Un modelo de metadatos para asistir en la composición dinámica de componentes de una Línea de Productos de Software

Maximiliano Arias, Agustina Buccella, Matías Pol'la, and Alejandra Cechich

GIISCO Research Group,
Departamento de Ingeniería de Sistemas - Facultad de Informática
Neuquén, Argentina
{maximiliano.arias,agustina.buccella,matias.polla,alejandra.cechich,}@fi.uncoma.edu.ar
http://giisco.uncoma.edu.ar

Abstract. El desarrollo basado en componentes y líneas de productos software se basa en la posibilidad de combinar piezas software como unidades de composición. Sin embargo, componer una aplicación software a partir de componentes existentes es todavía materia de estudio debido a la complejidad de las interacciones que en muchos casos deben adaptarse. En particular, realizar composiciones de manera automática implica desafíos aún mayores. En este trabajo, proponemos un modelo para el desarrollo de componentes con metadatos a modo de anotaciones para facilitar la composición. El modelo ha sido instanciado para el caso de plataformas de desarrollo Java y validado en el contexto de instanciación de una línea de productos software.

Keywords: Líneas de Productos de Software, Desarrollo de Software Basado en Componentes, Metadatos, Composición Dinámica

1 Introducción

Dentro del amplio conjunto de paradigmas de desarrollo de software basado en reuso se encuentran el Desarrollo de Líneas de Productos de Software (DLPS) [12] y el Desarrollo de Software Basado en Componentes (DSBC) [5]. Ambos pueden ser combinados de manera que las Líneas de Productos de Software (LPS) pueden implementarse utilizando componentes reemplazables en sus puntos de variabilidad. La composición de dichos componentes, para formar un producto o un nuevo componente, es generalmente realizada de forma manual, pudiendo diferenciar al menos dos tipos de composición [5, 14]. Si el componente se desarrolla de manera exclusiva para un punto de variabilidad específico, la composición puede realizarse sin ningún tipo de adaptación. Por otro lado, podría ocurrir que el componente desarrollado sea reutilizado en un dominio más general que aquel para el que fue desarrollado originalmente, o incluso podría haber sido desarrollado de manera externa, por otro grupo de desarrollo y para un propósito más general. En este caso, se necesita realizar una composición

adaptada, que permita conectarlo con el punto de acople. Al mismo tiempo, el proceso de composición puede verse afectado por un gran número de problemas [14] relacionados con incompatibilidades de interfaces, de operaciones, de excepciones, entre otros. Esto no implica que este proceso no pueda ser realizado de manera asistida o incluso automática [16]. Así, en este trabajo se ha diseñado, implementado y aplicado un sistema de metadatos para asistir en la composición dinámica de componentes reusables en una LPS. Para dicha tarea se realizó un análisis de los elementos que participan en el proceso de composición, se creó un proceso de definición de un sistema de metadatos y se utilizó para definir un sistema capaz de cumplir el objetivo propuesto. A su vez, se definió un proceso de aplicación y uso del sistema propuesto a los componentes de una LPS desarrollada en trabajos previos.

Este artículo se organiza de la siguiente manera, en la Sección 2 se presentan los antecedentes y trabajos relacionados, en las Secciones 3 y 4 se presenta la propuesta, explicando el proceso de construcción del mismo y una implementación particular. Luego, en la Sección 4.1 se muestra un caso de estudio realizado sobre trabajos previos y finalmente la Sección 5 presenta las conclusiones y trabajos futuros.

2 Trabajos relacionados y antecedentes

El problema de composición dinámica ha sido abarcado desde distintos acercamientos [7, 9, 16, 18] 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 número de tareas relacionadas con el proceso de desarrollo [2–4, 8, 10, 15] incluyendo testeo y validación, inclusión de descripciones en lenguaje natural para facilitar búsquedas en catálogos, configuración de herramientas, entre otras tareas.

En trabajos previos [6, 11, 13] se desarrolló una LPS para el subdominio de ecología marina, trabajando en forma conjunta con el Instituto de Biología Marina y Pesca Almirante Storni[1] (IMBPAS) y el Centro Nacional Patagónico[2] (CENPAT). Inicialmente la implementación se realizó en JavaScript [11] y fue posteriormente refactorizada [6, 13] haciendo uso de la plataforma JAVA[3] y Enterprise Java Beans[4]. Mediante estas herramientas se comenzó con la creación de componentes reusables para implementar los distintos puntos de variabilidad de la línea. Estos servicios se encuentran definidos en una taxonomía de servicios [13] extendida de la norma ISO/DIS 19119 para servicios de información geográfica.

