Una Especificación Precisa para Patrones GoF

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 ♦ Métodos Formales ♦ Orientación a Objetos ♦

1. Introducción

Una especificación formal que permita definir en forma precisa la semántica de patrones de diseño, en particular patrones de Gamma o GoF [1], puede ser la base para definir una herramienta que ayude a desarrollar software más seguro. En ciertos dominios, certificar el uso de métodos y técnicas es indispensable para asegurar la calidad del software producido. Sin embargo, una notación más formal para patrones – que permita especificar en forma más segura, consistente y completa – es todavía un desafío. Nuestro primer trabajo en esa dirección y su extensión para incluir características semánticas implícitas en la estructura de un patrón [2][3], basados ambos en RSL (RAISE Specification Language) [4], fue originalmente comunicado en [5]. Esa formalización incluía sólo algunos patrones del catálogo de Gamma, aunque para verificar eventualmente que un diseño utiliza algún patrón es necesario incluir mayor cantidad (y diversos tipos) de patrones. Por otra parte, variaciones de los patrones también deberían considerarse.

En este resumen, presentamos una semántica precisa para patrones con la que finalmente ha sido modelado todo el catálogo de Gamma. Esa semántica ha servido también para iniciar la formalización de variaciones de esos patrones y de otros aplicables al análisis de dominios en lugar de al diseño de software. Futuras extensiones son abordadas al final.

2. Un modelo para la definición precisa de patrones.

Para definir en forma precisa la semántica que sustenta a los patrones de diseño orientados a objeto, se han definido tres tipos de acciones: invocación, instanciación e invocación a sí mismo o a una clase superior en la jerarquía de herencia. Una invocación representa una interacción que corresponde a una relación de asociación o agregación: objetos de una clase requieren a objetos de otra clase para cumplimentar alguna acción en respuesta a la ejecución de un método. Alguna variable (generalmente el "nombre" de la relación) en la primer clase representa al objeto que recibe el requerimiento, mientras que el éste consiste del nombre del método que debería ejecutarse y de un grupo de parámetros para ese método. En la especificación RSL, las variables han sido representadas por el tipo "Variable_name" y los parámetros del requerimiento por el tipo "Actual_Parameters", que es básicamente sólo una lista de variables. Luego, el requerimiento es modelado usando el tipo "Actual_Signature" y la invocación con el tipo "Invocation":

```
Invocation ::
      call_vble : Variable_Name  call_sig : Actual_Signature,
Actual_Signature ::
      meth_name : Method_Name  a_params : Actual_Parameters,
Actual_Parameters = Wf_Variable_Name*
```

Una instanciación por otra parte consiste del nombre de la clase a ser instanciada junto con una lista de parámetros:

Instantiation :: class_name : Class_Name a_params : Actual_Parameters

Las invocaciones a sí mismo o a una clase superior son análogas a las invocaciones comunes, excepto que se producen sobre la misma clase o sobre una superclase respectivamente. El tipo "Invocation" es usado para modelar esas situaciones, pero en estos casos la variable invocada es la variable específica "self" o "super" respectivamente:

self, super:: Variable_Name

De manera similar se han modelado los requerimientos y las asignaciones de variables que constituyen el cuerpo de un método concreto o implementado. El resultado de aplicar ese método, sus parámetros formales – una lista de parámetros cada uno modelado como una variable que no puede ser "self" o "super" del tipo "Wf_Vble_Name" – con opcionalmente el nombre de una clase indicando el tipo de la variable y también su nombre, han sido modelados como:

Method :: f_params : Wf_Formal_Parameters
  meth_res : Result body : Method_Body,
Result = Variables,
Parameter ==
  var(Wf_Vble_Name) | paramTyped(paramName : Wf_Vble_Name, className : Class_Name)

El estado de una clase se define en forma análoga como un conjunto de variables. Luego, se especifican varias condiciones de consistencia sobre esta estructura y el modelo se aplica al catálogo completo de patrones de Gamma. Detalles de esas especificaciones pueden verse en [6][7][8] y [9].

3. Otras aplicaciones y/o ampliaciones del modelo formal.

Entre otras ampliaciones, el modelo formal de patrones ha sido extendido para incluir otras variaciones. Por ejemplo, un diseño compuesto por varios patrones relacionados posee propiedades adicionales derivadas de la interacción. Por ello, formalizar la composición de patrones – definida como la aplicación conjunta de patrones donde una clase puede poseer diversos roles, cada uno asociado a un patrón diferente -– es una tarea aún más compleja. Nuestro primer trabajo en esa dirección puede verse en [10].

