DEFINICIÓN DE MÉTRICA CON OCL PARA EL DISEÑO ORIENTADO A ASPECTOS USANDO PERFILES UML

Lorena Baigorria, Germán Montejano, Daniel Riesco
{flbaigor, gmonte, driesco}@unsl.edu.ar
Departamento de Informática, Universidad Nacional de San Luis
San Luis- CP 5700- Argentina

CACIC 2008- Ingeniería de Software

RESUMEN

La incesante búsqueda para mejorar el desarrollo de sistemas de software nos ha llevado a un nuevo paradigma; la Programación Orientación a Aspectos (POA) el cual ha surgido, basado en tecnologías existentes, como la orientación a objetos (OO) para mejorar la separación de las competencias en la programación de software.

Debido a la necesidad de calcular los recursos y calidad necesarios para el desarrollo de sistemas de software se han realizado gran cantidad de investigaciones acerca de distintas métricas de software orientadas a objetos y procedurales, pero no para software orientado a aspectos.

Las métricas de software son formas de calificar los diseños de software. Se puede decir entonces que las métricas aplicadas a sistemas orientados a aspectos son cruciales para determinar la efectividad de este paradigma como también de su uso en el diseño de sistemas de software.

Se propone en este trabajo la definición de distintas métricas aplicables al diseño orientado a aspectos. Para poder definirlas se ha realizado primero una extensión de UML 2.0 usando perfiles.

Las métricas son aplicables a modelos desarrollados con UML (Unified Modeling Language) y las restricciones semánticas de las mismas se especificarán con OCL (Object Constraint Language).

Palabras Claves: Métricas, Orientación a Aspectos, Perfiles UML, OCL, Métricas Orientadas a Aspectos.

1. INTRODUCCIÓN

Las métricas de software intentan medir la complejidad del software para predecir el costo total del proyecto y evaluar la calidad y efectividad del diseño. Las métricas de software tienen múltiples aplicaciones en las distintas tareas de la ingeniería de software tales como: pruebas, refactoring, gerenciamiento y mantenimiento [1].

La investigación de las métricas se ha ido adaptando a las necesidades, nuevas propuestas de desarrollo de software, nuevos lenguajes y nuevos paradigmas de programación, como es el caso de la POA.

La *programación orientada a aspectos* (*POA*) es una nueva metodología de programación que aspira a soportar la separación de competencias para los aspectos tanto en el diseño como en la implementación de software. Los lenguajes orientados a aspectos se utilizan para lograr esta separación y así obtener las ventajas proporcionadas por esta metodología como: el fácil desarrollo y mantenimiento del software.

El software Orientado a Aspectos (OA) es básicamente diferente del software desarrollado usando métodos convencionales. Lo más cercano a éste es el software Orientado a Objetos; por esto las métricas existentes para OO pueden ser un punto de referencia para la definición de métricas específicas para el software OA.

El software desarrollado con esta metodología es más fácil de desarrollar y mantener, debido a la separación de competencias entre las componentes y los aspectos del software.

Al aplicar métricas al diseño de sistemas OA podemos obtener una visión objetiva de la calidad del diseño.

Como con cualquier métrica, el principal objetivo de las métricas OA se derivan del software convencional: mejor comprensión de la calidad del producto, estimación de la efectividad del proceso y mejorar la calidad del trabajo realizado a nivel del proyecto.

2. ORIENTACION A ASPECTOS

Los aspectos son propiedades de un software que tienden a atravesar sus funcionalidades principales. Consideramos un *aspecto a una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación* [2].

La sincronización, el compartimiento y distribución de recursos son algunos ejemplos de aspectos. En desarrollos OO los aspectos están entrelazados en el corazón de los componentes del sistema causando el problema de tener código desordenado.

La Orientación a Aspectos trata de modularizar este entrecruzamiento de competencias y encapsularlos en módulos en vez de tenerlos dispersos en los componentes del sistema.

Para lograr programas OA utilizamos los lenguajes OA. Los lenguajes orientados a aspectos definen una nueva unidad de programación de software para encapsular las funcionalidades que cruzan todo el código. Además, estos lenguajes deben soportar la separación de aspectos como la sincronización, la distribución, el manejo de errores, la optimización de memoria, la gestión de seguridad, la persistencia. De todas formas, 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 tiene que interactuar de alguna manera. Para que ambos (aspectos y componentes) se puedan mezclar, deben tener algunos puntos comunes, que son los que se conocen como *puntos de enlace,* y debe haber algún modo de mezclarlos.

