Un Sistema de Anotaciones para la Especificación de Componentes en una Línea de Productos de Software

Matias Pol'la[1], Maximiliano Arias[2], Agustina Buccella[3], Alejandra Cechich[4]

[1,2,3,4]GIISCO Research Group- Departamento de Ingeniería de Sistemas - Facultad de Informática
Universidad Nacional del Comahue, Neuquén, Argentina

[1]matias.polla@fi.uncoma.edu.ar, [2]maximiliano.arias@fi.uncoma.edu.ar,
[3]agustina.buccella@fi.uncoma.edu.ar, [4]alejandra.cechich@fi.uncoma.edu.ar

[1,2,3,4]Consejo Nacional de Investigaciones Cientpificas y Técnicas - CONICET

Resumen: *Las Líneas de Productos de Software (LPS) promueven el reuso y la mejorara de tiempos, costos y calidad de los productos obtenidos. Además, utilizar un enfoque basado en componentes para el desarrollo de los servicios requeridos por las mismas, proporciona mayor reuso brindando flexibilidad para la creación de nuevas SPL o la instanciación de las existentes. Sin embargo, tanto el diseño como la implementación de estos componentes posee una serie de desafíos que se deben considerar con especial interés para poder garantizar su reuso efectivo. En este trabajo, proponemos un sistema de anotaciones el cual se define mediante un modelo de metadatos que permite especificar la información necesaria sobre la semántica de los servicios implementados por cada componente, sus requerimientos técnicos, y sus variabilidades. Este sistema se aplica luego a componentes reales de una LPS previamente desarrollada para el dominio de ecología marina.*

Palabras Claves: *Líneas de Productos de Software, Componentes, Metadatos .*

Abstract: *Software Product Lines (SPL) development promotes reuse, improves quality and reduces development timesands costs. Using a component-based development approach to implement SPL services provides further reuse, improving flexibility for creating new SPLs and instantiating products from an existing one. However, the design and implementation of these components has a number of challenges that must be taken into consideration in order to ensure its effective reuse. In this paper, we propose an annotation system defined by a metadata model. Such system allows to specify the necessary information about the semantics of the services implemented by each component, technical requirements, and variability. This system is then applied to real software components from a previously developed SPL for the marine ecology domain.*

Keywords: *Simulator, virtual machines, servers, networks, computer security, Systems Laboratory.*

INTRODUCCIÓN

El re uso de software es uno de los principales objetivos a alcanzar en todo desarrollo de software. No solo porque provee beneficios en cuanto a costos y tiempos, sino también porque facilita en gran medida el trabajo de etapas del desarrollo que consumen mucho trabajo y esfuerzo y son muchas veces relegadas en pos de otras que generan mayor visibilidad a los usuarios. El uso de artefactos, ya sean módulos, componentes, o subsistemas, previamente verificados y aplicados que sean luego fácilmente adaptados o integrados a nuevos sistemas en construcción, es uno de los temas de investigación que mas auge ha tenido en los últimos años dentro de la ingeniería de software. Sin embargo, diseñar

para el reuso es una tarea compleja ya que se deben considerar muchos aspectos en forma integral para garantizar justamente que los artefactos generados sean luego fácilmente reusados.

Dentro de los paradigmas mas utilizados en la actualidad que apuntan al desarrollo de artefactos de software reusables son: el Desarrollo de Software Basado en Componentes (BSBC) [1] y las Líneas de Productos de Software (LPS) [2, 3, 4]. El primero, provee un conjunto de técnicas y metodologías para desarrollar nuevos sistemas a partir de la combinación de componentes previamente creados y cuya funcionalidad ya fue comprobada. Busca producir software con el propósito de reusarlo en desarrollos futuros y en contextos distintos a aquellos para los que fue diseñado. El otro paradigma, LPS, se centra en dominios específicos con similares funcionalidades que serán luego adaptadas a las necesidades específicas de los nuevos productos derivados a partir de ellas. El aspecto fundamental de toda LPS es llegar a la construcción de una arquitectura que permita incorporar componentes reusables, de modo que al desarrollar nuevos productos en lugar de hacerlo desde el principio, se instancien los artefactos necesarios. A su vez, a pesar de que las LPSs y el DSBC son dos paradigmas que han surgido con objetivos diferentes, pueden ser combinados para promover el reuso y facilitar el desarrollo de sistemas.

Dicha combinación, aunque nos debería proveer de los beneficios de ambos en cuanto al reuso, tiempo, costo y calidad, nos heredaran inevitablemente sus complejidades.

Por lo tanto, se debe poner especial énfasis en los aspectos mas críticos de ambos paradigmas de manera de garantizar el éxito de un proceso de desarrollo de LPSs por medio de componentes. En particular, en este trabajo, hemos considerado dos de esos aspectos: la gestión de la variabilidad para las LPSs [5, 6, 7] y las especificaciones e información que pueda ser extraída desde los componentes [1, 8] para que puedan ser integrados y adaptados mas fácilmente.

Con esto en mente, hemos creado un sistema de anotaciones aplicado a componentes JAVA que define un modelo de metadatos almacenando la información necesaria para representar variabilidades y especificaciones técnicas de los mismos. Dichas anotaciones contemplan ademas artefactos de diseño que han sido creados en trabajos previos [9], otorgando a la LPS la trazabilidad necesaria para conocer la manera en que los requerimientos de los usuarios fueron diseñados e implementados. Tanto la definición del sistema de anotaciones, como su aplicación y uso son descriptos en este trabajo y aplicados a una LPS desarrollada en trabajos previos [9, 10, 11, 12]. Ademas, presentamos una herramienta prototipo que permite recuperar dichas anotaciones sistemáticamente para darles el uso necesario, tanto dentro del proceso de configuración y derivación de productos, como dentro del proceso de integración con otros componentes.