[1] http://www.ibmpas.org
[2] http://www.cenpat.edu.ar/
[3] https://www.java.com/
[4] http://www.oracle.com/technetwork/java/javaee/ejb/index.html

3 Desarrollo de un modelo de metadatos para composición dinámica

Para asistir en el proceso de composición dinámica de componentes de una LPS se propone desarrollar un modelo de metadatos que sea capaz de llevar a cabo esta tarea. Para esto, hemos definido un proceso con tres etapas fundamentales que permiten la creación del mismo. En la Figura 1 podemos observar dichas etapas junto con un detalle de las actividades de la segunda etapa. Dicho detalle se debe a que para esta etapa hemos también creado un nuevo proceso que permite definir, generar, y recuperar los metadatos definidos en el modelo.

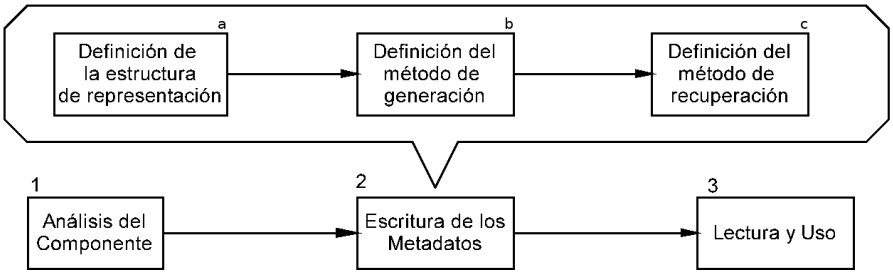

Fig. 1. Proceso de definición del modelo de metadatos para el desarrollo de un componente de una LPS

Cada una de las etapas se describen brevemente a continuación:

1. *Análisis del componente*: En un DSBC cada componente posee un conjunto de interfaces que le permiten describir sus funcionalidades y requerimientos de integración. De esta forma podemos identificar dos elementos: la interfaz funcional del componente [17] y los requerimientos (o interfaz) arquitecturales [1]. 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 esté 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 1 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, puede metadatos sean almacenados en archivos externos mediante el uso texto plano, archivos XML, entre otros. Por otro lado, puede suceder que los metadatos 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. Este proceso puede realizarse de manera manual, automática, o asistida. Esta tarea está 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 usará para generar o escribir los metadatos.

(c) *Definición del método de recuperación*: Por último, 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 además la necesidad de desarrollar una herramienta capaz de leer y mostrar los metadatos escritos.

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 está 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.

4 Implementación utilizando JAVA

En esta sección se describe la implementación en JAVA del proceso de definición del modelo de metadatos descripto previamente en la Figura 1. Como se puede observar en la Figura 2, en cada una de las etapas se obtienen una serie de diferentes artefactos que ayudarán a la construcción y uso del modelo.

En la primera etapa, *análisis del componente*, se realizó un estudio de los elementos que pueden identificarse en un componente JAVA. Debido a que JAVA

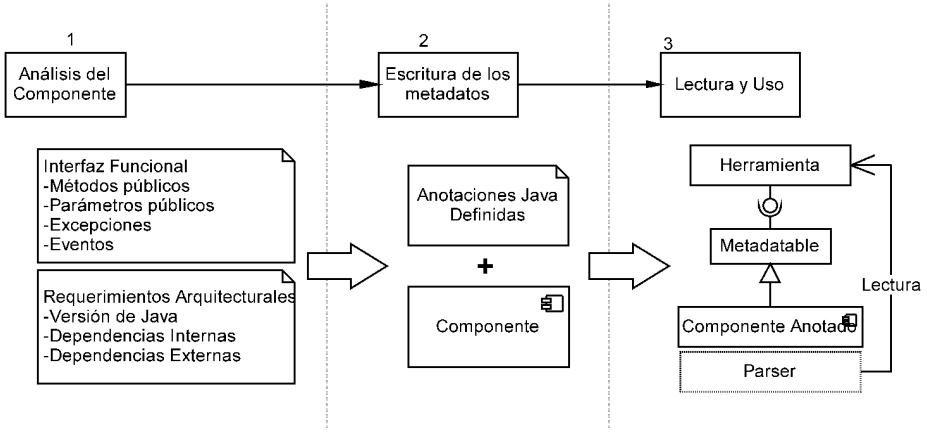

Fig. 2. Aplicación del sistema de metadatos para composición dinámica.

