionalidades que son compartidas por todos los sistemas y las variabilidades. Otro trabajo, surgido a partir de FODA, es el Modelado de Funcionalidad (FM-Feature Modeling) [10] que presenta un modelo de aplicación que permite definir tanto las variabilidades presentes como las partes comunes para soportar la derivación de los productos. En este sentido, FM presenta un modelo jerárquico, organizado en forma de árbol, que posibilita descomponer funcionalidades de niveles superiores en funcionalidades detalladas y específicas. Esta jerarquía impone ciertas restricciones de configuración al momento de llevar adelante la derivación de los productos, ya que la selección de una funcionalidad implica la selección del padre que la engloba. En [11] también se presenta un modelo de variabilidad jerárquica apoyado en la utilización del

DSBC, debido a que este paradigma reduce la complejidad del diseño y reuso de software, distribuyendo el trabajo y la implementación del sistema en el desarrollo de diferentes componentes independientes. Una característica importante que presenta este trabajo es la distinción entre componentes y subcomponentes variables. Aquí, se sostiene que la variabilidad debe ser representada y definida localmente en el componente variable, lo que favorece el desarrollo independiente del mismo. Además permite indicar el punto variante en donde se utilizará el componente o subcomponente variable. Otro de los enfoques más utilizados es el Modelo de Decisión [10], que surge como un refinamiento del Método de Síntesis [20]. Este enfoque se basa en la aplicación de un conjunto de decisiones válidas para la generación de un producto perteneciente a una familia de productos. En contraste con FM, se centra sólo exclusivamente en la representación de la variabilidad para apoyar la selección de decisiones, dejando atrás el modelado de las partes comunes de los sistemas. Esto se debe a que en este enfoque lo primordial es el proceso de derivación que surge tras la elección de las diferentes decisiones. Puede presentar un modelado jerárquico, a través de tablas de dependencias, pero solo será utilizada para guiar el proceso de decisión y no para restringir el modelado de la variabilidad. Los enfoques jerárquicos nombrados anteriormente así como el modelo de decisión permiten desarrollar un modelo ortogonal. La ortogonalidad es el grado en que las funcionalidades y características que contienen variabilidad puedan ser modeladas e implementadas de manera independiente a otras funcionalidades.

Los enfoques previamente presentados permiten implementar la variabilidad en artefactos de software separados a través de la utilización de componentes. Esto favorece el desarrollo, mantenimiento y trazabilidad del sistema, ya que permiten desarrollar una SPL como un conjunto de artefactos de software independientes encargados de cumplir funcionalidades particulares. Sin embargo, estos enfoques no tienen en cuenta diferentes problemáticas que pueden surgir a la hora de ser implementados, como por ejemplo que la instanciación de una cierta variabilidad implique la inserción de código en diferentes unidades del código, o no poseer la capacidad de introducir adaptaciones de granularidad fina que posibiliten por ejemplo cambiar unas pocas lineas de código, etc.

Estas problemáticas generaron el surgimiento de otros enfoques diferentes que abordan la implementación de la variabilidad directamente sobre el código de un sistema, sin la necesidad de utilizar artefactos de software independientes. Por ejemplo, los enfoques anotativos presentan una técnica sencilla de anotaciones sobre el código, que posibilita modelar la variabilidad permitiendo modificaciones de granularidad fina. Sin embargo, esta solución carece de ortogonalidad y trazabilidad, es decir, genera un diseño no modular difícil de mantener y modificar; y además resulta imposible determinar que porción de código implementa determinada variabilidad. Existen así otras investigaciones que intentan solucionar estos dos problemas que presentan los enfoques anotativos. Por ejemplo, en [12] se ha implementado una herramienta que apoya la derivación de un producto de una SPL a través de modificaciones en el código utilizando las ventajas del enfoque anotativo. Otro enfoque que busca solucionar dichos inconvenientes es la

Programación Orientada Delta (POD) [13, 21, 22]. En ella se representa la SPL por medio de un módulo núcleo y un conjunto de módulos delta. El primero define la plataforma común a todos los productos que pueden ser derivados de la línea, mientras que los "Deltas" especifican las posibles adaptaciones (variabilidades) para ajustarse a las necesarias particulares de cada producto. Cada delta genera cambios en el módulo principal (núcleo) a nivel de código tanto agregando, modificando o eliminando el mismo. Esta implementación para SPL es más flexible y menos restrictiva que el modelado de funcionalidades (FM) y otros enfoques modulares, ya que permite realizar modificaciones simples en el código. Además si se realiza una cuidadosa y detallada especificación y documentación de la línea, ésta puede brindar soporte para la trazabilidad de las variabilidades definidas, identificando qué delta implementa determinada funcionalidad.