Este artículo esta organizado de la siguiente manera. En la sección siguiente se presentan los trabajos relacionados y antecedentes. Luego, se detalla el sistema de anotaciones propuesto junto con su implementación en componentes JAVA. Luego, como caso de estudio, se describe la aplicación de dicho sistema a componentes de la LPS previamente desarrollada y se describe la herramienta prototipo que hace uso de los mismos. Por último se enumeran las conclusiones obtenidas y los trabajos futuros.

TRABAJOS RELACIONADOS Y ANTECEDENTES

En este trabajo debemos considerar dos líneas de investigación específicas: la gestión de la variabilidad y la integrabilidad de componentes. En cuanto a la primera, existen gran cantidad de trabajos y revisiones sistemáticas [6, 13] que proponen y analizan diferentes enfoques recientes.

En particular, en [6] 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 distintos 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 LPS, 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 Características) [14], el cual intenta capturar las características principales de un conjunto de sistemas similares pertenecientes a un dominio específico, diferenciando entre las características que son compartidas por todos los sistemas y las variabilidades. Otro trabajo, surgido a partir de FODA, es el Modelado de Características (FM-Feature Modeling) [15] 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 características de niveles superiores en características detalladas y especificas. 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 [16] 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 enfoques proponen la implementación de la variabilidad directamente sobre el código de un sistema, mediante la Programación Orientada Delta (POD) [17, 18]. En estos enfoques una LPS se representa por medio de un modulo 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 necesidades particulares de cada producto. Cada delta genera cambios en el modulo principal (núcleo) a nivel de código tanto agregando, modificando o eliminando el mismo. Esta implementación para LPSs es mas flexible y menos restrictiva que el modelado de características (FM) y otros enfoques modulares, ya que permite realizar modificaciones simples en el código. Ademas 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 segundo lugar, debemos considerar aquellos trabajos relacionados con la especificación de componentes que ayuden a mejorar la integrabilidad de los mismos. En general existen una gran gama de trabajos en ésta línea, las cuales algunas de ellas se centran en las capacidades de composición de los componentes. El problema de composición dinámica ha sido abarcado desde distintas propuestas [19, 20, 21, 22] haciendo uso de archivos de configuración XML, programación orientada a aspectos, especificaciones FODA (Feature Oriented Domain Analysis) e incluso lenguajes formales como ADL. Por otro lado, el uso de modelos de metadatos se ha aplicado para asistir en un gran numero de tareas relacionadas con el proceso de desarrollo [23, 24, 25, 26, 27] incluyendo testeo y validación, inclusión de descripciones en lenguaje natural para facilitar búsquedas en catálogos, configuración de herramientas,entre otras tareas.

En nuestro trabajo, considerando los trabajos mencionados y de acuerdo a las particularidades de la LPS previamente desarrollada, se decidió

realizar un sistema de anotaciones basado en un modelo de metadatos para gestionar la variabilidad y mejorar la especificación de los componentes. El sistema de anotaciones provee un enfoque mixto que intenta maximizar los beneficios tanto de los enfoques jerárquicos así como los que producen modificaciones directamente sobre el código. De esta manera el sistema de anotaciones permite gestionar la variabilidad a nivel de granularidad fina (a nivel de código), permitiendo una mayor flexibilidad y reduciendo el costo de derivación.

A su vez, provee de anotaciones de metadatos que proveen una detallada especificación de los aspectos técnicos y semánticos de los componentes aportando al entendimiento y trazabilidad de la línea.

ANTECEDENTES

En trabajos previos [9, 10, 11, 12] hemos desarrollado una LPS, en el dominio geográfico, que sirvió como una plataforma de servicios comunes, aplicados a todos los productos de la LPS, y servicios variables, aplicados solo a algunos productos. Dicha línea fue creada a partir de una metodología de desarrollo orientada a niveles de dominios [12, 28, 29] la cual combina ventajas de otras metodologías muy referenciadas tanto en el ámbito académico como en la industria [2, 3, 4]. Para la construcción de la línea se trabajó conjuntamente con dos organizaciones, el Instituto de Biología Marina y Pesquera "Almirante Storni"[1] y con el Centro Nacional Patagónico (CENPAT-CONICET)[2] las cuales brindaron el soporte necesario para la especificación y validación de las funcionalidades de los dominios analizados.

A su vez, se han realizado esfuerzos para estandarizar los servicios requeridos dentro de los mismos de manera de mejorar la comunicación entre las partes y partir de diseños comunes y consensuados. Así, entre las partes interesadas (biólogos e informáticos) se definió una taxonomía de servicios y guías de uso de los mismos; ambas basadas en los estándares de servicios dentro del dominio geográfico. En particular se extendió la taxonomía definida en la ISO/DIS 191193 enfocándonos en los servicios del subdominio de ecología marina. Dichos servicios fueron utilizados para la migración del sistema a un enfoque de componentes de software [29] para permitir la implementación, utilización y ensamblaje de los mismos en forma independiente. Estos trabajos nos permitieron desarrollar la línea de manera incremental agrupando las funcionalidades en componentes de acuerdo a los servicios que implementan. La LPS cuenta con un arquitectura de tres capas, respetando los estandartes utilizados para los sistemas de información geográfica. En la Figura 1 se pueden observar dichas capas, las cuales involucran, la Interfaz de Usuario, Procesamiento Geográfico y Modelo Geográfico; y donde cada una de ellas esta conformada por un grupo de componentes que responden a diferentes servicios. De esta manera, cada uno de los componentes incluidos en la LPS se corresponde con un conjunto de servicios detallados en la taxonomía de servicios definida. Por ejemplo, en la figura podemos observar la funcionalidad "Visualizar estadísticas geográficas" en donde vemos que el componente de Estadísticas Geográficas, perteneciente a la capa de Procesamiento Geográfico, implementa los servicios de Determinar Características en diferentes zonas (PS-S 2.1) y Determinar Características en diferentes estaciones (PS-S 2.2); mientras que el componente Estadísticas Gráficas, perteneciente a la capa de Interfaz de usuario, implementa los servicios Visualizar datos por Histogramas y Visualizar datos por tabla (HI-GS1.1 y HI-GS1.1 respectivamente). Ademas, este componente presenta un punto variante que permite configurar el tipo de

[1] http://www.ibmpas.org/
[2] http://www.cenpat.edu.ar/
[3] ISO/DIS 19119 : Geographic information Services, ISO/TS 2005

visualización deseada de los datos obtenidos.

