Generación de Código JAspect a partir de un Metamodelo Orientado a Aspectos

Corina Abdelahad, Daniel Riesco, Germán Montejano
Departamento de Informática
Facultad de Ciencias Físico-Matemáticas y Naturales
Universidad Nacional de San Luis
Ejercito de los Andes 950 – C.P. 5700 - San Luis- Argentina
{cabdelah, driesco, gmonte}@unsl.edu.ar

Resumen. En la actualidad no existe un estándar para soportar Diseños Orientados a Aspectos (DOA). La OMG tiene estándares que permiten la definición de Perfiles. A través de estos Perfiles es posible definir estereotipos. Muchas herramientas soportan estos estándares y hacen posible la definición de aspectos utilizando estereotipos para lograr modelos Orientados a Aspectos (OA). Utilizando una herramienta que permite diseñar un modelo OA es posible la construcción de un software que genere código OA. Este software será independiente a las herramientas que se utilicen para el modelado OA que sigan con las especificaciones de la OMG.
Este trabajo está enfocado en mostrar la semántica del Perfil OA mediante su equivalencia en código OA. Para esto, se ha llevado a cabo la transformación de un DOA a código OA haciendo uso del metamodelo de la OMG. Este aporte ayuda al ingeniero de software en el desarrollo de software OA, agilizando su tarea.
Palabras claves: Aspecto, Programación Orientada a Aspectos, Perfil UML, XML.

1 Introducción

La POA es una nueva metodología de programación que aspira a soportar la separación de los aspectos que entrecruzan y atraviesan todo el sistema *(crosscutting concern)* y que no pueden ser encapsulados con las técnicas tradicionales. Un aspecto es la unidad central en un lenguaje OA, de la misma manera que la clase es la unidad central en POO.

Por otro lado, los estándares de la OMG brindan la posibilidad de definir Perfiles [1], [2] los cuales hacen posible la construcción de diseños OA, ya que en la actualidad no existe un estándar en cuanto a estos diseños. A través de estos perfiles, es posible definir estereotipos para lograr el soporte de aspectos. La semántica que tendrán estos estereotipos parte de la correspondencia con el código OA, utilizando reglas/operaciones OCL [3].

Asimismo, muchas herramientas open source que sigan las especificaciones de la OMG, permiten definir perfiles y de esta manera se hace posible la construcción de una herramienta que genere código OA independiente de las herramientas utilizadas para la construcción del modelo. Es decir, la herramienta que genere código OA podrá

tomar como fuente cualquier herramienta de modelado que estén bajo las especificaciones de la OMG.

Además, de la misma manera que cuando se diseña un modelo en UML se puede generar código en un lenguaje específico, en este trabajo se muestra cómo es posible construir un modelo OA y generar código OA.

2 Diseño de Perfiles OA

La infraestructura de UML se define a través de una librería [4], ella define un metalenguaje base (metalanguage core) que puede ser reusado para definir metamodelos (incluido UML), además permite personalizar UML por medio de los Perfiles UML [5].

La extensión del perfil permite agregar al metamodelo, constructores propios para un dominio particular, sin modificar el existente [6].

En esta sección se muestra un resumen de la definición de un perfil para el modelado de los sistemas OA. El perfil es un paquete estereotipado que contiene elementos del modelo que han sido diseñados para soportar el DOA.

Para lograr definir un estereotipo se lo debe asociar con elementos del metamodelo (clase, operación, etc.) [7].

El estereotipo aspect permite modelar los aspectos de un sistema y es una extensión de la metaclass *Class*. El estereotipo realize modela la relación entre una clase y un aspecto. Este estereotipo indica que una clase usa uno o más aspectos, y es una extensión de la metaclass *Association*. El estereotipo pointcut permite modelar los puntos de cortes; actúa como filtro y está definido por un conjunto de puntos de enlace, además es una extensión de la metaclass *Operation*. De igual forma, el estereotipo advice es una extensión de la metaclass *Operation* y especifica una conducta que quiere ejecutar antes de (before), después de (after) o alrededor de (around) un punto de enlace.

Estas extensiones significan que una clase puede ser un aspecto, una asociación puede ser una relación realize, y una operación puede ser un pointcut o un advice.