no define por defecto una estructura para la creación de componentes, toda información de interfaz, excepciones y eventos generados está 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 projecto JAVA pueden encontrarse en sus archivos de configuración (properties,classpath, etc). Estos archivos suelen generarse automáticamente por los entornos de desarrollo utilizados como Eclipse[5] o NetBeans[6] 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 artefactos necesarios para la escritura de los elementos identificados en la etapa anterior. El primer paso es definir un modelo para el problema estudiado. Así, las tres actividades involucradas (Figura 1) se aplicaron al desarrollo de un modelo de metadatos en JAVA y se detallan a continuación:

1. *Definir la estructura de representación de los metadatos*: Para asistir en la composición dinámica de componentes es primordial poseer la información pública del componente, es decir aquellos elementos que utiliza para comunicarse con otros componentes y para ser configurado. Por otro lado, se necesita conocer los requisitos que establece el mismo para poder ser acoplado. Con estos criterios se definió un conjunto mínimo de datos que se necesitan poseer: interfaz y atributos públicos (o variables de configuración), excepciones y eventos generados, versión mínima de JAVA necesaria para una correcta ejecución, y otros componentes y/o elementos de los que depende.

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

Para estructurar los metadatos 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 subcategoría. La etiqueta permite identificar los metadatos y clasificarlos según el ámbito al que se refieren. De esta manera se obtuvieron los metadatos que se pueden observar en la Tabla 1. Por ejemplo, podemos ver que el metadato *FuncionalInterface/Method* se encarga de mantener toda la información relacionada con la signatura de los métodos públicos definidos en un componente.

Etiqueta	Sub-Etiqueta	Datos Contenidos
FuncionalInterface	Method	Métodos públicos del componente
FuncionalInterface	PublicParameter	Parámetros públicos del componente
FuncionalInterface	Events	Eventos generados por el componente
FuncionalInterface	Exceptions	Excepciones que pueden generarse
ArchiteturalInterface	JavaVersion	Versión de Java mínima necesaria
ArchiteturalInterface	InteralDependencies	SubComponentes utilizados
ArchiteturalInterface	ExternalDependencies	Librerías, componentes, bases de datos, etc.

Table 1. Conjunto de metadatos defindos para el caso de estudio.

2. *Definir el método de generación*: Debido a que se trabaja sobre la plataforma JAVA se eligió un método de generación acorde a la misma, haciendo uso de las librerías que dan soporte a JAVA Annotations[7]. De esta manera se creó una anotación para cada uno de los elementos que se pueden ver en la Tabla 1, se les asignó un nombre descriptivo con el prefijo "Meta" y se estableció para cada una de ellas un objetivo. Los objetivos de las anotaciones JAVA indican qué elemento del código anotan, es decir qué elemento debe estar escrito exactamente después de la anotación.
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 sólo requiere que las clases que los utilizan implementen la interfaz "Metadatable". Se

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

creó además una herramienta que respeta estos requerimientos y permite visualizar en forma de tabla los metadatos contenidos en un archivo JAVA.

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

4.1 Caso de estudio

La librería desarrollada se aplicó a la implementación de un nuevo componente para la LPS definida en trabajos previos. La misma se enfoca en la instanciación de Sistemas de Información Geográfica (SIG) dentro del dominio de ecología marina. Para su implementación se hace uso del framework Google Web Toolkit[8] (GWT), la librería OpenLayers[9] y la base de datos Postgres[10] con su correspondiente extension para SIG "PostGis[11]". Los servicios de la LPS se implementan mediante uno o varios componentes interconectados y acoplados a la interfaz gráfica por medio de un gestor de interfaces gráficas denominado UIManager. El objetivo del componente es el de implementar un servicio de historial de mapas haciendo uso de algunos componentes previamente definidos y bibliotecas externas. Este componente es capaz de clonar y generar nuevos mapas para trabajar en paralelo durante el estudio distribuciones de especies marinas. Por lo tanto, se requiere la habilidad de generar y mantener un historial de mapas derivados. La Figura 3 muestra una parte del diagrama de componentes de la LPS. En ella se puede observar el nuevo componente HistorialMapa, que hace uso del componente Mapa y es utilizado por el componente UIManager a través de la interfaz definida. Para su correcto funcionamiento, este componente hace uso de un conjunto de librerías externas dentro de las cuales se encuentran GWT, OpenLayers, ui-binder[12], validation-api[13] y otras. En cuanto a las excepciones y eventos, el componente no genera eventos pero si genera excepciones relacionadas con la existencia de un mapa que se quiera mostrar, ocultar o quitar. Como resultado de este proceso se obtiene un componente anotado capaz de ser acoplado a cualquier herramienta que desee utilizar el conjunto de metadatos.