En nuestro trabajo, considerando los trabajos mencionados y de acuerdo a las particularidades de la SPL previamente desarrollada, se decidió realizar un modelo de metadatos para gestionar la variabilidad. El mismo presenta un enfoque mixto que intenta maximizar los beneficios tanto de los enfoques jerárquicos así como los que producen modificaciones directamente sobre el código. Esto posibilita mantener el modelo jerárquico y generar adaptaciones de granularidad fina a nivel de código, permitiendo una mayor flexibilidad y reduciendo el costo de derivación.

3 Antecedentes

En trabajos previos [14–16] hemos presentado el desarrollado de una SPL dentro del dominio geográfico. La creación de la misma se ha realizado en conjunto con dos organizaciones, el IBMPAS[3] y el CENPAT-CONICET[4], que trabajan en el subdominio de ecología marina y que nos han proporcionado los requerimientos necesarios para llevar a cabo sus actividades diarias. En particular, ambas organizaciones recopilan información de diferentes especies marinas a través de censos o campañas donde se realizan diferentes muestreos. En este caso, se realizan diferentes investigaciones centradas en especies que habitan en los golfos San Matías, Nuevo y San Jorge (Patagonia - Argentina). El conjunto de funcionalidades definido en la SPL está dividido en dos grupos diferentes. Por un lado, tenemos aquellas funcionalidades comunes a los posibles productos resultantes que conforman la plataforma de servicios de la línea, junto con las variabilidades que permiten adaptarse a diferentes configuraciones particulares de cada producto. Por otro lado, cada producto derivado de la línea está conformado por un conjunto de funcionalidades específicas y particulares, ya que cada producto derivado es un sistema que solo cuenta con las funcionalidades comunes y las variabilidades seleccionadas e instanciadas.

Dentro de los trabajos realizados y para incrementar el reuso y trazabilidad de la línea, se desarrolló una taxonomía de servicios [23] derivada de la especia-

[3] IMBPAS Instituto Biología Marina y Pesquera Almirante Storni - http://www.ibmpas.org/

[4] CENPAT-CONICET Centro Nacional Patagónico - http://www.cenpat.edu.ar/

lización de la ISO/DIS 19119[5] enfocándose en los servicios del subdominio de la ecología marina. Dichos servicios fueron utilizados para la migración del sistema a un enfoque de componentes de software [24] que permita la implementación, utilización y ensamblaje de los mismos en forma independientemente. Este enfoque nos permitió desarrollar la línea de manera incremental agrupando las funcionalidades en componentes independientes de acuerdo a los servicios que implementan. La SPL cuenta con un arquitectura de tres capas, respetando los estandartes utilizados para los sistemas de información geográfica. En la Figura1 se pueden observar dichas capas, las cuales involucran, la *Interfaz de Usuario, Procesamiento Geográfico* y *Modelo Geográfico*; y donde cada una de ellas está conformada por un grupo de componentes. De esta manera, cada uno de los componentes incluídos en la SPL se corresponde con un conjunto de servicios detallados en la taxonomía de servicios definida. Por ejemplo, en la figura podemos observar que el componente de *Característica de Visualización* implementa los servicios de *Manipulación de Mapas* (HI-MM) y el componente *Agregar Mapa* implementa el servicio *Manejo y Almacenamiento de Múltiples Mapas*(HI-LM6).

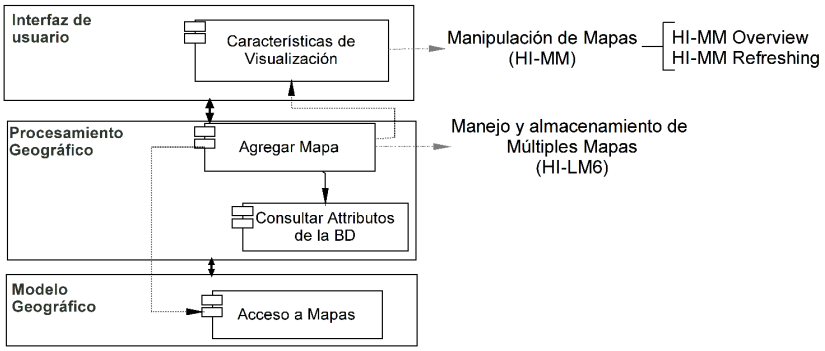

Fig. 1. Arquitectura y ejemplo de componentes y servicios de la taxonomía de la SPL previamente desarrollada