La Figura 1 muestra parcialmente la propuesta utilizada para la definición de un perfil UML el cual ha sido definido en [3], para lograr modelar sistemas OA.

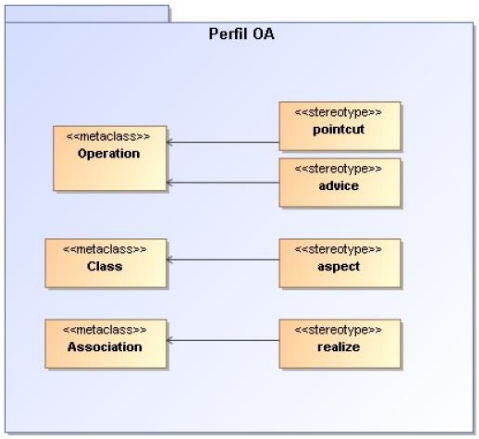

Figura 1: Definición de Perfil.

La figura 2 muestra como se aplican los estereotipos definidos anteriormente.

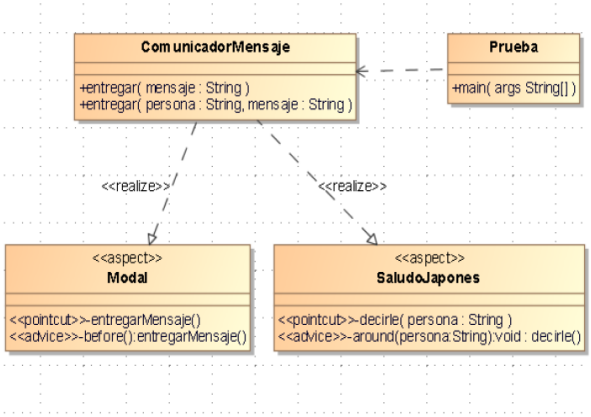

Figura 2. Ejemplo: Diseño Orientado Aspectos.

La ventaja de utilizar un estándar como UML es que no sólo permiten definir estereotipos para el soporte de aspectos sino además permiten guardar los documentos en un formato estandarizado como XML [8]. De esta manera es posible la construcción de una herramienta que genere código OA independientemente de las herramientas utilizadas para el modelado. Partiendo de esto, se construye el diseño, importando y aplicando la definición del Perfil OA visto anteriormente para el soporte de aspectos. El elemento al cual se le aplica el estereotipo tiene la misma estructura y comportamiento que el elemento base, pero su semántica está restringida mediante reglas/operaciones OCL.

A continuación se muestra ejemplos de operaciones y reglas OCL [9]:

IsAspect() es una operación que se utiliza para identificar si la clase está estereotipada como un aspecto:

Context Class:: IsAspect():Boolean
Pre:
Post: IsAspect= self.extension.ownedEnd- > exist(e/e.type.name='Aspect')

Se define la operación IsPointcut() de manera que indique si en las clases existe una operacion estereotipada como Pointcut:

Context Class::IsPointCut():Boolean
Pre:
Post: IsPointCut= self.extension.ownedEnd-> exist(e/e.name="PointCut")

La operación Pointcut sólo pueden pertenecer a un aspecto. Esta operación es propia de la OA, por lo que no pueden existir en las clases que no sean aspectos:

Context Operation
Inv: self.IsPointCut() implies self.class.IsAspect()

La OMG también da una especificación relacionada con UML la cual permite intercambiar información entre distintas herramientas. Esta especificación lleva el nombre de XMI (XML Metadata Interchange) y está basado en XML. En este trabajo se especifica un subconjunto pequeño de un modelo XML de la OMG. Para ello se define un esquema XML [10], el cual define una estructura válida para un tipo de documento XML.

La figura 3 muestra esquemáticamente una estructura la cual está constituida por clases, relaciones y operaciones. Esta estructura es necesaria para guardar toda la información que será relevante para la generación de código OA. En dicha estructura estarán las clases, junto con sus atributos y operaciones, además de las relaciones. Asimismo, contiene información sobre que clases están estereotipadas con aspect, como así también que relaciones están estereotipadas con realize, y que operaciones están estereotipadas con pointcut o advice.