Para validar el funcionamiento del sistema se creó una herramienta de lectura capaz de leer y mostrar de forma tabular los metadatos escritos. Estos datos pueden luego ser utilizados por una herramienta de composición dinámica

[8] http://www.gwtproject.org/
[9] http://openlayers.org
[10] http://www.postgresql.org.es
[11] http://postgis.net
[12] http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html
[13] http://mvnrepository.com/artifact/javax.validation/validation-api

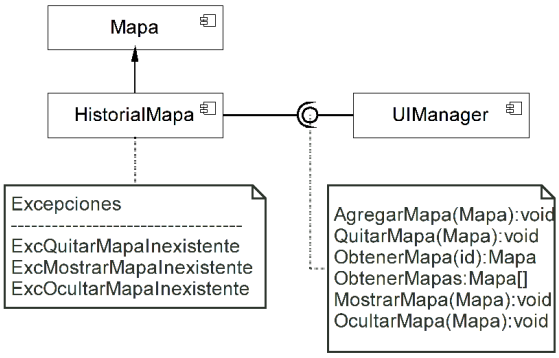

Fig. 3. Parte del diagrama de componentes de la LPS que involucra al componente HistorialMapa

otorgándole a la misma la capacidad de conocer los detalles internos de un componente cerrado sin tener que realizar ingeniería inversa sobre el mismo. La Figura 4 muestra el proceso de utilización de los metadatos para composición manual y automática. En ella podemos observar como los metadatos escritos en un componente son leídos haciendo uso del parser correspondiente y obteniendo información organizada y estructurada. Una herramienta de composición dinámica es capaz de leer esta información ya estructurada y utilizarla para realizar una verificación de compatibilidad (y generación de adaptadores si fuera necesario). Por otro lado, el uso de la herramienta de lectura y visualización de la información estructurada puede facilitar la composición manual.

A pesar de que nuestro proceso no fue validado formalmente, es evidente que si éste se realizara sin la utilización de los metadatos se debería efectuar un análisis manual de los componentes que intervienen en la composición para extraer la información necesaria. Si posteriormente la composición se quisiera realizar de manera automática, la información analizada debería además ser estructurada según lo establecido por la herramienta a utilizar. Esta complejidad crece a medida que crece el número de componentes involucrados. Con la utilización de metadatos, cada componente está encargado de declarar su propia información y estructurarla, eliminando la necesidad de análisis. Finalmente, al declarar una interfaz pública de acceso a los metadatos, se puede brindar acceso a la estructura interna de un componente cerrado y funcionan a su vez como una forma de autodocumentación.

5 Conclusiones y trabajos futuros

A modo de conclusión se puede resaltar que un sistema de metadatos puede asistir en distintas tareas del proceso de desarrollo de software de manera ad-hoc y por tal motivo hemos definido un proceso general para el desarrollo y uso de los mismos de manera repetida y consistente. Si bien el uso del proceso agrega una tarea a los desarrolladores y es susceptible a errores cometidos al completar los

