Definición del Diseño Orientado a Aspectos según el Metamodelo de la OMG

Corina Natalia Abdelahad, Daniel Riesco, Lorena Baigorria, Norma Beatriz Perez

Departamento de Informática, Universidad Nacional de San Luis,
+54(2652)424027 — Fax: +54(2652)430224
Ejército de los Andes 950
5700 — San Luis, Argentina.
e-mail:{cabdelah,driesco,flbaigor,nbperez}@unsl.edu.ar

Resumen Con la evolución de la Ingeniería de Software, se ha introducido conceptos que llevan a una programación de más alto nivel. Debido a que la Programación Orientada a Objetos (POO) posee aspectos que no pueden encapsularse aumentando la indertependencia entre las clases lo cual no es deseable.
El enfoque de nuestro trabajo esta dirigido a la definición de un diseño Orientado a Aspecto (OA) basado en el metamodelo de la OMG, siendo éste un aporte con el fin de agilizar el desarrollo de software con la construcción de una herramienta que genere código OA. Como en la actualidad no hay un estándar en cuanto al diseño OA, proponemos construir el diseño a través del lenguaje estandarizado UML utilizando los mecanismos de extensión que éste provee. Se define un metamodelo en XML con la finalidad de utilizar distintas herramientas ya sea para el modelado como para la generación de código OA.

Palabras claves: UML, POO, POA, XML, XQuery, aspecto, perfil, estereotipo, punto de corte, aviso.

1. Introducción

La Ingeniería de Software tiene como objetivo construir o mejorar un producto de software.

En este trabajo se prestará especial atención en la etapa 3, que tiene como propósito crear una abstracción de la implementación (refinamiento directo del diseño), permitiendo así el uso de tecnologías como la *generación de código* y la *ingeniería de ida y vuelta* entre el diseño y la implementación. Por este acercamiento entre el diseño y la implementación es que, el diseño detallado está directamente relacionado con los lenguajes de programación y en esta etapa, se deben construir modelos dependiendo del tipo de programación que se utilice para el desarrollo del software.

A continuación se brinda una descripción de los tres pilares fundamentales que intervienen en nuestro trabajo:

Programación Orientada a Aspectos (POA). Lenguaje basado en la POO, que provee soporte explícito para tratar los aspectos que entrecruzan y atraviesan todo el sistema (*crosscutting concern*) y que no pueden ser totalmente separados con las técnicas de POO ya que ésta tiene como principal desventaja la obtención de ejecuciones ineficientes debido a que las unidades de descomposición no van siempre acompañadas de un buen tratamiento de los aspectos tales como sincronización, manejo de errores y manejo de excepciones, administración de memoria, gestión de seguridad entre otros.

UML. Lenguaje iniciado por la organización OMG (Object Management Group) que se utiliza para modelar la mayoría de los dominios, pero no todos los dominios son factibles de ser modelados en este lenguaje, para esto UML incluye características de extensión [1].

Cuando se diseña un modelo en UML se puede generar código en un lenguaje específico, de igual manera se puede construir un *modelo OA* y poder generar código para distintos tipos de lenguajes OA.

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 aplicar. Éstas no sólo permiten definir *estereotipos* para el soporte de aspectos sino además guardar los documentos en un formato estandarizado como XML (eXtensible Markup Language) [2, 3, 4], posibilitando una factible construcción de una herramienta que genere código OA siendo ésta independiente de las herramientas utilizadas para el modelado.

XQuery. Lenguaje de consulta, desarrollado por el grupo de trabajo en consultas XML de la W3C [5]. Se diseñó para escribir de manera rápida y compacta consultas sobre bases de datos con formato XML, siendo su principal función proporcionar los medios para extraer y manipular información de documentos XML o de cualquier fuente de datos que pueda ser representada mediante XML. Cada consulta es una expresión que es evaluada y devuelve un resultado [6].

2. Orientación a Aspectos

Los aspectos son propiedades de un software que tienden a atravesar sus principales funcionalidades. Formalmente se define:

Un aspecto: *es una unidad modular que se disemina por la estructura de otras unidades funcionales. Ellos existen tanto en la etapa de diseño como en la etapa de implementación* [7].

Por lo general los aspectos están diseminados por todo el sistema causando el problema de tener código desordenado. La OA tiene como objetivo soportar la separación de dichos aspectos encapsulándolos en módulos en vez de tenerlos dispersos en los componentes del sistema.