El encargado de realizar este proceso de mezcla se conoce como *tejedor.* El tejedor se encarga de mezclar los diferentes mecanismos de abstracción y composición que aparecen en los lenguajes de aspectos y componentes ayudándose de los puntos de enlace [2].

3. EXTENSION DE UML USANDO PERFILES

En la actualidad, UML (Unified Modeling Language) es uno de los lenguajes de modelado más utilizados para el diseño de sistemas [3]. Este lenguaje permite especificar, construir, visualizar y documentar artefactos de un sistema de software de manera sencilla.

Muchas veces UML es demasiado general y no es lo suficientemente expresivo para modelar elementos específicos de un dominio particular. En estas situaciones es necesario, extender UML.

Por esta razón, UML estándar incluyó un mecanismo para extender y adaptar UML a diferentes dominios y plataformas: el "Perfil UML" ("UML Profile")[4]

Para realizar la extensión UML 2.0 [5] incluyó la definición del paquete "Profile" (figura 1). Éste paquete incluye los mecanismos para redefinir estereotipos, valores etiquetados y restricciones.

La propuesta de extensión del perfil, no permite modificar al metamodelo existente, sino adaptar al metamodelo existente, agregando constructores propios para un dominio particular.

Los estereotipos son especificados como metaclases, los valores etiquetados como metaatributos y los perfiles como una clase de paquete.

El mecanismo de extensión del metamodelo impone restricciones en cómo el metamodelo UML puede ser modificado. No está permitido insertar nuevas metaclases, o modificar definiciones de la metaclase (agregando metaasociaciones).

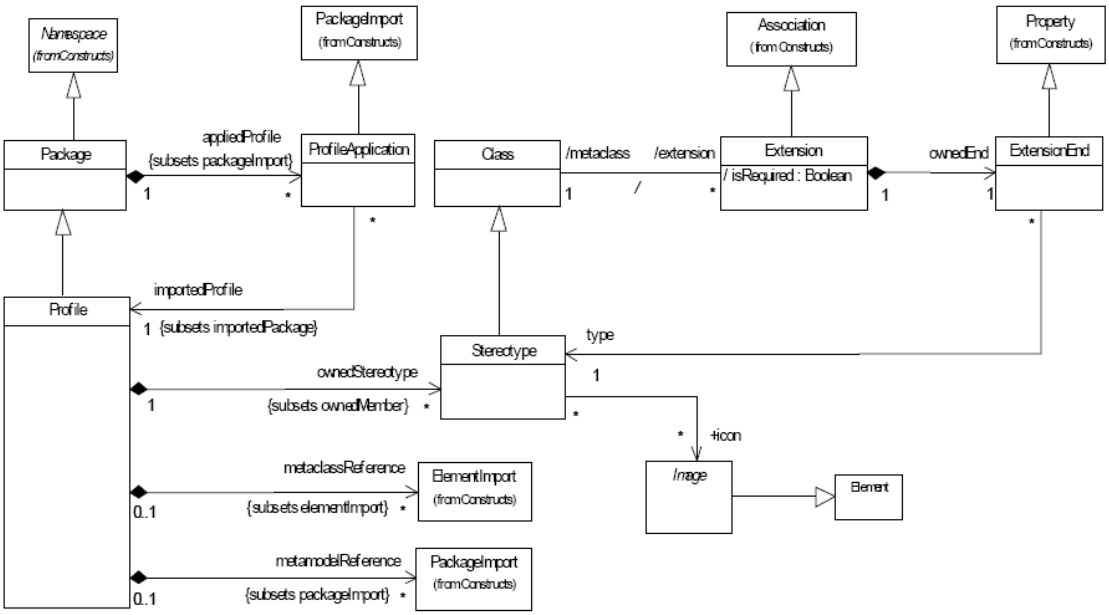

Figura 1: Paquete Profile de UML 2.0

3.1 Perfil para sistemas OA

En ésta sección se presenta la definición de un Perfil para modelar sistemas Orientados a Aspectos.

Un Perfil es un paquete estereotipado "package" que contiene los elementos que han sido extendidos para un dominio específico en este caso para Sistemas Orientados a Aspectos.

En la figura 2 se muestra el perfil para sistemas OA. El estereotipo 'Aspect' modela los aspectos de un sistema y es una extensión de la metaclase Class, el estereotipo 'Realize' modela la relación que existe entre una clase y un aspecto, donde el aspecto controla los componentes funcionales y es una extensión de la metaclase Association.