El trabajo conjunto con el IBMPAS y el CENPAT-CONICET genero la derivación de dos productos de la LPS. Los servicios presentes en ambos productos generados son la visualización de las campañas, estaciones y zonas así como la visualización de diferentes estadísticas en diversos formatos. Ademas de contar con las funcionalidades comunes, el primer producto, implementado para el IBMPAS, cuenta con servicios de creación de histogramas para observar diferentes estadísticas de los fenómenos de estudio, junto con un modulo de detección de patrones de distribución de especies presentado en [11]. Un prototipo del primer producto puede visualizarse en http://gissrv.fi.uncoma.edu.ar/SaoProjectUI/. Por otro lado, el segundo producto posibilita la creación de diferentes mapas para visualizar dos o mas zonas en paralelo. También presenta un componente de procesamiento encargado de analizar el avance (crecimiento y distribución) de determinadas especies a lo largo de un determinado periodo.

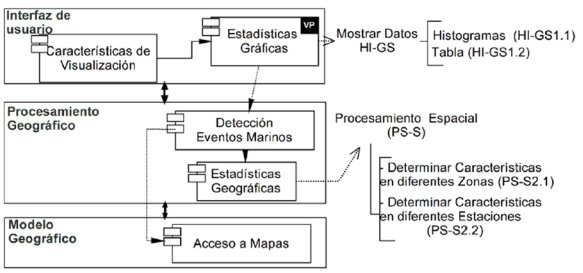

Figure 1: Arquitectura de la LPS junto diseño de componentes y servicios para la funcionalidad visualizar estadísticas gegráficas.

La implementación actual de la LPS fue realizada mediante una tecnología puramente basada en componentes utilizando Enterprise Java Beans 4 (EJB) junto con la herramienta Google Web ToolKit 5 (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 [9]. 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 código JavaScript y HTML equivalente.

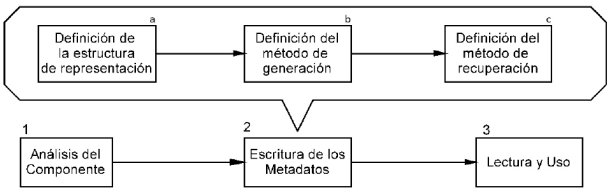

Figure 2: Proceso de definición del modelo de metadatos para el sistema de anotaciones de una LPS.

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.

A pesar de que, todo el trabajo realizado ha generado buenos resultados a la hora de la derivación de los nuevos productos.

UN SISTEMA DE ANOTACIONES PARA COMPONENTES DE UNA LPS

Para mejorar algunos de los aspectos mencionados previamente se cero un sistema de anotaciones mediante la definición de un modelo de metadatos que nos permite representar la información estandarizada para los dominios involucrados incluyendo aspectos que ayuden a la gestión de la variabilidad e integrabilidad de los componentes

[4]Oracle, http://www.oracle.com/technetwork/java/javaee/ejb/index.html
[5]http://code.google.com/intl/es-ES/webtoolkit/

implementados. Así, en esta sección describimos el proceso de definición de metadatos que hemos creado a nivel general y luego su aplicación a componentes Java dentro de la LPS.

PROCESO DE DEFINICIÓN DEL MODELO DE METADATOS

En la Figura 2 podemos observar las tres etapas principales para crear los metadatos necesarios que conformaran luego nuestro sistema de anotaciones dentro de los componentes.

La figura muestra, a su vez, el detalle de las actividades de la segunda etapa las cuales son necesarios para la especificación de otro proceso que permite definir, generar, y recuperar los metadatos definidos en los componentes.

CADA UNA DE LAS ETAPAS SE DESCRIBEN BREVEMENTE A CONTINUACIÓN:

1. Análisis del componente: Previo a definir los metadatos, los componentes deben ser analizados de acuerdo al conjunto de interfaces que le permiten describir sus funcionalidades y sus requerimientos de integración. De esta forma podemos identificar dos elementos: la interfaz funcional del componente [1] y los requerimientos (o interfaz) arquitecturales [30]. La primera describe el comportamiento del componente, incluyendo todos los atributos públicos, métodos, eventos emitidos y excepciones generadas. Por otro lado, los requerimientos arquitecturales definen las necesidades del componente para ser integrado al resto. Dentro de estos requerimientos podemos encontrar el lenguaje en el que se implementa el componente, puntos de entrada, dependencias externas (librerías, otros componentes, bases de datos, etc.), Dependencias internas (en el caso que un componente este compuesto por otros), etc. De esta manera el proceso de análisis debe identificar todos estos elementos para poder ser utilizados en el siguiente paso.

2. Escritura de los metadatos: Como el proceso de escritura de los metadatos requiere primeramente de una correcta definición del modelo a utilizar, hemos definido un proceso general para tal tarea. La Figura 2 muestra cómo se organiza dicho proceso, identificando tres actividades:

(a) Definición de la estructura de representación: Durante esta actividad se realiza el análisis del problema que se desea resolver y a través de esto se obtiene el conjunto de metadatos que resulten pertinentes. Una vez definido dicho conjunto, se establece una estructura para representar los mismos. Existen distintos métodos de representación según las necesidades del problema que resolverán los metadatos. Por un lado, los metadatos pueden almacenarse en archivos externos mediante el uso texto plano, archivos XML, entre otros. Por otro lado, puede suceder que se almacenen de manera interna al código haciendo uso de comentarios especialmente marcados, anotaciones, o alguna otra tecnología ofrecida por la plataforma sobre la que se desarrolla.

(b) Definición del método de generación: Una vez definida la estructura de representación se establece la forma en que los datos serán generados. El proceso puede realizarse de manera manual, automática, o asistida. Esta tarea esta muy relacionada a la plataforma sobre la cual se trabaja. Durante la misma se utiliza la estructura de representación definida en el paso anterior y se realiza un análisis tecnológico para seleccionar o construir el método que se usaría

para generar o escribir los metadatos.