Para lograr escribir programas OA se utiliza una clase especial de lenguajes llamados *Lenguajes OA*. Ellos definen la manera de encapsular las funcionalidades que cruzan todo el código. Además, estos lenguajes deben soportar la separación de los aspectos vistos anteriormente. Sin embargo, estos conceptos no son totalmente independientes, y está claro que hay una relación entre los componentes y los aspectos, y que por lo tanto, el código de los componentes y de estas nuevas unidades de programación tienen que interactuar de alguna manera. Para que ambos (aspectos y componentes) se puedan combinar, se necesita una interacción entre el código de los componentes y el código de los aspectos. Esta interacción se logra teniendo puntos en común, conocidos como *puntos de enlace*, y debe haber algún modo de mezclarlos. El encargado de realizar este proceso de mezcla se lo conoce como *tejedor* (weaver), ayudado por los puntos de enlace él se encarga de mezclar los diferentes mecanismos de abstracción y composición que aparecen en los lenguajes de aspectos y componentes [8].

3. Diseño OA

UML no es lo suficientemente expresivo, como se menciono en las secciones anteriores, como para representar conceptos específicos de dominios particulares. Por esta razón, UML estándar incluye un mecanismo para extender y adaptar UML a diferentes dominios y plataformas: el *Perfil UML* (UML Profile). La infraestructura de UML se define a través de una librería [9], 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.

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

El perfil UML incluye los elementos necesarios para hacer posible la extensión del lenguaje. Dichos elementos son.

- `Estereotipos (stereotypes).`
- `Valores etiquetados (tags values).`
- `Restricciones (constrains).`

Un estereotipo permite crear nuevos tipos de bloques de construcción parecidos a los existentes, pero que son específicos a un problema particular, para más detalle ver [1]. Durante su definición, se lleva a cabo su asociación con elementos del metamodelo (clase, operación, etc.). Dentro de los modelos se denota un estereotipo de la siguiente manera: «`stereotype-name`» [10].

Los perfiles UML permiten extender no sólo la sintaxis sino además la semántica UML para modelar elementos de dominios particulares. En este trabajo hacemos uso de la ventaja de los mecanismos de extensión que brinda UML, cómo se muestra en la Figura 1.

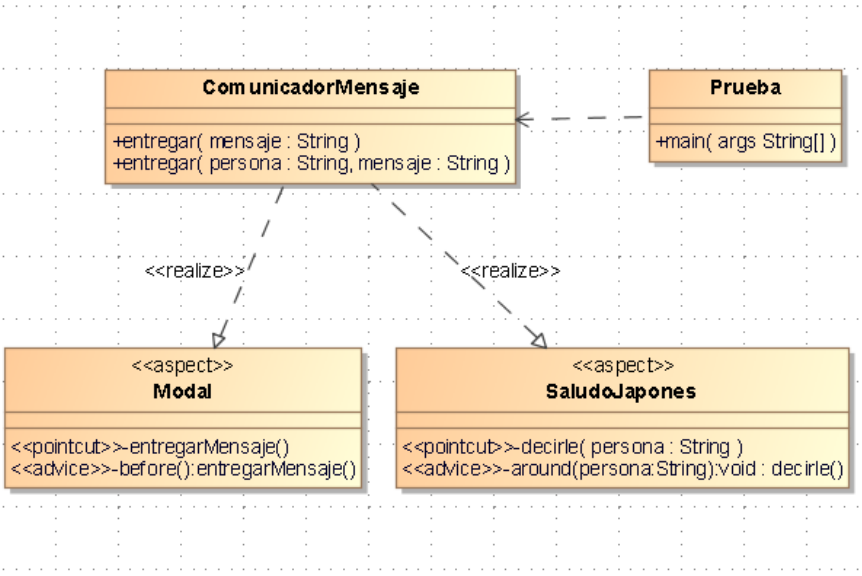

Figura 1. Ejemplo: Diseño Orientado Aspectos.

La Figura 2 muestra la definición de un perfil UML para el modelado de sistema OA, siendo sus componentes principales:

- `Aspect` $\Longrightarrow$ estereotipo que modela los aspectos como un *tipo particular* de clase. Un aspecto comprende un conjunto de características y un conjunto de puntos de corte.

- `PointCut`$\Longrightarrow$ estereotipo que modela el punto de corte; actúa como filtro y está definido por un conjunto de puntos de enlace.

- `Realize`$\Longrightarrow$ relación entre una clase y un aspecto. Indica que una clase usa uno o más aspectos.