Estas extensiones indican que una clase puede ser un 'Aspect' y que una asociación puede ser una relación 'Realize'.

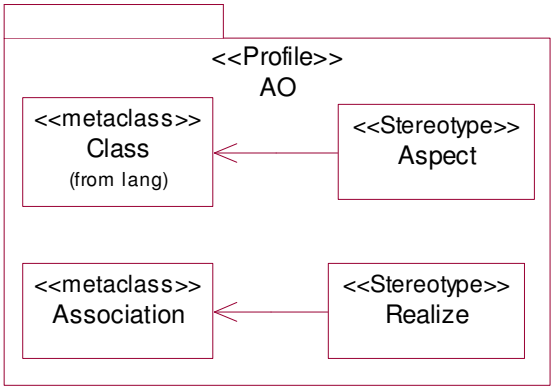

Figura 2: Perfila para sistemas OA

4. OCL (OBJECT CONSTRAINT LANGUAGE)

El Object Constraint Language es un lenguaje de expresión que permite describir operaciones y restricciones sobre modelos Orientados a Objetos (OO) y otros artefactos de modelado. [6]

OCL es un lenguaje tipado. Los tipos pueden ser cualquier clase de clasificadores dentro de UML.

Cabe destacar que no es un lenguaje de programación, por lo que no es posible escribir lógica de programas o flujos de control

El mismo OCL es utilizado para definir la semántica de UML, especificando reglas bien formadas sobre el metamodelo. Las reglas bien formadas son definidas como invariantes sobre las metaclases en la sintaxis abstracta. También es usado para definir operaciones adicionales, cuando las reglas bien formadas las precisan.

En las expresiones OCL, la palabra clave “context” introduce el contexto para una expresión: es decir desde qué metaclase se puede utilizar dicha operación/restricción y cual es el ámbito de la misma. Las palabras claves “inv”, “pre” y “post” denotan respectivamente a los estereotipos «invariant», «precondition», y «postcondition».[7]

En la práctica hay muchas reglas sobre una clase en un modelo, las cuales no pueden ser especificadas gráficamente. Llamamos a esas reglas restricciones. Una restricción es una expresión que da información adicional al modelo visual o artefacto.

Las restricciones se aplican a elementos de modelos o de sistemas OO y siempre restringen valores de esos elementos. Las restricciones deben ser verdaderas en un momento dado para que el modelo o el artefacto sea válido.

Si vemos el Perfil para sistemas OA, podemos notar que la extensión está basada en la metaclase Class, un aspecto (‘Aspect’) es una clase “particular”, es por esto que podemos aplicar restricciones a un aspecto.

5. METRICAS

La mayor parte del diseño OA es subjetivo, un diseñador “sabe” como caracterizar un sistema OA para implementar efectivamente los requerimientos del cliente. Pero se sabe que una vista más objetiva del diseño sería más beneficiosa. La definición de métricas da esta vista objetiva del diseño realizado [8]

Se pueden encontrar distintos trabajos acerca de métricas aplicadas a diferentes metodologías. En particular las métricas para sistemas OO es un campo muy estudiado. Trabajos como [9,10] definen formal o semi formalmente métricas aplicables a la OO.

En cuanto a métricas específicamente para sistemas OA hay muy poco realizado, algunos como el trabajo [11] han establecido que si se aplica la metodología de OA para el desarrollo de software se mejoran los valores obtenidos en las métricas para OO; es decir si se compara un sistema OA con un sistema OO usando métricas OO, se puede observar que la aplicación de la OA mejora el diseño del sistema.

Teniendo en cuenta las similitudes entre la OA y la OO, algunas métricas OO, como las C&K o las métricas MOOD, sirven de base para la definición de métricas para sistemas OA. Además al tener una extensión de UML a través de un perfil podemos definir métricas para sistemas OA utilizando OCL.

6. DEFINICION DE METRICAS AO

Las métricas que siguen a continuación son específicas para aspectos, es por esto que se establece como precondición, que la clase a la cual se le aplica la operación que especifica la métrica es un aspecto.

Para establecer esta condición se ha definido una operación isAspect(), la cual devuelve un valor booleano (true o false) indicando si la clase tiene un estereotipo de extensión 'Aspect'; es decir es un aspecto. En la figura 1 se puede observar el paquete Profile y la metaclase Class del mismo.

```
Context Class:: IsAspect():boolean

Pre:

Post: IsAspect=self.extension.ownedEnd->exist(e/e.type.name='Aspect')
```

En las siguientes subsecciones se mostrará la definición de algunas métricas, las cuales se utilizan en la etapa del diseño del software OA. Para realizar esta definición se toma el perfil OA definido y la especificación de la Infraestructura de UML 2.0.