(c) Definición del método de recuperación: Por ultimo, de acuerdo a la estructura de representación y al método de generación establecido, se define un método de recuperación y uso de los metadatos generados. Este método se encuentra muy relacionado a los elementos anteriormente definidos y puede resultar necesaria la construcción de herramientas para tal fin. Al igual que en la actividad anterior, es necesario realizar un análisis tecnológico de la plataforma elegida y realizar la selección o creación del método de recuperación. Durante esta actividad puede surgir ademas la necesidad de una herramienta que sea capaz de leer y mostrar los metadatos escritos.

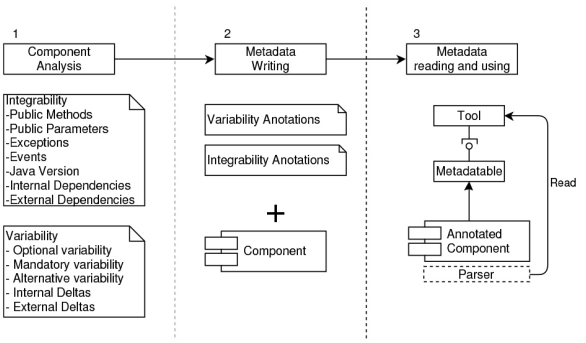

Figure 3: Aplicación de los metadatos para crear el sistema de anotaciones a componentes de una LPS.

3. Lectura y uso: La etapa final del proceso es la lectura de los metadatos escritos y el uso de los mismos para el fin que hayan sido creados. De esta manera el proceso de lectura se adapta a lo establecido durante el proceso de definición del modelo y el uso estará definido por la tarea que se quiera cumplir. Esta tarea esta íntimamente relacionada con el problema a resolver, como por ejemplo, la automatización o asistencia en la ejecución de un proceso, configuración de herramientas, trazabilidad, entre otras.

IMPLEMENTACIÓN DEL SISTEMA DE ANOTACIONES UTILIZANDO JAVA PARA COMPONENTES DE UNA LPS

El proceso definido en la sección anterior (Figura 2) puede ser aplicado directamente a componentes JAVA considerando a su vez particularidades de las LPSs. Así, en cada una de las etapas se obtienen una serie de diferentes artefactos que ayudaran a la construcción y uso de los metadatos y definirán el sistema de anotaciones propuesto.

En la primera etapa, análisis del componente, debido a que JAVA no define por defecto una estructura para la creación de componentes, toda información de interfaz, excepciones y eventos generados esta relacionada a los métodos públicos de las clases del componente. Si bien JAVA permite el uso del framework provisto por Enterprise Java Beans para desarrollo de componentes, la única diferencia con lo antes mencionado es que los métodos públicos se encuentran centralizados en un único archivo de interfaz. Las dependencias arquitecturales de un proyecto JAVA pueden encontrarse en sus archivos de configuración (properties, classpath, etc). Estos archivos suelen generarse automaticamente por los entornos de desarrollo utilizados como Eclipse6 o Net-Beans7 y de ellos se puede conseguir una representación escrita de los elementos antes mencionados.

En la segunda etapa, escritura de los metadatos, se deben definir los metadatos necesarios para la escritura de los elementos identificados en la etapa anterior. Así, las tres actividades involucradas (Figura 2) se aplicaron al desarrollo de metadatos para componentes JAVA dentro de las LPSs y se detallan a continuación:

1. Definir la estructura de representación de los metadatos: Como el sistema de anotaciones persigue dos objetivos principales, lograr la

representación y gestión de la variabilidad y promover la integrabilidad de los componentes, se definieron metadatos específicos para cada uno. Para el caso de la representación de la variabilidad hemos definido tres metadatos principales: los deltas, los puntos variantes y los tipos de variabilidad requeridos (Figura 4). Los primeros son las modificaciones que sufre el código para posibilitar la circunstancian correcta. Estos deben corresponderse con variantes externas, que son aquellas que son implementadas en componentes externos o componentes, 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 ultimo, 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 mas de una de las opciones posibles.
- Variabilidad Alternativa: solo es posible instanciar a lo sumo una de las alternativas presentadas.

Para el caso de los 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. Ademas, 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. Este determina las restricciones al momento de realizar la configuración de cada punto variante. Para los metadatos de Tipos de Variabilidad (Opcional, Alternativa Obligatoria), se almacenan el código de servicio determinado por la taxonomía de servicios [9], junto con una descripción del mismo. Ademas 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.

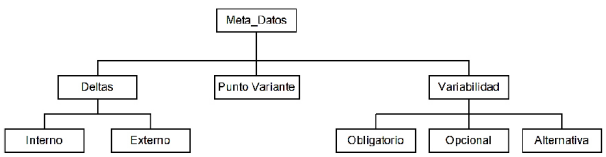

Figure 4: Metadatos para la representación de la variabilidad.

Por otro lado, los metadatos propuestos para la integrabilidad se asocian principalmente con los aspectos de interfaz funcional y arquitectural de los componentes. Con respecto a la interfaz funcional, es fundamental conocer la información publica del componente, es decir aquellos elementos que utiliza para comunicarse con otros componentes y para ser configurado. De esta manera, es necesario conocer los elementos que describen el comportamiento de un componente, incluyendo los métodos públicos, las variables de configuración o parámetros públicos, los eventos registrados y las excepciones que puede lanzar. La

[6]https://www.eclipse.org/
[7]https://netbeans.org/

interfaz arquitectural se refiere a todos aquellos requisitos del componente necesarios para poder ser acoplado a otros. Esta categoría incluye la versión de JAVA mínima para una correcta ejecución, las dependencias internas (otros componentes de la línea con los que se relaciona) y las dependencias externas (librerías, componentes de terceros, bases de datos, etc.).

