Vers la Formalisation de la Composition de Patrons de Conception

Halima Douibi
Laboratoire LIRE
Université Mentouri
Constantine - Algérie
douibi_halima@yahoo.fr

Faiza Belala
Laboratoire LIRE
Université Mentouri
Constantine - Algérie
belalafaiza@hotmail.com

Kamel Boukhelfa
Laboratoire LIRE
Université Mentouri
Constantine - Algérie
boukhelfakamel@yahoo.fr

Résumé

Nous présentons dans ce papier une contribution à la spécification formelle des patrons de conception. Les deux aspects structurels et comportementaux d'un patron sont définis au moyen des théories orientées objets de la logique de réécriture. A la différence des autres approches de formalisation, notre méthode se contente d'un seul cadre mathématique basé logique de réécriture qui supporte à la fois le raisonnement rigoureux sur ces patrons et la formalisation des relations qui peuvent les unir.

1 Introduction

Les recherches sur les patrons de conception s'inscrivent dans le cadre des approches tentant de décrire des solutions standards pour répondre à des problèmes récurrents d'architecture et de conception des logiciels. La description actuelle des patrons [1] ne permet ni leur spécification formelle, ni la mécanisation de leur application, ni leur combinaison.

Or, la description formelle de la structure des patrons ainsi que leurs composition permet la détection de problèmes tôt dans le cycle de vie du logiciel.

De nombreux travaux se sont penchés sur la problématique de la spécification formelle des patrons de conception. Des approches à base de diagrammes (UML, méta-modèle, ou autres), [4], [2], [3] ont proposé des outils ou modifié des outils existants pour représenter ou mettre en application des patrons. Cependant, les représentations adoptées sont ambiguës et dépendent généralement des objectifs de chaque approche. Peu de travaux s'attaquent au problème de la composition des patrons. Nous citons en particulier la contribution de Mikkonen [7] pour composer deux patrons à base d'un langage appelé *Disco*. Chaque patron est formalisé par une couche du langage *DisCo* et la composition des patrons est vue comme un raffinement sur les couches de spécifications. Dans cette approche l'aspect structurel des patrons est négligé. Et c'est aussi le cas du travail de Saeki [8] qui a utilisé le langage *LOTOS* pour spécifier le comportement et la composition de deux exemples de patrons. Par ailleurs, Taibi et Ngo [10] proposent initialement une spécification formelle de l'aspect structurel des patrons de conception à base de logique du premier ordre (*FOL*) et l'aspect comportemental dans la logique temporelle de l'action (*TLA*), ensuite l'un des auteurs dans [9] procède à la formalisation de la composition des patrons sans se préoccuper des aspects de vérification et d'analyse. Nous suggérons dans ce travail un support mathématique unique pour définir les aspects structuraux et comportementaux des patrons de conception, tout en exploitant l'expressivité et la puissance de *la logique de réécriture* via son langage *Maude*[5] [6]. L'objectif de ce travail est de proposer d'une part, une approche de modélisation (basée méta-modèle) des patrons de conception dotée d'un fondement mathématique approprié et d'autre part, de pouvoir raisonner correctement sur leur composition.

Dans la section 2, nous détaillons notre approche de formalisation des patrons de conception à base de logique de réécriture. Nous commençons tout d'abord par proposer une modélisation générique et extensible des patrons de conception que nous dotons d'un modèle formel basé sur des théories de réécriture orientées objet. Dans la section 3, nous montrons comment exploiter ce modèle pour permettre la spécification de l'opération de composition des patrons de conception et les possibilités de son analyse. Finalement, la conclusion propose une synthèse du travail réalisé et des perspectives liées à la poursuite de ce travail.

2 Les Patrons de conception et leur Méta-modèle dans Maude

En se basant sur les méthodes de description des patrons de conception à l'aide des méta diagrammes et celles fondées sur des formalismes purement mathématiques, nous proposons dans cet article un nouveau modèle qui combine les avantages de ces deux types de méthodes.En effet, l'existence d'une variété de patrons de conception justifie la définition d'un méta-modèle unificateur, qui permet de décrire toutes les caractéristiques des patrons de conception dans un même niveau d'abstraction (figure 1).