En la figura 3 se muestra el metamodelo correspondiente a Class de UML 2.0, en el que se pueden observar las metaclases utilizadas para la definición de las métricas.

6.1 Cantidad de operaciones de un aspecto

Esta medida indica las operaciones de un aspecto sin tener en cuenta las operaciones heredadas.

```
Context Class::COA():Integer

Pre: self.isAspect()

Post: COA=self.ownedOperation->size()
```

6.2 Cantidad de atributos de un aspecto

Esta medida indica los atributos de un aspecto sin tener en cuenta los atributos heredados.

```
Context Class:CAA():Integer()

Pre:self.isAspect()

Post: CAA=self.ownedAttribute->size()
```

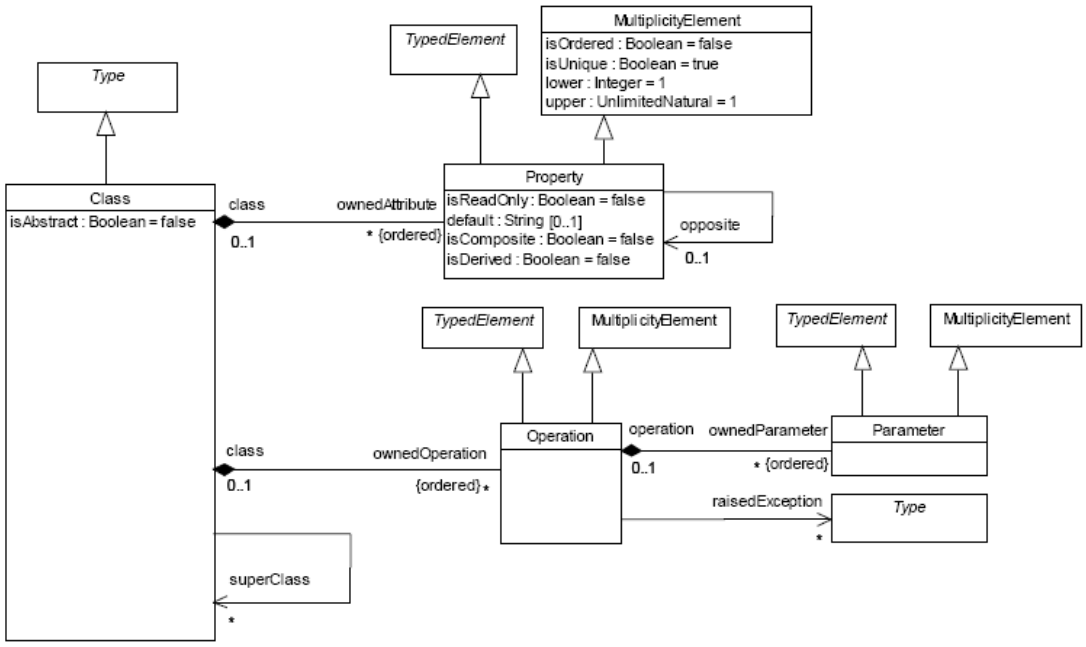

Figura 3: Metamodelo 2.0 Class

6.3 Cantidad total de operaciones de un aspecto

Esta medida indica las operaciones de un aspecto teniendo en cuenta las operaciones heredadas. Recordemos que una clase puede heredar de múltiples padres.

```
Context Class::CTO():Integer

Pre: self.isAspect()

Post: CTO=(self.COA()-> union (collect(self.superclass.CTO())))-> asSet-> sum()
```

6.4 Cantidad total de atributos de un aspecto

Esta medida indica los atributos de un aspecto teniendo en cuenta los heredados. Recordemos que una clase puede heredar de múltiples padres.

```
Context Class:CTA():Integer

Pre: self.isAspect()

Post: CTA= (self.CAA()-> union (collect(self.superclass.CTA())))->asSet->sum()
```

6.5 Tamaño de un aspecto

Teniendo en cuenta que el tamaño total de un aspecto depende de la cantidad total de atributos y operaciones que contiene, es directa la definición del tamaño de un aspecto.

```
Context Class:TA():Integer

Pre: self.isAspect()

Post: TA= self.CTA()->union (self.CTO)->sum()
```