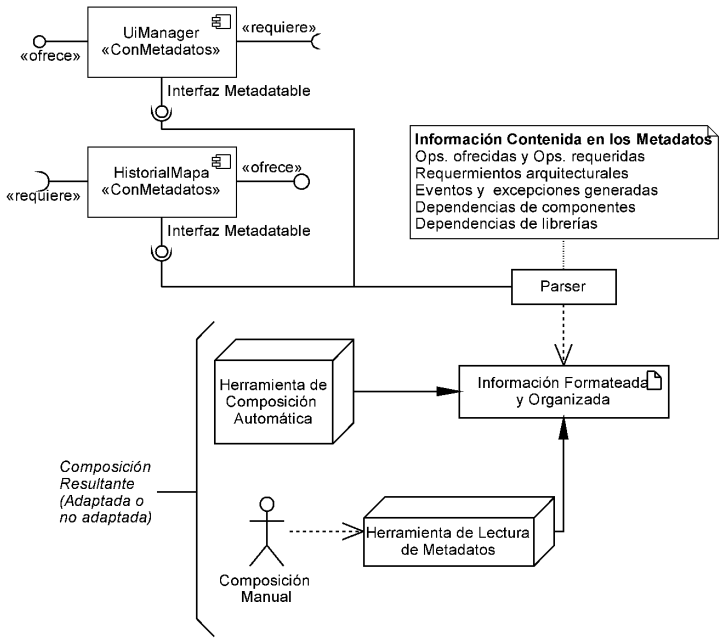

Fig. 4. Proceso de lectura y utilización de los metadatos para composición manual y automática.

campos de las anotaciones, simplifica la tarea de análisis automatizado mediante una herramienta que dependa de ellos. De esta manera nuestro trabajo propone un punto de partida para asisitr a la composición dinámica de componentes. Por otro lado, como se expuso anteriormente, al realizar manualmente las tareas de escritura se expone al sistema a un factor de error humano. Por esta razón se propone además como trabajo futuro la automatización de las mismas mediante un analizador de código, minimizando el problema del esfuerzo de desarrollo agregado.

References

1. B.Y. Alkazemi. Towards a software product line framework to support cross-domain component composition. In *Computational Science and Its Applications (ICCSA), 2012 12th International Conference on*, pages 130–133, 2012.
2. M Nawaz Brohi and Fakhra Jabeen. Article: A metadata-based framework for object-oriented component testing. *International Journal of Computer Applications*, 41(15):8–18, March 2012. Published by Foundation of Computer Science, New York, USA.
3. John C. Grundy. Aspect-oriented requirements engineering for component-based software systems. In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, RE '99, pages 84–91, Washington, DC, USA, 1999. IEEE Computer Society.

4. Mary Jean Harrold, Alessandro Orso, David Rosenblum, Gregg Rothermel, and Mary Lou Soffa. Using component metadata to support the regression testing of component-based software, 2001.
5. George T. Heineman and William T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
6. Natalia Huenchuman. Reestructuración de una línea de productos de software para el subdominio de ecología marina, 2013.
7. In-Gyu Kim, Doo-Hwan Bae, and Jang-Eui Hong. A component composition model providing dynamic, flexible, and hierarchical composition of components for supporting software evolution. *Journal of Systems and Software*, 80(11):1797 – 1816, 2007.
8. 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*, pages 151–160, July 2001.
9. Hong Mei, Feng Chen, Qianxiang Wang, and Yao-Dong Feng. Abc/adl: An adl supporting component composition. In *Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering*, ICFEM '02, pages 38–47, London, UK, UK, 2002. Springer-Verlag.
10. Alessandro Orso, Mary Jean Harrold, and David S. Rosenblum. Component metadata for software engineering tasks. In *Revised Papers from the Second International Workshop on Engineering Distributed Objects*, EDO '00, pages 129–144, London, UK, UK, 2001. Springer-Verlag.
11. Patricia Pernich. Diseño e implementación de una arquitectura de líneas de productos para servicios gis, 2011.
12. K. Pohl, G. Bockle, and F. Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.
13. Matias Pol'la. Instanciación y validación de una línea de productos de software aplicada al subdominio de la ecología marina, 2013.
14. Ian Sommerville. *Software Engineering*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 9. edition, 2010.
15. Judith A. Stafford and Alexander L. Wolf. Annotating components to support component-based static analyses of software systems. In *IN PROCEEDINGS OF GRACE HOPPER CONFERENCE 2000 (TO APPEAR), HYANNIS*, 2000.
16. G. Succi, R. Wong, E. Liu, and M. Smith. Supporting dynamic composition of components. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 787–, 2000.
17. Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1998.
18. Sahil Thaker, Don Batory, David Kitchin, and William Cook. Safe composition of product lines. In *Proceedings of the 6th International Conference on Generative Programming and Component Engineering*, GPCE '07, pages 95–104, New York, NY, USA, 2007. ACM.