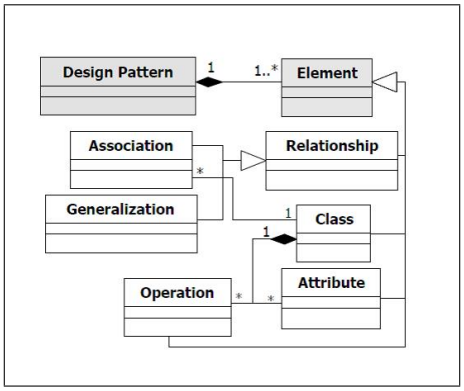

FIGURE 1 – Un méta-modèle simplifié des patrons

Nous considérons la partie solution d'un patron de conception comme une collection d'éléments. Dans le contexte de ce travail et pour des raisons de simplicité, nous ne considérons pas les contraintes `OCL` (Object Constraint Language). Le méta-modèle proposé présente une solution générique qui permet de spécifier n'importe quel patron et peut être facilement étendu pour prendre en charge d'autres éléments d'un patron tels que les contraintes ou encore l'aspect dynamique des patrons. Dans le méta-modèle des patrons (figure 1), la classe `Design Pattern` sert à générer les instances de patrons. Les différents types d'éléments sont liés par des associations spécifiques.

Le méta-modèle est implémenté en Maude en tant qu'un multi-ensemble d'objets et de messages juxtaposés, tandis que les interactions concurrentes entre objets sont gouvernées par les règles de réécriture. Les objets représentent les éléments du modèle qui peuvent avoir un certain comportement. Ainsi, la sémantique des interactions de ces objets est concrétisée par la définition de messages. Cette formalisation attribue aux aspects structuraux et comportementaux des patrons un modèle sémantique précis à base de catégories. De plus, la spécification constitue un programme Maude exécutable. Nous définissons la classe `Design Pattern` dans

notre module Maude orienté-objet global `Meta-model-DP` comme une classe Maude nommée `Meta-DP` avec un attribut spécifique `Name-ID` de type `QID`. `Element` dans le méta-modèle est déclaré comme une classe appelée `D-element` ayant un attribut `Meta-DP` liant tous les éléments du méta-modèle.

```
omod Meta-model-DP is
Including QID. Class Meta-DP |NameID : QID .
Class D-element | Meta-Dp: oid .
...
Endom
```

Nous définissons les éléments de type `Class`, `operation` et `attribute` comme des sous-classes (*subclass*) de la classe `D-element`.

```
Subclass D-class < D-element
Class D-operation | Cl : oid. Class D-attribute | Cl : oid
Subclasses D-operation D-attribute < D-element
```

`Relationship` représente une classe abstraite pour les différents types d'associations. Ainsi, elle peut être une association simple entre deux classes ou bien une relation de généralisation. Nous définissons `Relationship` comme une classe `D-relationship`. Pour `Association` et `Generalization`, nous utilisons respectivement les classes `D-association` et `D-generalization`, qui sont des sous-classes de la classe `D-relationship`. Des attributs sont rajoutés afin d'identifier pour une associations simple la classe à chacune de ses extrémités et pour la généralisation sa classe parente.

```
Class D-relationship < D-element .
Class D-association | Ass-End1 :
oid , Ass- End2 : oid.
Class D-generalization | Cl-parent :
oid , Cl-child : oid.

Subclasses D-generalization D-association
<D-relationship .
```

Afin d'explorer la structure du méta-modèle et vérifier l'existence de ses composants, nous proposons une stratégie de réécriture de messages d'objets :

```
msg_get-classes : Oid -> Msg .
msg_get-operations_ : Oid Oid -> Msg.
msg_get-attributes_ : Oid Oid -> Msg .
msg_get-associations_ :Oid Oid -> Msg .
msg_get-generalization_ :Oid Oid -> Msg .
```

```
rl[get-classes]:
< Mod : Meta-DP | NameID : M1 >
< Obj : Dclass | Meta-Dp : Mod >
Mod get-classes =>
< Obj : D-class | Meta-Dp : Mod>...
```

Le module Maude `meta-model-DP` implémentant ce méta-modèle est suffisamment générique pour qu'il puisse aisément être enrichi afin de prendre en considération tous les patrons *GOF*. Il suffit de l'insérer dans le méta niveau de Maude (module prédéfini *META-LEVEL*), puis d'ajouter un certain nombre de constantes (opérateurs ou équations) qui vont contrôler le mécanisme d'instanciation du méta-modèle et par conséquent, générer la structure du patron en question. Quelques exemples de patrons classiques (singleton, mediator, observer, etc) ont été déjà implémentés et testés selon cette approche.