La implementación actual de la SPL fue realizada mediante una tecnología puramente basada en componentes utilizando Enterprise Java Beans [6] (EJB) junto con la herramienta Google Web ToolKit [7] (GWT) para ser utilizada bajo la Web. Esta tecnología e infraestructura desarrollada genera una estructura jerárquica que permite incrementar tanto el reuso efectivo de la plataforma, así como la modificabilidad de los componentes existentes y la creación e integración de los nuevos [16]. Por otro lado, la utilización de la herramienta GWT nos permite generar sistemas web a través de un proceso de compilación de un conjunto de componentes específicos implementados en Java, los que son traducidos a

[5] ISO/DIS 19119 : Geographic information Services, ISO/TS 2005
[6] Oracle, http://www.oracle.com/technetwork/java/javaee/ejb/index.html
[7] http://code.google.com/intl/es-ES/webtoolkit/

código JavaScript y HTML equivalente. De esta manera, se pueden distinguir dos tipos de componentes desarrollados. Por un lado, tenemos los componentes que conforman la capa de Interfaz de Usuario, implementados con GWT; y por otro lado los componentes EJB, implementados en lenguaje Java, que corresponden a la capa de Procesamiento Geográfico.

El proceso de derivación junto con la validación de la SPL como paradigma que soporta el reuso ha sido descripta en [17]. Aunque dicho proceso nos entregó resultados satisfactorios en cuanto a los productos generados a partir de la línea, el mismo generó una gran cantidad de trabajo manual para realizar los ajustes necesarios para ensamblar los componentes con las funcionalidades requeridas; es decir, fue compleja la instanciación de la variabilidad. Por ejemplo, para instanciar una variabilidad específica fue necesario realizar pequeñas modificaciones en diversos puntos del código, lo que implicó un gran conocimiento previo de la implementación realizada. Esta problemática provocó la necesidad de asistir el proceso de instanciación de los productos a través de la implementación de la variabilidad en la SPL. En la siguiente sección se detalla el modelo de variabilidad implementado.

4 Modelo para la Gestión de la Variabilidad

Para llevar a cabo el modelo de variabilidad se adaptaron algunas particularidades presentadas en [25] sobre la SPL previamente creada. Para esto hemos definido un modelo de metadatos que nos permite luego dar soporte a la gestión de la variabilidad. Así, el mismo consta de tres etapas: la definición, aplicación y uso de metadatos. A continuación se describen dichas etapas mostrándo como han sido aplicadas luego a la SPL previamente desarrollada.

4.1 Proceso de definición del Modelo de Metadatos

En este proceso definimos los datos necesarios para representar el modelo de metadatos propuesto. El mismo puede observarse en la Figura 2 la cual muestra tres metadatos principales: *los deltas, los puntos variantes y los tipos de variabilidad requeridos*. Los primeros son las modificaciones que sufre el código para posibilitar la instanciación correcta. Estos deben corresponderse con variantes externas, que son aquellas que son implementadas en componentes externos o subcomponentes, o con variantes internas asociadas a variabilidades implementadas en el mismo componente. Los segundos, representan el lugar donde debe instanciarse o seleccionarse una variante correspondiente a una determinada variabilidad, es decir, el punto configurable de acuerdo a las necesidades del usuario. Por último, los *tipos de variabilidad* son importantes ya que presentan diversas restricciones al momento de ser instanciados:

- ***Variabilidad Obligatoria:*** la instanciación de un determinado producto para el punto variante asociado no puede ser vacía.

- ***Variabilidad Opcional:*** la instanciación de las variabilidades definidas pueden estar o no presentes en el producto derivado. Este tipo de variabilidad, permite instanciar más de una de las opciones posibles.
- ***Variabilidad Alternativa:*** sólo es posible instanciar a lo sumo una de las alternativas presentadas.