- `WeavedClass`$\Longrightarrow$ es la unión de la clase y el/los aspectos que la afectan.

- `Advice`$\Longrightarrow$ código adicional que se ejecuta en un punto de corte, especifica una conducta que quiere ejecutar antes de (*before*), después de (*after*), o alrededor de (*around*) un punto de enlace [11].

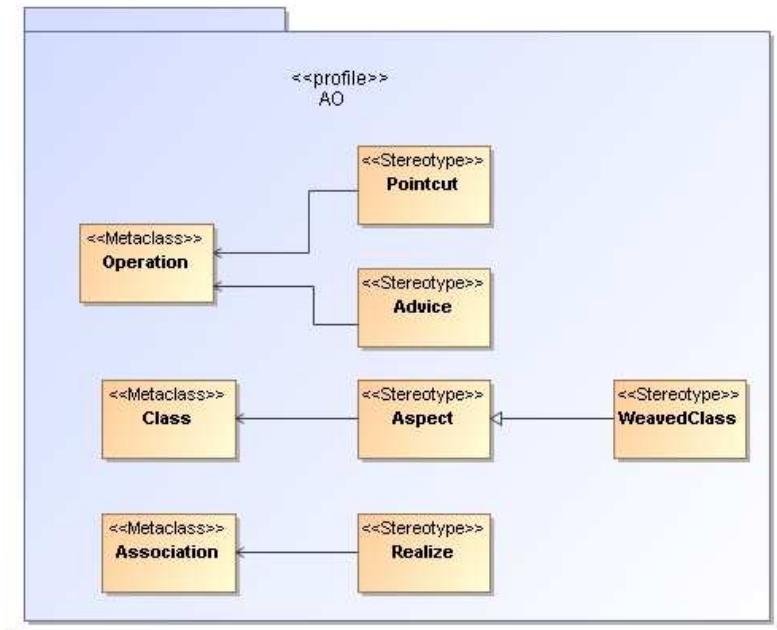

Figura 2. Definición de perfil.

4. Definición del Diseño O.A. según el metamodelo de la OMG

Este trabajo tiene como principal ventaja agilizar la tarea del ingeniero de software, ya que hoy por hoy no existe un estándar para un diseño OA ocasionando que cada ingeniero construya su propio diseño. Para evitar esto proponemos una definición del diseño OA según el metamodelo de la OMG.

A continuación el siguiente código muestra el metamodelo en XML que debe cumplir si desea poder utilizar una posible herramienta que genere el código Orientado a Aspectos.

El diseño OA mostrado en las secciones anteriores fue construido con la herramienta de modelado *MagicDraw*, la cual permite guardar el diagrama con formato XML. El siguiente paso fue construir consultas con el lenguaje XQuery [12], para filtrar y transformar el código que se generó con la herramienta de modelado, con el fin de poder construir el metamodelo en XML mencionado anteriormente.

Se ha decidido construir un metamodelo en XML con el objetivo de que ambas herramientas puedan interactuar (herrramienta de modelado — herramienta que permite la generación de código OA).