La generalidad de nuestra especificación formal favorece su aplicación a otro tipo de patrones. Por ejemplo, los patrones de análisis de Fowler [11] modelan un dominio completamente distinto, pero la descripción de su estructura y comportamiento siguen lineamientos similares a los de los demás patrones. Así, reusando las especificaciones formales un patrón de análisis se define como:

Analysis_Pattern = G. Pattern_Name x PS.WF_Pattern_Structure

Algunas de sus propiedades han debido definirse especialmente, es decir no han sido reusadas desde el modelo original. Por ejemplo, cada relación de herencia se clasifica ahora como *completa* o *incompleta* y se modela con una definición variante; la noción de instanciación de una estructura usada para especificar restricciones semánticas se define como el tipo "Instances"; etc.

Inheritance_type = = complete | no_complete
inheritance_type: R.Wf_Relation → G.Inheritance_type

Instances :: Instance-**set**
Instance ::
  fixed_object : G.Concrete_Object
  connected_objects : G.Concrete_Object-**list**

Las diversas restricciones de consistencia aplicadas sobre estos tipos, así como la formalización de propiedades semánticas propias de los patrones de análisis, pueden verse en [12]. Por ejemplo, el

operador semántico que restringe a los objetos conectados por una asociación a formar un grafo acícilico dirigido (dag), se modela como una función en la que toda instanciación de una clase particular de la estructura – conectada por una relación recursiva – cumple con la restricción mencionada.

dag_type: PR.Role_Type x PR.Signature_Type → PS.Wf_Pattern_Structure → **Bool**
dag_type(ro, rid)(psc, psr) ≡
∀ cl: C.Wf_Pattern_Class, re: R.Wf_Relation, co: Wf_Instances • cl ∈ psc ∧ re ∈ psr ∧
is_instantiation(co, psr) ∧ C.role_type(C.p_role(cl) = ro ∧ R.signature_type(re) = rid ∧
R.relation_type (re) = G. association ∧ exists_loop(re, psr) ⇒ is_dag(co)

La formalización de otras propiedades asociadas al comportamiento de los patrones de análisis también han sido direccionadas. Algunas de esas especificaciones pueden verse en [13].

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".

La especificación completa del catálogo de Gamma permite iniciar ahora la construcción de una herramienta de software que verifique en forma semi-automática diseños realizados con patrones. Así, cualquier desarrollo podría certificar que el uso dado a los patrones no contradice su semántica, de acuerdo a como ha sido formalmente definida.

La construcción de esta herramienta es parte de las actividades actuales de nuestro proyecto, así como la aplicación del modelo formal a dominios específicos.

Referencias

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

[2] Cechich A. and Moore R., *A Formal Basis for Object-Oriented Patterns* , APSEC'99 – 6th Asia-Pacific Software Engineering Conference, Takamatsu, Japón, Diciembre 1999, 284-291

[3] Cechich A. and Moore R., *A Formal Model for Behavioural Properties of Object-Oriented Patterns*, CLEI 2000 – XXVI Conferencia Lationamericana en Informática, México D.F., 18-22 Septiembre 2000

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

[5] Cechich A., Moore R., *Un Modelo Formal de Patrones Orientados a Objetos*, Workshop de Investigadores en Ciencias de la Computación, WICC 2000, La Plata, Mayo 2000, 86-89.

[6] Reynoso L. and Moore R., *GoF Behavioural Patterns: A Formal Specification*, UNU/IIST Technical Report 201, May 2000

[7] Aranda G. and Moore R., *GoF Creational Patterns: A Formal Specification*, UNU/IIST Technical Report 224, December 2000

[8] Flores A. and Moore R., *Analysis and Specification of GoF Structural Patterns*, 19th IASTED (International Association of Science and Technology for Development) – International Conference: Applied Informatics (AI2001), Innsbruck, Austria, Febrero 19-22, 2001

[9] Flores A., Moore R., Reynoso L., *A Formal Model of Object-Oriented Design and GoF Design Patterns*, Formal Methods Europe, FME 2001, Berlin, Alemania, 12-16 Marzo 2001

[10] Aranda G. and Moore R., *Formally Modelling Compound Design Patterns*, UNU/IIST Technical Report 225, December 2000

[11] Fowler M., *Analysis Patterns – Reusable Object Models*, Addison-Wesley, 1997

[12] Cechich A., *A Formal Model for Semantic Statements of Analysis Patterns*, IV Workshop Iberoamericano de Ingeniería DE Requisitos y Ambientes de Software, San José, Costa Rica, 3-6 Abril, 2001

[13] Buccella A. and Cechich A., *A Formal Model for Some Behavioural Features of Analysis Patterns*, 6° Congreso Argentino en Ciencias de la Computación, CACIC'2000, Usuhaia, 2-7 Octubre 2000, 123-132, ISBN 950-763-033-3