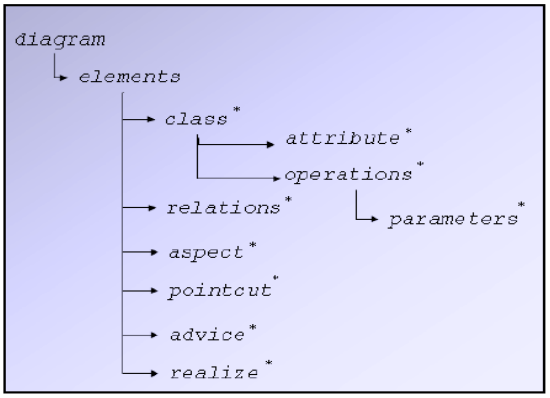

Figura 3: Descripción del esquema XML

3 Transformación de un Perfil OA a Código AspectJ

Partiendo de lo visto anteriormente, se lleva a cabo la construcción de un diseño OA a través de la aplicación de un Perfil OA el cual está basado en el metamodelo de la OMG. Una vez definido un diseño OA de un sistema particular, basado en el metamodelo de la OMG, se obtiene un metamodelo expresado en XMI, el cual representa la semántica de dicho diseño.

Con el fin de obtener el código significativo para la generación de código OA, se utilizó tecnología Xquery [11], en particular se utilizó un motor XQuery llamado Saxon [12] el cual es opensource bajo licencia GPL.

La Figura 4 muestra parte de una consulta sobre el metamodelo OA expresado en XMI para obtener información necesaria para la generación de código OA.

En base a este tipo de consultas es posible extraer y transformar el código XMI, generado por la herramienta del modelado. A través de esta transformación se logra construir un metamodelo en código XML, con el fin de que puedan interactuar las herramientas de modelado (que sigan con el estándar de la OMG) y la herramienta que genera código OA.

```
<diagram>
        <elements>{
         for $a in //uml:Model/ownedMember
         for $b in //xmi:XMI
         return if ($a/@xmi:type="uml:Class")  then
                    <class name="{data($a/@name)}"
                   id="{data($a/@xmi:id)}">
```

Figura 4: Ejemplo parcial de una consulta en Xquery.

Del metamodelo se obtienen las metaclases expresadas como *Class* las cuales están directamente involucradas con la OA ya sea o porque es una clase estereotipada con aspect, o porque es una clase relacionada con un aspecto a través de la relación realize, la cual indica que una clase se ve afectada por uno o más aspectos.

Del mismo modo, se obtienen del metamodelo, generado por la herramienta de modelado, las operaciones estereotipadas con pointcut o advice.

De esta manera se trasforma el metamodelo XMI a un metamodelo XML, para el cual se ha definido un esquema OA. Dicho esquema XML contendrá todo el código significativo para la generación de código.

Una vez que se ha generado el metamodelo XML se realiza la transformación de código XML a AspectJ [13]. Esta transformación se hace por medio de un programa escrito en lenguaje de programación Java. Dicho programa parsea el metamodelo XML generando de esta manera un archivo .java el cual contendrá código OA en el lenguaje de programación JAspect.

La Figura 5 muestra los pasos seguidos para la construcción de la herramienta que genera código AspectJ.