Para estructurar todos estos metadatos, de variabilidad e integrabilidad, se decidió utilizar un esquema de dos elementos: una etiqueta y los datos relacionados a la misma. A su vez, esta etiqueta está formada por dos componentes: una categoría y una categorismo. La etiqueta permite identificar los metadatos y clasificarlos según el ámbito al que se refieren. La Tabla 1 muestra el modelo final de metadatos propuesto para aportar a los dos objetivos. Por ejemplo, podemos ver que el metadato Funcional Interface/Method se encarga de mantener toda la información relacionada con la signatura de los métodos públicos definidos en un componente. Con respecto a la variabilidad, podemos observar el metadato variability opcional utilizado para demarcar un punto variante opcional, permitiendo definirlos deltas y las instancias asociadas al mismo.

2. Definir el método de generación: Debido a que estamos considerando componentes en la plataforma JAVA, hemos elegido método de generación acorde a la misma, haciendo uso de las librerías que dan soporte a JAVA Annotations8. Éstas permiten insertar diferentes directivas a nivel de código y están estructuradas utilizando el carácter especial "@" junto con un nombre o etiqueta que las 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 "method" ya que para nuestra estructura necesitamos anotar métodos. Por ejemplo, en el Código 1 se puede observar la implementación para la definición de la anotación Meta Variabilty Optional correspondiente al metadato Variability Optional (Tabla 1). En la misma se distinguen la política de retención utilizada y el objetivo junto con los parámetros correspondientes. Este estilo de implementación se utilizó indistintamente, tanto para variabilidad como para integrabilidad. Cada una de las anotaciones definidas varia en el objetivo anotado, su nombre y sus parámetros. De esta manera, todas las anotaciones implementadas comparten la misma política de retención.

ETIQUETA	SUB-ETIQUETA	DATOS CONTENIDOS
INTEGRABILITY	FUNCTIONALIFC/METHOD	MÉTODOS PÚBLICOS DEL COMPONENTE
INTEGRABILITY	FUNCTIONALIFC/PUBLICPARAMETER	PARÁMETROS PÚBLICOS DEL COMPONENTE
INTEGRABILITY	FUNCTIONALIFC/EVENTS	EVENTOS GENERADOS POR EL COMPONENTE
INTEGRABILITY	FUNCTIONALIFC/EXCEPTIONS	EXCEPCIONES QUE PUEDEN GENERARSE
INTEGRABILITY	ARCHITECTURALIFC/JAVAVERSION	VERSIÓN DE JAVA MÍNIMA NECESARIA
INTEGRABILITY	ARCHITECTURALIFC/INTERALDEPENDENCIES	SUBCOMPONENTES UTILIZADOS
INTEGRABILITY	ARCHITECTURALIFC/EXTERNALDEPENDENCIES	LIBRERÍAS, COMPONENTES, BASES DE DATOS, ETC.
VARIABILITY	OPTIONAL	ATRIBUTOS PARA DEFINIR VARIABILIDAD OPCIONAL
VARIABILITY	MANDATORY	ATRIBUTOS PARA DEFINIR VARIABILIDAD OBLIGATORIA
VARIABILITY	ALTERNATIVE	ATRIBUTOS PARA DEFINIR VARIABILIDAD ALTERNATIVA
VARIABILITY	VARIANT	ATRIBUTOS PARA DEFINIR VARIABILIDAD VARIANTE
VARIABILITY	DELTA EXTERNAL	PARÁMETROS, LIBRERÍAS Y COMPONENTES EXTERNOS
VARIABILITY	DELTA INTERNAL	DEFINICIÓN DE PARÁMETROS Y LIBRERÍAS

Table 1: Modelo de metadatos defindo para componentes de una LPS.

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

```
import java.lang.annotation.*;


@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Meta_VariabilityOptional
{
   public String serviceCode();
   public String serviceDescription();
   public String []instance();
   public String []deltaId();
}
```

Código 1: Implementación de una anotación java personalizada.

En la Figura 5 puede observarse el sistema de anotaciones definido de acuerdo al modelo de metadatos analizado previamente (Tabla 1). Esta estructura esta, al igual que el modelo de metadatos, también dividida en dos grupos de anotaciones principales, aquellas relacionadas con la variabilidad y aquellas relacionadas con la integrabilidad. A su vez cada uno de ellos se encuentra subdividido en dos categorías. En primer lugar, las anotaciones relacionadas con la variabilidad se dividen en Meta VariantPoint y los Meta Deltas que se corresponden con los elementos Punto Variante y Deltas de la Figura 4. La primera categoría esta formada por las anotaciones @Meta Variability Optional, @ Meta VariabilityMandatory y @Meta Variability-Alternativa. 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 1. La segunda categoría, involucra a las anotaciones @Meta DeltaInternal y @ Meta DeltaExternal que son las encargadas de representar las diferentes modificaciones que sufre una LPS 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 anotaciones 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), ademas de una descripción específica del Delta (Instance Description) que indica las particularidades del mismo. El segundo grupo de anotaciones, que engloba todas aquellas relacionadas con la integrabilidad, se divide nuevamente en las dos categorías, la interfaz funcional y la interfaz arquitectural. La primera de las categorías incluye las anotaciones @Meta Method, @Meta PublicParameter, @Meta Events y @ Meta Exception. Al igual que las anteriores cada una de ellas posee un conjunto de parámetros que las describe. La anotación @Meta Method se escribe sobre todos los métodos públicos del componente y almacena su nombre, su lista de parámetros y su tipo de retorno. @Meta PublicParameter anota aquellos parámetros públicos del componente que funcionan como variables de configuración, y posee el nombre y el tipo de los mismos. La anotación @Meta Events almacena aquellos eventos que pueden ser generados por el componente y almacena su nombre, sus pará-

metros y el tipo del manejador necesario. Por ultimo, @Meta Exception almacena información referida a todos los tipos de excepciones que pueden dispararse durante la ejecución del componente. La categoría de interfaz arquitectural incluye las anotaciones @Meta JavaVersion, @Meta External Depencies y @ Meta Internal Dependencies. La primera de ellas, almacena la versión mínima de JAVA necesaria para el correcto funcionamiento del componente. La segunda almacena todas las dependencias externas del componente (librerías, bases de datos, componentes de terceros, etc) necesarias para que el componente pueda realizar sus tareas. Finalmente, Meta Internal Dependencies se encarga de almacenar toda la información relacionada con los componentes de la línea utilizados por el componente para llevar a cabo sus tareas.