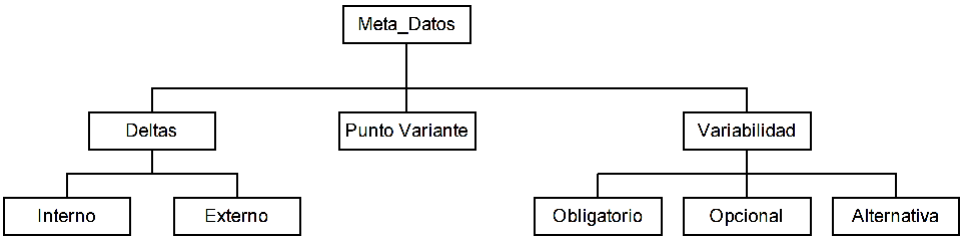

Fig. 2. Metadatos identificados

A continuación se determinó la información necesaria para representar cada uno de estos metadatos identificados. De esta manera, para los metadatos *Deltas*, tanto internos como externos, se almacena un *id* que lo identifica unívocamente, un *código* para invocar el método correspondiente, el conjunto de *librerías* utilizadas por el método junto con una *descripción* de la variante particular correspondiente al Delta. Además para los *Deltas Externos* se debe registrar el/los *componente/s* donde se implementa la variante correspondiente.

Luego, para cada uno de los metadatos de *Puntos Variantes* sólo es necesario indicar el *tipo de variabilidad*. Éste determina las restricciones al momento de realizar la configuración de cada punto variante.

Por último para los metadatos de *Tipos de Variabilidad* (*Opcional, Alternativa Obligatoria*), se almacenan el *código de servicio* determinado por la taxonomía de servicios [16], junto con una *descripción* del mismo. Además se registra un *conjunto de pares ordenados* de las posibles variantes junto con los *Deltas* que implementa cada una de ellas. En el caso de que el tipo de variabilidad sea *Obligatoria*, se debe indicar el delta por defecto.

Una vez determinada la información necesaria para cada metadato, implementamos la estructura para soportar el modelo de variabilidad. Debido a que la SPL ha sido desarrollada utilizando el lenguaje Java, la estructura del modelo de metadatos fue implementada con una herramienta propia del lenguaje, llamada Java Annotations[8]. Ésta permite insertar diferentes directivas a nivel de código y están estructuradas utilizando el carácter especial "@" junto con un nombre o etiqueta que la identifica. Las anotaciones cuentan con un conjunto de parámetros y presentan un objetivo y una política de retención. En nuestro caso utilizamos la política de retención "*source*" ya que sólo nos interesa mantener la anotación a nivel de código; y como objetivo solo utilizamos

[8] Oracle Corporation. http://docs.oracle.com/javase/tutorial/java/annotations/

"*method*" ya que para nuestra estructura necesitamos anotar métodos. Por ejemplo, en la Figura 3 se puede observar la implementación para la definición de la anotación *Meta_VariabilityOptional* correspondiente al metadato *Variabilidad Opcional* (Figura 2). En la misma se distinguen la política de retención utilizada y el objetivo junto con los parámetros correspondientes.

```
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.METHOD)

public @interface Meta_VariabilityOptional {
        public String serviceCode();
        public String serviceDescription();
        public String[] instance();
        public String[] deltaId();
}
```

Fig. 3. Implementación de Variabilidad Opcional utilizando Java Annotations

En la Figura 4 puede observarse la estructura del modelo de metadatos junto el sistema de anotaciones definidas. Esta estructura está dividida en dos grupos de anotaciones principales, los *Meta_VariantPoint* y los *Meta_Deltas* que se corresponden con los elementos *Punto Variante* y *Deltas* de la Figura 2. El primer grupo está formado por las anotaciones *@Meta_VariabilityOptional, @Meta_VariabilityMandatory y @Meta_VariabilityAlternativa*. Estas tres anotaciones son aplicadas para determinar los puntos variantes con su respectiva variabilidad. Cada una de ellas cuenta con los datos descriptos anteriormente. Cabe destacar que por limitaciones propias de la herramienta Java Annotations, el conjunto de pares (*Instancia, Delta*), el cual almacena las posibles variantes a través de la relación entre deltas e instancias, fue implementado por medio de dos arreglos simples de una dimensión, como puede observarse en la Figura 3.

El segundo grupo, formado por las anotaciones *@Meta_DeltaInternal y @Meta_DeltaExternal* son los encargados de representar las diferentes modificaciones que sufre la SPL a nivel de código para cada variabilidad implementada. La distinción entre variante Interna y Externa nos permitirá generar una herramienta de derivación de productos flexible, que denote un rango de configuraciones lo suficientemente amplio para abarcar tanto los pequeños cambios en el código (granularidad fina) así como la utilización y composición de componentes reusables (granularidad gruesa). A su vez, estas anotaciónes también se definen a nivel de métodos y están representadas por un Id (DeltaId) que lo identifica unívocamente, por el código para invocar el método (CallerCode), las librerías(CallerImports) y componentes necesarios para su ejecución (Components), además de una descripción específica del Delta (InstanceDescription) que indica las particularidades del mismo.

