# Un Modelo Formal de Patrones Orientados a Objetos

Alejandra Cechich
Departamento de Informática y Estadística
Universidad Nacional del Comahue
Buenos Aires 1400, (8300) Neuquén, Argentina
E-mail: acechich@uncoma.edu.ar

Richard Moore
International Institute for Software Technology
United Nations University
P.O. Box 3058, Macau
E-mail: rm@iist.unu.edu

PALABRAS CLAVES: Patterns ♦Frameworks ♦ Métodos Formales ♦ Orientación a Objetos ♦

## 1. Introducción

La mayoría de los métodos de desarrollo de software orientado a objetos imponen ciertos conceptos que son generalmente aceptados. Frameworks y patterns expresan ejemplos de buenas prácticas que pueden usarse para alcanzar resultados más efectivos [Bus96]. Pero en algunos casos particulares, es necesario una especificación formal que pueda verificarse a fin de producir sistemas más seguros.

Los patrones de Gamma [Gam95] (GoF patterns) desempeñan varios roles en el proceso de desarrollo orientado a objetos: proveen un vocabulario común para diseño; constituyen una base de experiencia para construir software reusable; y actúan como elementos básicos a partir de los cuales pueden construirse diseños más complejos. Una notación más formal que permita especificar en forma más segura, consistente y completa es todavía un desafío. Un primer trabajo en esa dirección [EdeA] representa patrones como formulas de LePus, un lenguaje definido como un fragmento de la lógica de primer orden [EdeB]. En [Mik98] se presenta un modelo abstracto de patrones, construido usando el método DisCo que se fundamenta en la lógica temporal.

En este resumen, presentamos nuestro modelo formal de patrones [Cec99B] basado en RSL (RAISE Specification Language) [Geo92], y el estado actual de su desarrollo. Futuras extensiones son abordadas al final.

## 2. Una Base Formal para Patrones Orientados a Objetos

Un patrón GoF se define en RSL como un tipo producto compuesto de tres elementos. La cabecera del patrón se conforma con el nombre y la categoría, por *ejemplo bridge, object-structural*; y es usado para identificar un patrón particular. Los otros dos elementos – estructura y colaboraciones – representan la *solución* descripta por un patrón.

GoF_Pattern = G. Pattern_Head X PS.WF_Pattern_Structure X CO.WF_Colls

**Estructura.** La estructura es un conjunto de clases bien formadas relacionadas por un conjunto de relaciones bien formadas. Las clases corresponden a nuestra declaración *Pattern_Class* y las relaciones corresponden a nuestra declaración *Pattern_Relationship* [Cec99A].

Pattern_Structure = C.Wf_Class-**set** X R.Wf_Relation-**set**

Las restricciones aplicadas sobre la estructura de un patrón pueden verse en [Cec99A].

**Colaboraciones.** El tipo *Collaboration* se compone de un par de objetos conectados y de un mensaje. Un objeto puede invocar a otro objeto enviando un mensaje del tipo *Signature_Head*. Cada objeto corresponde a una clase particular del tipo *Pattern_Class*, pero el tipo de relaciones entre clases es diferente de las colaboraciones, por ejemplo podemos relacionar dos clases por medio de una

agregación pero esas clases colaboran entre sí como un par emisor-receptor. Entonces, cada colaboración representa una invocación entre un par de objetos.

Ya que la información usada para instanciar un objeto es esencialmente la misma información contenida en las responsabilidades de los participantes del patrón, no necesitamos modelar las colaboraciones. Por ejemplo, en el patrón *Abstract Factory*, una clase "abstract factory" delega la creación de sus objetos "product" a sus subclases "concrete factory", y para crear diferentes objetos producto un cliente utiliza distintas fábricas concretas. Esas propiedades han sido modeladas usando las relaciones establecidas en la estructura y en las responsabilidades asociadas con un participante.

En el modelo, usamos identificadores como una forma de distinguir colaboraciones individuales. Para cada colaboración en el patrón hay un conjunto, posiblemente vacío, de colaboraciones que deben ejecutarse antes. En algunos patrones, cada colaboración tiene sólo un pre-requisito. Por ejemplo, en el patrón *Bridge* sólo *Operation()* debe ejecutarse antes de *OperationImp()*; en el patrón *observer* sólo *Notify()* debe ejecutarse antes de *Update()*; y en el patrón *State* sólo *Request()* debe ejecutarse antes de *Handle()*.

El registro compuesto Coll es usado para especificar la colaboración y sus pre-requisitos, y se utiliza una correspondencia entre identificadores y registros para modelar las colaboraciones del patrón.

Collaboration:: sender_o : G.Concrete_Object
receiver_o : G.Concrete_Object
signature_coll : P.Signature_Head,

Coll :: col : Collaboration prereq : G.Coll_Id-**set**,

Collaborations = G.Coll_Id —**m**→ Coll

Las colaboraciones deben satisfacer además condiciones de consistencia relacionadas con las clases y relaciones en el patrón. Primero, las clases deben pertenecer a la estructura del patrón y el mensaje debe estar definido en la interface del receptor; y segundo, no puede haber colaboraciones entre clases relacionadas por herencia.