Como puede observarse los puntos variantes no son representados por medio de anotaciones ya que estos sólo indican el lugar puntual donde debe situarse la variabilidad. En la siguiente sección se explica como son representados estos puntos a nivel de código.

3. Definir el método de recuperación: Al tomar la decisión sobre la generación de los metadatos, se condiciona la definición del método de recuperación. Debido a que los metadatos se escriben de manera interna utilizando anotaciones y no se utilizan archivos externos para almacenarlos, se necesita diseñar un método de recuperación de los mismos. Para esto, y con el fin de facilitar la tarea de desarrollo de herramientas que puedan leer y usar estos metadatos se desarrolló un Parser de código JAVA capaz de extraerlos y volcarlos en objetos de un tipo predefinido “Metadato”. El Parser definido solo requiere que las clases que los utilizan implementen la interfaz “Metadatable”. Se creó ademas una herramienta que respeta estos requerimientos y permite visualizar en forma de tabla los metadatos contenidos en un archivo JAVA.

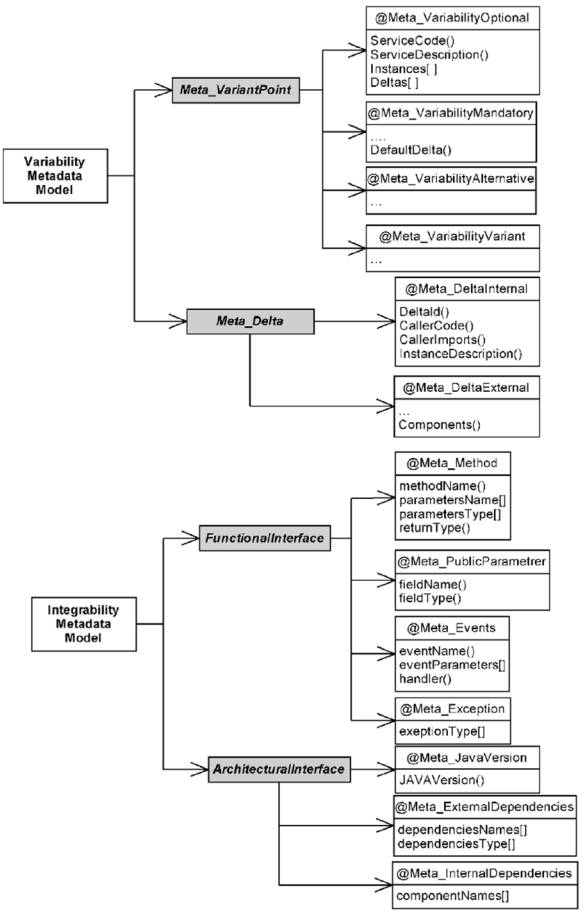

Figure 5: Sistema de anotaciones para los metadatos definidos en la Tabla 1.

Para la tercera etapa, lectura y uso, se desarrolló una herramienta de lectura en JAVA capaz de interpretar la interfaz “Metadatable”. Su objetivo es realizar la lectura de los metadatos escritos en el componente haciendo uso del parser definido. Esta herramienta no realiza procesamiento alguno sobre los elementos leídos, definiendo una “plantilla” para el desarrollo de otros sistemas que sí deseen realizarlo. Todos los elementos descriptos anteriormente (anotaciones, interfaz y parser) se empaque-

taron en una librería JAR, para ser importados por aquellos proyectos que quieran utilizarlos.

Estos dos últimos pasos, se describirán en detalle en la sección siguiente aplicados a componentes de la LPS previamente desarrollada.

CASO DE ESTUDIO

Una vez definido el sistema de anotaciones, se aplico el mismo a la implementación de algunos servicios de la LPS desarrollada en trabajos previos. Para ello fue necesario modificar el conjunto de algunos de estos componentes que conforman la plataforma de la LPS, incorporando los metadatos de variabilidad e integrabilidad según corresponda. Luego se desarrolló una herramienta prototipo capaz de leer y mostrar los diferentes metadatos presentes en los componentes y asistir al proceso de derivación de nuevos productos. Esta herramienta facilita principalmente la ejecución de dos tareas. Por un lado permite utilizar los datos recolectados para mejorar la especificación de los componentes, sin necesidad de aplicar técnicas de ingeniería inversa; y por el otro lado, asiste al proceso de derivación.

En el ejemplo presentado en el Código 2 se muestra el código del componente Estadísticas Gráficas (Figura 1) que incluye el modelo de metadatos definido. En el mismo se pueden observar las anotaciones asociadas a la integrabilidad como @ Publicparameter y @Meta_Method. Ademas se puede ver el método dummy variant Point MostrarDatos que indica el Punto Variante para la variabilidad ”Mostrar Datos”, junto con una de las posibles variantes (”Visualización en forma de histogramas”) representada por Delta1.