Como puede observarse los puntos variantes no son representados por medio de anotaciones ya que estos sólo indican el lugar puntual donde debe situarse

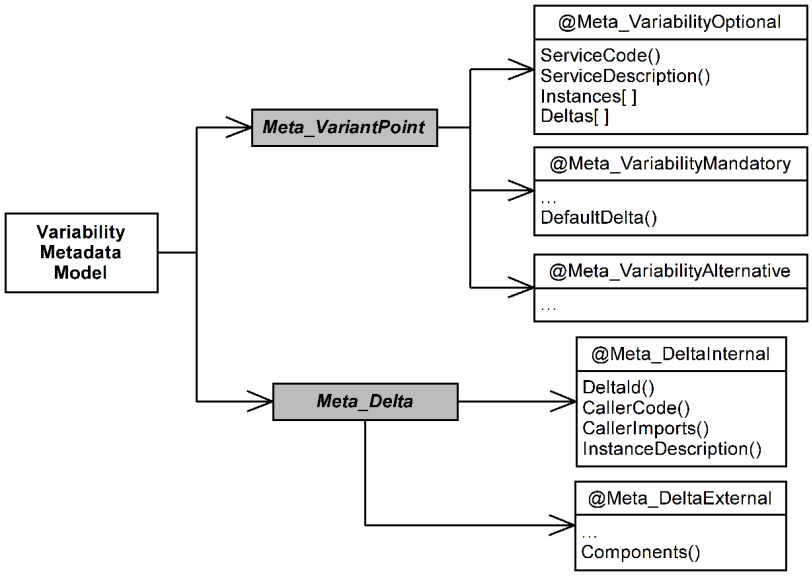

Fig. 4. Estructura de anotaciones para el modelo de metadatos para variabilidad

la variabilidad. En la siguiente sección se explica como son representados estos puntos a nivel de código.

4.2 Proceso de Aplicación del Modelo de Metadatos

En este proceso describimos como aplicar el modelo de metadatos mediante el sistema de anotaciones definido en la Sección 4.1 para la SPL previamente creada. El modelo fue diseñado para aplicarse a un conjunto de métodos específicos y así representar los distintos tipos de variabilidad.

Cada punto variante se define utilizando un método *dummy*[9], el cual sólo es usado para indicar el punto exacto en donde debe ser instanciada determinada variabilidad. La razón de esta decisión de diseño se debe a la restricción de la tecnología subyacente, ya que la versión de java utilizada (Java 1.6+) no permite insertar anotaciones en cualquier punto del código. Luego para cada método *dummy* se aplica una de las anotaciones pertenecientes a *Meta_VariantPoint* (Figura 4) de acuerdo al tipo de variabilidad necesaria para este punto variante. Por último para cada uno de los Deltas se utilizan diferentes métodos junto con la anotación correspondiente al tipo de delta aplicado (Interno o Externo). Estos métodos contienen el código necesario para utilizar una variante determinada. Una vez seleccionada esta variante, respetando las restricciones según el tipo de variabilidad asociada, el *método dummy* será reemplazado por la llamada al/los deltas correspondientes (CallerCode).

[9] Método inútil o Irrelevante (sin contedido)

A modo de ejemplo, podemos observar en la Figura 5 el código correspondiente a la aplicación del modelo utilizando la anotación definida para el tipo de variabilidad opcional (*@Meta_VariabilityOptional*). Aquí se visualiza el punto variante definido en el método *foo*, y el método *dummy* para representar un punto variante correspondiente a una variabilidad opcional (*puntoVatiante()*), en donde se define la variabilidad correspondiente al servicio HI-LM6 (*serviceCode*) de acuerdo a la taxonomía de servicios de la SPL previa. En la misma anotación podemos distinguir el conjunto de pares ("Instancia1","Delta1") y ("Instancia2","Delta2") que representan las posibles variantes para este punto. Además en el ejemplo se muestra la aplicación de la anotación *@Meta_DeltaExternal* para el método *agregarMapaDelta1()* en el cual se define una de las instancias. En éste se detallan los atributos correspondientes, particularmente indicando el componente externo *"Agregar_Mapa"* (*Components*).

```
public void foo(){
//Previous Code
        puntoVariante();
//Following Code
}
@Meta_VariabilityOptional (
    serviceCode = "HI-LM6",
    serviceDescription= "Manipulacion de Multiples Mapas",
    public String[] instance={"instancia1","instancia2"},
    public String[] deltaId={"Delta1","Delta2"})
public void puntoVatiante(){ ... }
@Meta_Delta_External(
    DeltaId = "Delta1"
    CallerImports = "...."
    CallerCode = "agregarMapaDelta1(...);"
    DeltaDescription = "Multiples Mapas mediante
                        division vertical de pantalla"
    Components = "Agregar_Mapa")
public void agregarMapaDelta1(.ParameterList_Delta1){...}
```