Metamodelo en XML

```
<diagram>
  <elements>
    <class name="ComunicadorMensaje"
         id="_12_5_1_24100552_1248559736171_533213_253">
      <operations>
        <operation name="entregar" visibility="public"
           id="_12_5_1_24100552_1248560172906_511909_332">
          <parameters>
            <paramenter name="mensaje" type ="String"/>
          </parameters>
        </operation>
        <operation name="entregar" visibility="public"
           id="_12_5_1_24100552_1248560578875_745998_336">
          <parameters>
            <paramenter name="persona" type ="String"/>
            <paramenter name="mensaje" type ="String"/>
          </parameters>
        </operation>
      </operations>
     </class>
    <class name="Prueba"
        id="_12_5_1_24100552_1248559741671_25693_273">
      <operations>
        <operation name="main" visibility="public"
           id="_12_5_1_24100552_1248560657046_568832_340">
          <parameters>
            <paramenter name="args" type ="String[]"/>
          </parameters>
        </operation>
      </operations>
     </class>
    <class name="Modal"
        id="_12_5_1_24100552_1248559744000_443735_293">
      <operations>
        <operation name="entregarMensaje" visibility="private"
           id="_12_5_1_24100552_1248642587500_34511_122">
        </operation>
        <operation name="before():entregarMensaje"
           visibility="private"
           id="_12_5_1_24100552_1248642887171_149824_141">
       </operation>
      </operations>
     </class>
    <class name="SaludoJapones"
```

```
        id="_12_5_1_24100552_1248642960515_797226_144">
      <operations>
        <operation name="decirle" visibility="private"
           id="_12_5_1_24100552_1248643179437_930981_164">
          <parameters>
            <paramenter name="persona" type ="String"/>
          </parameters>
        </operation>
        <operation name="around(persona:String):void : decirle"
           visibility="private"
           id="_12_5_1_24100552_1248643366281_3602_166">
        </operation>
      </operations>
    </class>
    <relations type="Dependency"
       id="_12_5_1_24100552_1248643920875_823152_357"
       supplier="_12_5_1_24100552_1248559736171_533213_253"
       client="_12_5_1_24100552_1248559741671_25693_273"/>
    <relations type="Realization"
        id="_12_5_1_601020a_1248882521671_526011_161"
        supplier="_12_5_1_24100552_1248559744000_443735_293"
        client="_12_5_1_24100552_1248559736171_533213_253"/>
    <relations type="Realization"
       id="_12_5_1_601020a_1248882552421_412678_178"
       supplier="_12_5_1_24100552_1248642960515_797226_144"
       client="_12_5_1_24100552_1248559736171_533213_253"/>
      <aspect>
        <class id="_12_5_1_24100552_1248642960515_797226_144"/>
        <class id="_12_5_1_24100552_1248559744000_443735_293"/>          .
      </aspect>
      <pointcut>
        <operation id="_12_5_1_24100552_1248643179437_930981_164"/>
        <operation id="_12_5_1_24100552_1248642587500_34511_122"/>
      </pointcut>
      <advice>
        <operation id="_12_5_1_24100552_1248643366281_3602_166"/>
        <operation id="_12_5_1_24100552_1248642887171_149824_141"/>
      </advice>
      <realize>
        <relation id="_12_5_1_601020a_1248882552421_412678_178"/>
        <relation id="_12_5_1_601020a_1248882521671_526011_161"/>
      </realize>
  </elements>
</diagram>
```

5. Conclusiones

Para lograr realizar la construcción del diseño OA se utilizó el mecanismo de extensión que provee UML: *el perfil*, en particular el uso de estereotipos. Como primera fase se construyó el diseño OA basado en un ejemplo particular. En una segunda fase se llevó a cabo consultas basadas en el lenguaje de consulta XQuery siendo su finalidad filtrar y transformar el diseño OA con formato XML a fin de lograr construir el metamodelo en XML, permitiendo facilitar la tarea del ingeniero de software obteniendo un concenso en el diseño OA, además de lograr una documentación mejorada. Como última fase de nuestro trabajo se realizó la construcción del metamodelo en XML para un diseño OA en particular.

Se ha logrado a través de este trabajo mostrar las ventajas de la utilización de herramientas estándar bajo las especificaciones de la OMG que permiten que el diseño sea guardado con formato XML.

Referencias

[1]Jacobson, I. and Booch, G. and Rumbaugh, J. El Lenguaje Unificado de Modelado — Addison-Wesley, 1999.

[2]XML, *http://www.w3.org/TR/1998/REC-xml-19980210.*

[3]IBM: XML (*Extensible Markup Language*) — Copyright International Business Machines Corporation 1998, 2001.

[4]XML: Carrasco, R.,The eXtensible Markup Language —Departamento de Lenguajes y Sistemas Informáticos. Universidad de Alicante, 2007.

[5]XQuery, *http://www.w3.org/TR/XQuery/.*

[6]Jong-Hyun Park, Ji-Hoon Kang, Internet and Web Applications and Services — http://ieeexplore.ieee.org/, 2007.

[7]Kiczales G., Lamping J., Mendhekar A., Maeda C., Videira Lopes C., Loingtier J-M, Irwin J., Aspect-Oriented Programming —Published in the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997.

[8]Quintero, A. R. Visión General de la Programación Orientada a Aspectos — Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla, 2000.

[9]UML 2.2 Infrastructure Specification —*http://www.omg.org/spec/UML/2.2/*, 2009.

[10]Debnath N.C., Garis, A., Riesco D., Montejano G. Defining Patterns Using UML Profiles — Computer Systems and Applications, 2006. IEEE International Conference on. March 8, 2006.

[11]Laddad, R. AspectJ in Action – Practical Aspect-Oriented Programming —Manning Publications Co., 2003.

[12]Walmsley P. XQuery —First Edition. Published by OReilly April 2007.