En la Figura 6 podemos observar la visualización del resultado de la herramienta prototipo del sistema de anotaciones, aplicado al componente Estadísticas Gráficas. En dicha figura se pueden ver los datos de InternalDependencies, Exception, PublicParameter, JavaVersion y ExternalDependecies pertenecientes a los metadados de integrabilidad. Ademas podemos observar el Delta Internal perteneciente a los metadatos de variabilidad, el cual esta asociado al “Delta1” correspondiente al servicio de “Detección de Eventos Marinos”.

```
@Meta_JavaVersion("1.6")
@Meta_InternalDependencies(
componentNames = {"DeteccionDeEventosMarinos"})
@Meta_ExternalDependencies(
  names = {"Sencha","GWT-1.5"},
  types = {"Library","Framework"})
@Meta_Exception(
  exceptionType={"DataNotFoundException")
public class EstadisticasGraficas implements
    Metadatable{
  @Meta_PublicParameter
    (name="MAX_GRAPHS",type="int")
  publi int MAX_GRAPHS;

  @Meta_Method(
    methodName="variantPointMostrarDatos",
    parameterNames={},
    parameterTypes={},
    returnType="void")
  @Meta_Variability_Optional (
    ServiceCode = "HI-GS",
    serviceDescription= "Visualizaci n de datos
        estad sticos.",
    instance={"inst1"},
    deltaId={"Delta1","Delta2"})
  public void variantPointMostrarDatos()
    throws DataNotFoundException
  {...}

  @Meta_Delta_Internal(
    DeltaId = "Delta1’"
    CallerImports = {"DeteccionDeEventosMarinos"}
    CallerCode = {"Delta1(Parameters);"}
    InstanceDescription = "Visualizaci\’on en
        forma de histogramas")
  private void Delta1()
  {...}
  //More Code..
}
```

Código 2: Ejemplo código JAVA con anotaciones de integrabilidad y variabilidad aplicadas.

Figure 6: Herramienta de lectura del sistema de anotaciones.

La herramienta desarrollada permite a su vez asistir al proceso de derivación de productos de una LPS. Esta herramienta recibe como entrada la información referente a la variabilidad recuperada de la lectura de los metadatos que, una vez obtenidos, ejecuta un proceso de reestructuración interno de los mismos para ser mostrados al usuario. En la Figura 7, puede observarse una interfaz de dicha asistencia, la cual presenta, a la izquierda, la taxonomía de servicios geográficos, 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 Interacción Humana / Visualizador Geográfico/ Manipulación de Mapas la variabilidad Opcional definida para el servicio Agregar Mapa junto con sus variantes (Clonar Mapa y Manejo y almacenamiento de múltiples mapas).

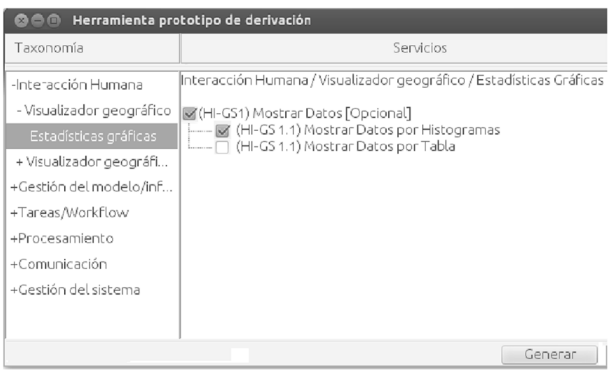

Figure 7: Intefaz para asistir al proceso de derivacion de productos de la LPS en base al sistema de anotaciones implementado en los componentes.

El resultado de este proceso de asistencia es el código instanciando las variantes seleccionas de todas las variabilidades presentes. El sistema de anotaciones definido está configurado para que luego de realizar la derivación de un determinado producto, sólo permanezcan en el código aquellos metadatos que asisten a la integrabilidad de componentes, eliminando aquellos que definen la variabilidad. Esto implica que el producto derivado tendrá un código mas legible y limpio, y a su vez contar con la información necesaria para permitirá añadir las funcionalidades particulares de cada producto. De esta forma, el modelo de metadatos queda parcialmente oculto a los potenciales productos derivados.

CONCLUSIONES Y TRABAJO FUTURO

En este trabajo hemos definido un sistema de anotaciones sobre componentes JAVA que asisten a la gestión de la variabilidad y a la integrabilidad de los componentes. El principal aporte del sistema de anotaciones es un modelo de metadatos generado, el cual contempla la información mas relevante a ser mantenida dentro de los componentes para maximizar su uso, tanto en la derivación de los productos como en la integración de unos con otros. Nuestros trabajos previos, en donde hemos generado varios artefactos para el desarrollo una LPS basada en componentes reusables, nos brindaron el soporte necesario para aplicar el sistema de anotaciones sobre experiencias reales que puedan medir el impacto de la aplicación del mismo y los beneficios obtenidos. Si bien sabemos que el uso de las anotaciones agrega una tarea mas a los desarrolladores y es susceptible a errores cometidos al completar los metadatos, simplifica luego la tarea de un análisis automatizado mediante una herramienta que dependa de ellos. Para ilustrar esto, hemos desarrollado una herramienta prototipo que utiliza el sistema de anotaciones para soportar el

proceso de derivación de nuevos productos en una LPS y permite recuperar sus requerimientos técnicos y semánticos. Como trabajo futuro, se propone construir, por un lado, herramientas que asistan al proceso de anotación de los componentes para minimizar la posible tasa de error en el rellenado de los mismos. A su vez, es necesario conducir tareas de validación que permitan evaluar este sistema de anotaciones mediante dos parámetros principales: medir costo y esfuerzo de reimplementar el sistema de anotaciones sobre componentes previamente desarrollados, y medir los beneficios obtenidos a la hora de derivar nuevos productos y utilizar su especificación para fines como la trazabilidad o composición.

AGRADECIMIENTOS

Queremos agradecer a dos organizaciones que están colaborando activamente en este proyecto: IBMPAS y CENPATCONICET. Este trabajo esta parcialmente soportado por el proyecto UNComa "Reuso de Software orientado a Dominios" parte del programa 04/F001: "Desarrollo de Software Basado en Reuso".

BIBLIOGRAFÍA

C. A. Szyperski, Component software - beyond objectoriented programming. Addison-Wesley-Longman, 1998.

K. Pohl, G. B¨ockle, and F. J. v. d. Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

P. C. Clements and L. Northrop, Software Product Lines : Practices and Patterns. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

J. Bosch, Design and use of software architectures: adopting and evolving a product-line approach. New York, NY, USA: ACMPress/Addison-Wesley Publishing Co., 2000.

M. A. Babar, L. Chen, and F. Shull, "Managing variability in software product lines," IEEE Software, vol. 27, pp. 89–94, 2010.