Fig. 5. Aplicación de la Variabilidad Opcional - Punto variante

La relación descripta entre deltas e instancias de las variabilidades permite definir diferentes deltas asociados a la misma instancia en diferentes puntos variantes dispersos en varios componentes. Esto posbilita realizar modificaciones en diferentes unidades de código asociadas a una determinada instancia.

4.3 Proceso de Uso del Modelo de Metadatos

Para poder hacer uso de los metadatos definidos en los componentes, hemos desarrollado una herramienta prototipo que asiste el proceso de derivación de productos de una SPL. La misma recibe como entrada la información recuperada por una herramienta de lectura de metadatos, previamente implemen-

tada en [25], la cual realiza el análisis[10] sobre diferentes clases ubicadas en los componentes, obteniendo la información de las anotaciones y sus respectivos parámetros. Luego, una vez obtenidos dichos metadados, la herramienta prototipo realiza un proceso de reestructuración interno de los mismos para ser mostrados al usuario. En la Figura 6, puede observarse la herramienta prototipo. A la izquierda, presenta las categorías y subcategorías definidas por la Taxonomía de servicios, y a la derecha, los servicios pertenecientes a cada categoría indicando el tipo de variabilidad correspondiente. Por ejemplo, en este caso se observa para la categoría *InteracciónHumana/VisualizadorGeográfico/Manipulación de Mapas* la variabilidad *Opcional* definida para el servicio *AgregarMapa* junto con sus variantes (*Clonar Mapa* y *Manejo y almacenamiento de múltiples mapas*).

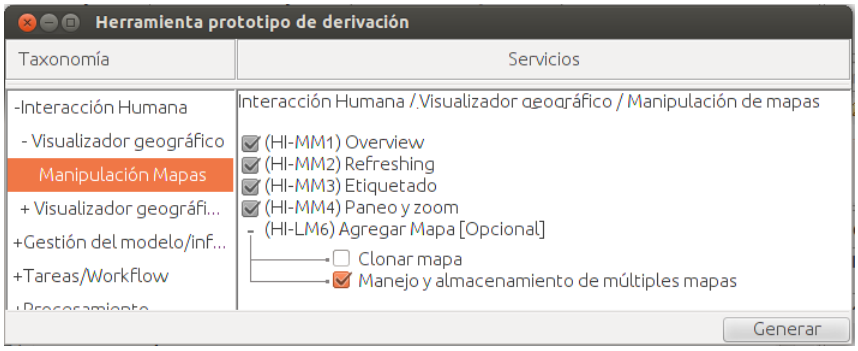

Fig. 6. Herramienta prototipo

El proceso de reestructuración permite agrupar los metadatos para representar la estructura jerárquica que presenta nuestra SPL. Para ello, como podemos observar en la Figura 7 se utiliza un patrón *Composite* componente Director que engloba al resto de los componentes. Cada componente puede estar compuesto por un conjunto de puntos variantes. Los puntos variantes deben estar asociados a un conjunto de Deltas (Internos o Externos). Para los Deltas Externos se deben asociar los componentes que implementan las variantes correspondientes. Esta estructura interna de la herramienta prototipo permite asistir el proceso de derivación y generar una correcta visualización para la configuración por parte del usuario.