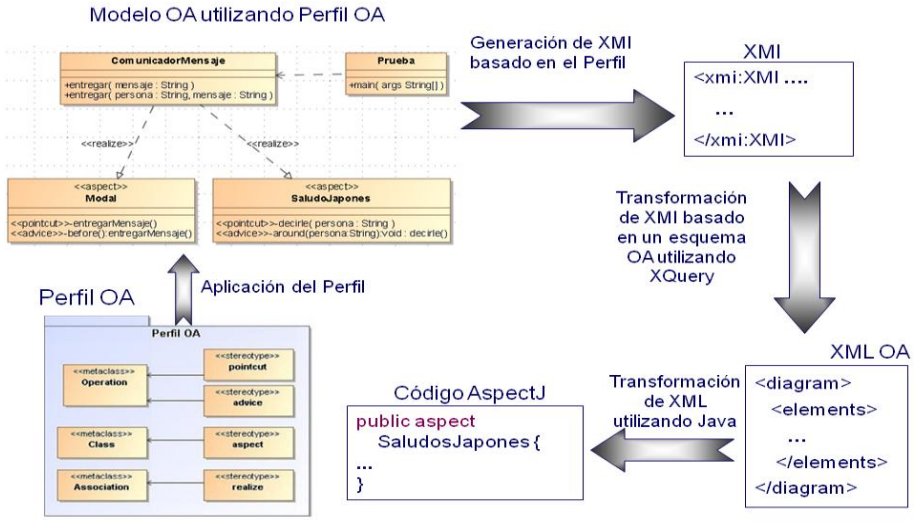

Figura 5: Transformaciones

5 Conclusiones

Tanto el diseño como el modelado de sistemas ocupan un lugar importante en la ingeniería de software. Los modelos brindan la posibilidad de abstraer sistemas y facilitar la implementación. La construcción de buenos modelos asegura un correcto desarrollo de la arquitectura del sistema.

El objetivo de este trabajo fue definir la semántica de un perfil OA a través del metamodelo de la OMG, con reglas OCL y su correspondencia con un lenguaje de programación OA. Para ello se ha desarrollado una herramienta que genera de manera automática código Orientado a Aspectos (OA) a partir de un modelo construido con el lenguaje UML.

Para realizar un modelo OA se utilizó una extensión de UML para el soporte de aspectos: los Perfiles. Asimismo, las herramientas que están bajo las especificaciones de la OMG, como UML, no sólo permiten definir estereotipos para el soporte de aspectos sino además guardar los documentos en un formato estandarizado como XMI.

La ventaja de utilizar un estándar como UML para construir el modelo de diseño es que hay una gran cantidad de herramientas en el mercado que se pueden utilizar. Éstas posibilitan la definición de estereotipos, que pueden ser usados para el soporte de aspectos. También permiten guardar los documentos como metamodelos en un formato estandarizado como XML (eXtensible Markup Language). Esto ayuda a que la herramienta que genera código OA sea independiente de las herramientas utilizadas para el modelado. La importancia de utilizar estándares radica en que garantiza que la herramienta que genera código OA es portable a distintos tipos de herramientas.

Basándonos en el metamodelo de la OMG, se construyó el metamodelo XML a través del cual es posible automatizar el desarrollo de software construyendo una herramienta que genere código OA, ayudando de esta manera al ingeniero de software, agilizando la construcción de sistemas.

6 Referencias

1. OMG Unified Modeling Language Specification. http://www.omg.org
2. Jingjun Zhang; Yuejuan Chen; Guangyuan Liu; Modeling Aspect-Oriented Programming with UMLProfile, 2009. First International Workshop on Education Technology and Computer Science. ETCS '09.
3. N. Debnath L. Baigorria, D. Riesco, G. Montejano “Metrics Applied to Aspect Oriented Design Using Uml Profiles” IEEE symposium on computers and communications 2008 (ISCC 2008) Press Marruecos.
4. UML 2.1.2 “Infrastructure Specification” http://www.omg.org/spec/UML/2.2/ 2009
5. N.C. Debnath, A. Garis, D. Riesco, G. Montejano “Defining Patterns Using UML Profiles” Computer Systems and Applications, 2006. IEEE Press.
6. Fuentes L. Vallecillo A., Johson R., Vlissides J. “Una Introducción a los Perfiles UML” Novotica Vol. 168, pg. 6-11. 2004
7. A. Zakaria, H. Hosny, A.Zeid; “A UML extension for Aspect Oriented Systems”
8. XMI http://www.w3.org

9. Baigorria L. Montejano G. Riesco D. “Definición y Aplicación de Perfil UML para AspectJ”; ASSE; 39 Jornadas Argentinas de Informática; 2010. Buenos Aires. Universidad Argentina de la Empresa (UADE).
10. XML Schema Definition http://www.w3.org/standards/xml/schema
11. Xquery http://www.w3.org/TR/XQuery/
12. http://www.saxonica.com/documentation/index/gettingstarted/gettingstartedjava.html
13. R. Laddad “AspectJ in Action” Practical Aspect-Oriented Programming , Manning Publications Co. 2003