L. Chen and M. A. Babar, "A systematic review of evaluation of variability management approaches in software product lines," Information and Software Technology, vol. 53, no. 4, pp. 344 – 362, 2011.

M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques: Research articles," Software-Practice & Experience, vol. 35, no. 8, pp. 705–754, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1002/spe.v35:8.

G. T. Heineman andW. T. Councill, Eds., Component based software engineering: putting the pieces together. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

A. Buccella, A. Cechich, M. Arias, M. Pol'la, M. del Socorro Doldan, and E. Morsan, "Towards systematic software reuse of gis: Insights from a case study," Computers & Geosciences, vol. 54, no. 0, pp. 9 – 20, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098300412003913

M. Pol'la, A. Buccella, A. Cechich, and M. Arias, "Un modelo de metadatos para la gestión de la variabilidad en lıneas de productos de software," in Proceedings of the ASSE'14: 15th Simposio Argentino de Ingenieria de Software, Buenos Aires, Argentina, 2014.

M. Polla, A. Buccella, A. Cechich, S. Doldan, E. Morsan, and M. Arias, "Deteccion de patrones de distribucion en ecologıa marina: Un caso de estudio," in Proceedings of the CONAIISI'13: 1st Congreso Nacional de Ingenieria Informatica, Cordoba, Argentina, 2013.

P. Pernich, A. Buccella, A. Cechich, S. Doldan, E. Morsan, M. Arias, and M. Polla, "Product-line instantiation guided by subdomain characterization: A case study," Journal of Computer Science and Technology, Special Issue 12(3), vol. 12, no. 3, pp. 116–122, 2012.

M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou, "Variability in software systems - a systematic literature review," IEEE Transactions on Software Engineering, vol. 40, no. 3, pp. 282–306, 2014.

K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA., Technical Report CMU/SEI-90-TR-21, 1990.

K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A.Wäsowski, "Cool features and tough decisions: a comparison of variability modeling approaches," in Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, ser. VaMoS '12. New York, NY, USA: ACM, 2012, pp. 173–182.

A. Haber, H. Rendel, B. Rumpe, I. Schaefer, and F. van der Linden, "Hierarchical variability modeling for software architectures," in Software Product Lines - 15th International Conference, 2011, pp. 150–159.

M. Helvensteijn, R. Muschevici, and P. Wong, "Delta modeling in practice: A fredhopper case study," in Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, ser. VaMoS '12. New York, NY, USA: ACM, 2012, pp. 139–148.

I. Schaefer, L. Bettini, F. Damiani, and N. Tanzarella, "Delta-oriented programming of software product lines," in Proceedings of the 14th International Conference on Software Product Lines: Going Beyond, ser. SPLC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 77–91. [Online]. Available: http://dl.acm.org/citation.cfm?id=1885639.1885647.

I. Kim, D. Bae, and J. Hong, "A component composition model providing dynamic, flexible, and hierarchical composition of components for supporting software evolution," Journal of Systems and Software, vol. 80, no. 11, pp. 1797 – 1816, 2007. H. Mei, F. Chen, O. Wang, and Y. Feng, "Abc/adl: An adl supporting component composition," in Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering, ser. ICFEM '02. London, UK, UK: Springer-Verlag, 2002, pp. 38–47. [Online]. Available: http://dl.acm.org/citation.cfm?id=646272.685804.

G. Succi, R. Wong, E. Liu, and M. Smith, "Supportingdynamic composition of components," in Software Engineering, 2000. Proceedings of the 2000 International Conference on, 2000.

S. Thaker, D. Batory, D. Kitchin, and W. Cook, "Safe composition of product lines," in Proceedings of the 6th International Conference on Generative Programming and Component Engineering, ser. GPCE '07. New York, NY, USA: ACM, 2007, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/1289971.1289989.

M. Brohi and F. Jabeen, "A metadata-based framework for object-oriented component testing," International Journal of Computer Applications, vol. 41, no. 15, pp. 8–18, March 2012, published by Foundation of Computer Science, New York, USA.

J. C. Grundy, "Aspect-oriented requirements engineering for component-based software systems," in Proceedings of the 4th IEEE International Symposiumon Requirements Engineering, ser. RE'99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 84–91. [Online]. Available: http://dl.acm.org/citation.cfm?id=647646.731259.

E. Martins, C. M. Toyota, and R. L. Yanagawa, "Constructing self-testable software components," in Dependable Systems and Networks, 2001. DSN 2001. International Conference on, July 2001, pp. 151–160.

A. Orso, M. Harrold, and D. S. Rosenblum, "Component metadata for software engineering tasks," in Revised Papers from the Second International Workshop on Engineering Distributed Objects, ser. EDO '00. London, UK, UK: Springer-Verlag, 2001,

pp. 129–144. [Online]. Available: http://dl.acm.org/citation.cfm?id=647869.737134.

J. A.. Stafford and A. l. Wolf, "Annotating components to support component-based static analyses of software systems," University of Colorado at Boulder, Tech. Rep., 1999, cU-CS 896-99.

P. Pernich, A. Buccella, A. Cechich, S. Doldan, E. Morsan, M. Arias, and M. Pol'la, "Developing a subdomain-oriented software product line," in Proceedings of the CACIC'11: 17th Congreso Argentino en Ciencias de la Computación, La Plata, Argentina, 2011.

N. Huenchuman, A. Buccella, M. Pol'la, , A. Cechich, S. Doldan, E. Morsan, and M. Arias, "Reingenieria de una lınea de productos de software: Un caso de estudio en el subdominio de ecologıa marina," in Proceedings of the CACIC'13: XIX Congreso Argentino de Ciencias de la Computacion, Mar del Plata, Argentina, 2013.

B. Y.. Alkazemi, "Towards a software product line framework to support cross-domain component composition," in 12th International Conference on Computational Science and Its Applications (ICCSA), 2012, pp. 130–133.

Fecha de Presentación: 15/05/15
Fecha de Aprobación: 08/06/15