El modelo de metadatos con anotaciones que hemos definido está configurado para permanecer sólo en el código antes del proceso de derivación. De esta manera el código del producto derivado será más legible y limpio, y permitirá añadir las funcionalidades particulares del mismo sin generar mayores dificultades. Así, el modelo de metadatos queda totalmente oculto. Por ejemplo, en la

[10] Denominado parsing en inglés, el cual denota el proceso de analizar una secuencia de símbolos a fin de determinar su estructura gramatical

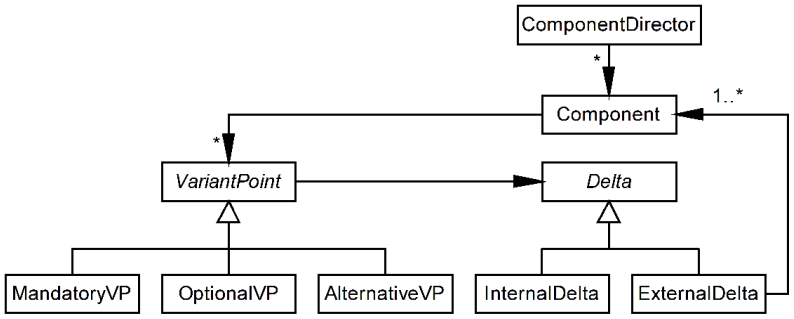

Fig. 7. Estructura interna de la herramienta prototipo

Figura 8 puede observarse el códico resultante para el punto variante de la Figura 5, tras seleccionar la opción correspondiente al "Delta1". Como vemos, la llamada al método *puntoVatiante()* fue remplazada por el código correspondiente al *callerCode* del Delta1 (Figura 5) y las anotaciones ya no están presentes.

```
public void foo(){
//Previous Code
    agregarMapaDelta1(...);
//Following Code
}
public void agregarMapaDelta1(.ParameterList_Delta1){...}
```

Fig. 8. Códido resultante tras proceso de derivación

5 Conclusiones y Trabajos Futuros

En este trabajo, hemos definido un modelo de metadatos que permite modelar y gestionar la variabilidad en una línea de productos de software. El mismo ha sido desarrollado mediante una combinación de dos enfoques altamente referenciados en la literatura: el modelo jerarquíco [11] y el modelado Delta [22]. Dichos enfoques fueron aplicados de manera de maximizar los beneficios de cada uno. Así, el modelo fue definido mediante un sistema de anotaciones que permite, por un lado contar con una estructura jerárquica para la creación y adaptación de nuevos componentes, además de facilitar la modificabilidad de los mismos; y por el otro realizar pequeños ajustes en el código posibilitanto variabilidades y configuraciones pequeñas. De esta forma, el modelo intenta ser lo suficientemente flexible permitiendo tanto varia-bilidades de granularidad fina como gruesa. A su vez, hemos desarrollado una herramienta prototipo que utiliza el modelo de metadatos para soportar el proceso de derivación de nuevos productos en una

SPL. Todo este trabajo ha sido ilustrado mediante la aplicación del mismo en un caso de estudio real dentro del subdominio de ecología marina. Dicha SPL previa poseía ciertas restricciones, en cuanto a tecnología subyacente y desiciones de diseño, que debieron contemplarse a la hora de definir nuestro modelo.

Como trabajo futuro es necesario realizar las tareas de validación que permitan evaluar nuestra propuesta mediante dos parámetros principales: medir costo y esfuerzo de reimplementar el modelo de metadatos sobre componentes previamente desarrollados, y medir los beneficios obtenidos a la hora de derivar nuevos productos utilizando ahora la herramienta prototipo que asiste dicho proceso. Sin embargo, podemos afirmar que dichos tiempos van a ser mejorados ya que en la SPL anterior la variabilidad debía ser gestionada en forma manual. Igualmente, debemos evaluar también el costo de la reestructuración realizada por la herramienta de acuerdo a la complejidad de posibles estructuras jerárquicas presentes en la SPL.

Por otro lado, una de las limitaciones que presenta esta propuesta es que está fuertemente ligada al lenguaje de programación Java, ya que fue desarrollada utilizando herramientas propias del mismo. Por lo tanto, se está trabajando en extender este modelo de metadatos para que sea independiente del lenguaje de programación utilizado.

References

1. G. T. Heineman and W. T. Councill, editors: Component-based software engineering: putting the pieces together. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.2001
2. Clemens A. Szyperski.:Component software - beyond object-oriented programming. Addison- Wesley-Longman. 1998
3. Frank van der Linden, Klaus Schmid, and Eelco Rommes: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
4. Klaus Pohl, Gnter Bckle, and Frank J. van der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
5. F. Bachmann and P. Clements: Variability in Software Product Lines. Software Engineering Institute,Technical Report CMU/SEI-2005-TR-012, 2005
6. Matthias Galster and Danny Weyns and Dan Tofan and Bartosz Michalik and Paris Avgeriou:Variability in Software Systems - A Systematic Literature Review. IEEE Transactions on Software Engineering, 2013.
7. Chen, Lianping and Ali Babar, Muhammad and Ali, Nour: Variability Management in Software Product Lines: A Systematic Review. 13th International Software Product Line Conference, SPLC '09, 2009.
8. Michael P. Papazoglou and Paolo Traverso and Schahram Dustdar and Frank Leymann:Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer, volume 40, page 38-45, 2007
9. Chang-ai Sun and Rowan Rossing and Marco Sinnema and Pavel Bulanov and Marco Aiello:Modeling and managing the variability of Web service-based systems. Journal of Systems and Software, volume 8, page 502-516, 2010