6.6 Indicación de herencia múltiple

Esta medida indica si el aspecto hereda o no de múltiples superclases, y en caso de que sea así indica de cuantas.

```
Context Class:HM():Integer

Pre: self.isAspect()

Post: HM= self.superClass->size()
```

Considerando que un aspecto puede tener como superclase solamente a otro aspecto debemos agregar a la medida anterior la siguiente restricción.

```
Context Class

Inv: self.isAspect()implies self.superclass-> forall(c/c.isAspect() and c.SA())
```

Donde SA() indica recursivamente si las superclases son también aspectos

```
Context Class::SA():Boolean

Pre: self.isAspect()

Post: SA= if (self.superClass->notempty())
                then forall(c/c.isAspect() and c.SA())
                else true
          endif
```

7. CONCLUSIONES

La POA es una nueva metodología para el desarrollo de sistemas de software la cual permite un manejo más fácil del seguimiento y mantenimiento del software desarrollado. Como toda metodología debe ser evaluada a través de distintas herramientas para poder así calificar el uso de la misma. Una de las herramientas que permiten medir de manera más objetiva el diseño realizado mediante la aplicación de distintas técnicas, en particular de los sistemas OA, son las métricas. Esto nos lleva a profundizar el análisis de las métricas para poder aplicarlas al diseño OA y encontrar además de los posibles efectos, nuevas métricas que ayuden a calificar objetivamente un sistema OA.

Como se dijo anteriormente las métricas proporcionan una vista más objetiva del desarrollo que se realiza, esto es crucial en el momento de estimar recursos ya que no solo cumplir con los requisitos del cliente es importante sino también limitar y estimar los recursos necesarios. Por esto que el análisis de las métricas aplicadas a distintas etapas del desarrollo de software es un tema de investigación que debe ampliarse, en particular para la Orientación a Aspectos ya que su uso en desarrollos grandes aumenta crecientemente.

En este trabajo se presenta la definición de métricas específicas para sistemas OA, aplicables en la etapa del diseño ya que es ésta etapa la más costosa y en la cual surge el modelado de los aspectos del sistema. Para realizar la definición de las métricas se utilizo OCL, el cual como se mencionó anteriormente es un lenguaje no ambiguo y que permite expresiones lógicas.

Para poder realizar la definición de las métricas se realizó una extensión de UML 2.0 a través del uso de perfiles, lo que permite extender UML al dominio específico de la OA.

Es posible incluso validar las métricas obtenidas en herramientas existentes que permitan incorporar perfiles.

Es entonces la intención de este trabajo realizar éste análisis para ayudar a los desarrolladores de software que decidan incorporar la OA en su sistema de software.

REFERENCIAS

[1] Zhao, J. Measuring Coupling in Aspect-Oriented Systems. 10th International Software Metrics Symposium (Metrics 04), 2004. http://citeseer.ist.psu.edu/article/zhao04measuring.html

[2] Quintero A.; Visión General de la Programación Orientada a Aspectos. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. http://www.us.es/centrosdptos/departamentos/departamento_I0A0, último acceso octubre 2007

[3] UML 2.0 Specification Object Management Group (OMG) V2.0 www.omg.org, último acceso Julio 2008

[4] Fuentes L. Vallecillo A., Johson R., Vlissides J. "Una Introducción a los Perfiles UML" Novotica Vol. 168, pg. 6-11. 2004

[5] UML 2.0 Infraestructure Specification. http://www.omg.org/technology/documents. pg. 235-237

[6] Jos Warner; Anneke Kleppe; "The Object Constraint Language" Precise Modeling with UML; Eddison Wesley Publishing 1999

[7] OCL2.0 Specification, http://www.omg.org, julio del 2008.

[8] Lorenz, Mark and Kidd, Jeff, "Object Oriented Software Metrics", Ed. Prentice Hall, 1994

[9] Balasubramanian, N.V., IEEE "Object-Oriented Metrics" IEEE Transactions on Software Engineering, 1520-9321, Diciembre de 1999

[10] D. Riesco, G. Montejano, R. Uzal, et al, "A Technique Based on the OMG Metamodel: A Definition of Object Oriented Metrics applied to UML Models", The 3rd ACS/IEEE International Conference on Computer Systems and Applications January 3-6, 2006, Cairo, Egypt. www.ieee.org. IEEE Press. Pg. 118.

[11] Zakaria; Hosny; Metrics for Aspect-Oriented Software Design. The American University in Cairo. http://www.aucegypt.edu/ResearchatAUC/conferences/Pages/default.aspx, ultimo acceso Julio 2006.