Las restricciones aplicadas sobre las colaboraciones de un patrón pueden verse en [Cec99A].

## 3. Modelo de Colaboraciones Extendido

En el modelo presentado en la sección anterior, no ha sido contemplada la información semántica asociada a algunas partes de la representación de patrones. Por ejemplo, los comentarios agregados a la estructura de un patrón detallan la manera en que las clases colaboran para cumplir con su intención.

Para incluir esa información semántica es necesario modificar algunas declaraciones y/o extenderlas. Por ejemplo, la definición *Pattern_Class,* declara a la interface de esa clase como un tipo abstracto, sin embargo la interface representa el grupo de operaciones que esa clase puede atender, y cada operación se asocia a una signatura, al cuerpo de un método y al resultado que se produce por la aplicación de ese método.

Method :: meth_sig: Signature
meth_bod: Method_Body
meth_res: G.Vble

A su vez, cada signatura especifica el nombre de la operación y los objetos que toma como parámetros; el cuerpo del método especifica la lista de invocaciones que deberían realizarse y los cambios de variables producidos como resultados de esas invocaciones.

El cambio en las variables es modelado mediante una correspondencia entre la variable modificada y el objeto, junto con la signatura que ha producido el cambio; mientras que una invocación se modela por medio de una referencia al objeto receptor y a la signatura invocada en ese objeto:

```
Method_Body::     meth_vbles: G.Vble – m → G.Vble X Signature_Head
                  meth_invok: Meth_Call*

Meth_Call::       call_vbles: G.Vble*
                  call_sig: Signature_Head
```

Con este nuevo modelo, las colaboraciones se modelan implícitamente dentro del cuerpo de una clase y la definición de patrón cambia a:

GoF_Pattern = G. Pattern_Head X PS.WF_Pattern_Structure

Algunas propiedades definidas en [Cec99A] deben adaptarse al nuevo modelo y nuevas restricciones basadas en la semántica pueden ser agregadas. Por ejemplo, podemos modelar una propiedad general indicando que las variables modificadas deben estar incluidas en el estado de la clase de la siguiente manera:

```
changed_vble_in_state: Pattern_Class → Bool
changed_vble_in_state( c ) ≡   ( ∀ m: P.Method • m ∈ class_methods( c ) ⇒
                                   dom P.meth_vbles(P.meth_bod(m)) ∈ class_state( c
                               ) )
```

## 4. Futuras Extensiones

El modelo formal de patrones es un producto que se desprende del proyecto de investigación "Orientación a Objetos y Métodos Formales para la Especificación de Dominios".

El modelo ha sido extendido para incluir características semánticas implícitas en la estructura de un patrón. La formalización actual incluye sólo algunos patrones del catálogo de Gamma. Para poder eventualmente verificar que un diseño utiliza algún patrón, es necesario incluir mayor cantidad (y diversos tipos) de patrones formalmente. Por otra parte, variaciones de los patrones también deberían considerarse. Esas extensiones ya están siendo modeladas.

Pero para incluir mayor semántica en el modelo es necesario relacionarlo a un dominio. Por ello, se está analizando en forma paralela un dominio de alto riesgo (Unidades de Terapia Intensiva), en colaboración con clínicas y hospitales de la zona. La aplicación del modelo formal a este dominio generará un framework especificado en forma completa y consistente.

## Referencias


[Bus96] Buschmann F., Meunier R., Rohnert H., Sommerland P., and Stal M., *Pattern-Oriented Software Architecture*. John Wiley and Sons, 1996.

[Cec99A] Cechich A. and Moore R., *A Formal Specification of GoF Design Patterns*, UNU/IIST Technical Report 151, http://www.iist.unu.edu, January 1999

[Cec99B] Cechich A. and Moore R., *A Formal Basis for Object-Oriented Patterns*, APSEC'99 – 6th Asia-Pacific Software Engineering Conference, Takamatsu, Japón, Dec. 1999

[EdeA] Eden A., Gil J., Hirshfeld Y., and Yehudai A., *Towards a Mathematical Foundation for Design Patterns*, http://www.math.tau.ac.il/~eden/bibliography, 1998

[EdeB] Eden A., Hirshfeld Y., and Yehudai A., *LePUS – A Declarative Pattern Specification Language*, http://www.math.tau.ac.il/~eden/bibliography, 1998

[Gam95] Gamma, Helm, Johnson & Vlissides - *Design Patterns Elements of Reusable Object-Oriented Software* - Addison Wesley, 1995

[Geo92] George C., Haff P., Havelund K., Haxthausen A., Milne R., Nielsen C., Prehn S., and Wagner K. The RAISE Specification Language, Prentice Hall 1992.

[Mik98] Mikkonen T., *Formalizing Design Patterns*, 20th International Conference on Software Engineering Proceedings, 1998, pág. 115-124