10. Czarnecki, Krzysztof and Grünbacher, Paul and Rabiser, Rick and Schmid, Klaus and Wasowski, Andrzej:Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. Sixth International Workshop on Variability Modeling of Software-Intensive Systems, 2012.
11. Haber, Arne and Rendel, Holger and Rumpe, Bernhard and Schaefer, Ina and van der Linden, Frank:Hierarchical Variability Modeling for Software Architectures. 15th International Software Product Line Conference, 2011.
12. Christian Kastner and Salvador Trujillo and Sven Apel:Visualizing Software Product Line Variabilities in Source Code. In Proc. SPLC Workshop on Visualization in Software Product Line Engineering (ViSPLE), 2008.
13. Schaefer, Ina and Bettini, Lorenzo and Damiani, Ferruccio and Tanzarella, Nico: Delta-oriented Programming of Software Product Lines. 14th International Conference on Software Product Lines: Going Beyond, 2010
14. P. Pernich, A. Buccella, A. Cechich, M. S. Doldan, E. Morsan, M. Arias, and M. Pol'la:Developing a Subdomain-Oriented Software Product Line. CACIC 2011: XVII Congreso Argentino en Ciencias de la Computacin, 2011
15. P. Pernich, A. Buccella, A. Cechich, M. Pol'la, M. Arias, M. S. Doldan and E. Morsan: Product-Line Instantiation Guided By Subdomain Characterization: A Case Study. Journal of Computer Science and Technology 2012, Special Issue 12(3). ISSN: 1666-6038. Disponible en http://journal.info.unlp.edu.ar/journal/journal34/this.html (116-122), 2012
16. A. Buccella and A. Cechich and M. Pol'la and S. Doldan and E. Morsan:Towards Systematic Software Reuse of GIS: Insights from a Case Study. Computers & Geosciences. 54. Elsevier Science Publishers B. V. ISSN: 0098-3004, DOI: 10.1016/j.cageo.2012.11.014. (9-20), 2013.
17. M. Pol'la, A. Buccella:Instanciación y Validación de una Línea de Productos de Software aplicada al subdominio de la Ecología Marina. Tesis de Licenciado en Ciencias de la Computación, Facultad de Informática, U.N.Co., 2013.
18. Cheng, Betty H. et all :Software Engineering for Self-Adaptive Systems: A Research Roadmap. Software Engineering for Self-Adaptive Systems,2009
19. K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson: Feature-oriented domain analysis (FODA) feasibility study. Technical report, CMU/SEI-90TR-21, 1990.
20. Software Productivity Consortium Services Corporation,Technical Report SPC-92019-CMC. Reuse-Driven Software Processes Guidebook, Version 02.00.03, 1993
21. Lienhardt, Michael and Clarke, Dave: Row types for delta-oriented programming. VaMoS - Sixth International Workshop on Variability Modeling of Software-Intensive Systems - 2012.
22. Helvensteijn, Michiel and Muschevici, Radu and Wong, Peter Y. H.: Delta Modeling in Practice: A Fredhopper Case Study. Sixth International Workshop on Variability Modeling of Software-Intensive Systems, 2012.
23. A. Buccella and A. Cechich and M. Polla and M. Doldan and E. Morsan: A Taxonomy to Facilitate Systematic Reuse of Marine Ecology Services. GeoInformatica - International Journal on Advances of Computer Science for Geographic Information Systems. 2013
24. N. Huenchuman, A. Buccella , A. Cechich, M. Polla, M.S. Doldan, E. Morsan and M. Arias: Reingeniería de una Línea de Productos de Software: Un Caso de Estudio en el Subdominio de Ecología Marina CACIC 2013: XVII Congreso Argentino en Ciencias de la Computacin, 2013
25. M. Arias, A. Cechich: Capacidades de Composición Dinámica de Componentes para una Línea de Productos de Software. Tesis de Licenciado en Ciencias de la Computación, Facultad de Informática, U.N.Co., 2014.