3 Sémantique de la composition des patrons

Les relations qui peuvent exister entre deux patrons de conception donnés peuvent s'exprimer dans notre approche par une opération qui lie les deux termes Maude de sorte *module* définissant ces patrons. La sémantique de cette opération si elle est d'ordre statique est donnée au moyen d'équations dans le module `ML-Meta-model-DP`. Dans le cas où la relation est dynamique, sa sémantique est définie au moyen d'un ensemble de règles de réécriture pour composer deux patrons sous forme de recouvrement(partager un ensemble d'éléments)[11]. Dans le context de ce travail, nous définissons la composition de deux patrons au méta niveau de Maude par l'opération de *sommation* des deux modules Maude correspondants tout en appliquant l'opération de renommage pour identifier les éléments communs aux deux patrons. La sommation de deux modules Maude `A` et `B` permet de définir un nouveau module `C` (noté : $A + B$) combinant les informations contenues dans les deux sous-modules `A` et `B`. c-à-d :

$$\Sigma_C = \Sigma_A \cup \Sigma_B \ / \ E_C = E_A \cup E_B \ / \ R_C = R_A \cup R_B$$

Elle permet d'importer les éléments contenus dans les deux modules `A` et `B` dans un nouveau module `C`. Si A et B présentent des éléments en communs, on peut procéder à leur renommage au moyen de l'opération *renaming* offerte par Maude. Les éléments concernés par cette opération peuvent être des identificateurs d'opération, de sorte, ou même des étiquettes de règles.
Soient `MD1` le module Maude spécifiant un patron `D1` et `MD2` le module Maude spécifiant un patron `D2`. Le module expression est noté :

```
"MD1" * ( op ME_{1} to E_{1}',...) + "MD2" * ( op ME_{2} to E_{2}',...)
```

Il permet de définir le patron composé des deux patrons `D1` et `D2`, l'opération de *renommage* permet de donner un même nom aux les éléments en communs aux deux patrons, et l'opération *somme* permet de réunir les élément de même nom dans un seul élément, ce dernier contenant toutes les informations des différents éléments réunis. Nous illustrons notre formalisation par l'exemple de composition des deux patrons *Mediator* et *Observer*. dont le code simplifié est donné par la figure 2.

La composition dans cet exemple consiste en un recouvrement : 1) De la classe `concrete-subject` du patron *Observer* et de la classe `concrete-colleague1` du patron *Mediator* d'une part ;2)De la classe `concrete-observer` du patron *Observer* et de la classe `concrete-colleague2` du patron *Mediator* d'autre part. Les deux premières classes sont remplacées par la classe `subject-colleague`. Les deux autres sont par contre remplacées par la classe `observer-colleague` (figure 3). La composition des patrons *Observer* et *Mediator* permet aux sujets et aux observateurs de communiquer soit directement (patron *Observer*), soit à travers un médiateur qui fournit une boîte aux lettres pour la communication (patron *Mediator*).

Pour exprimer la composition de ces deux patrons dans Maude, nous réutilisons leurs spécifications en important le module `ML-Meta-model-DP` :

```
( mod compose-DP is
protecting  ML-Meta-model-DP .
op observer-mediator : -> Module .
...
endm )
```

```
(omod 'observer-DP is                          (omod 'Mediator-DP is
 op Observer-DP : -> Module                     subsort 'MediatorDP < 'configuration
 eq Observer-DP =                               op 'Mediat: -> 'Meta-DP
(omod 'Observer-DP is                           op 'mailbox-class : -> 'D-class
subsort 'ObserverDP < 'configuration            op 'concrete-mediator-class : -> 'D-class
 op 'Observ : -> 'Meta-DP                       op 'colleague-class : -> 'D-class
op 'concrete-subject-class : -> 'D-class        op 'concrete-colleague1-class : -> 'D-class
 op 'concrete-observer-class : -> 'D-class      op 'concrete-colleague2-class : -> 'D-class
... endom )                                     ... endom )
```

FIGURE 2 – Le code des patrons "Observer" et "Mediator"

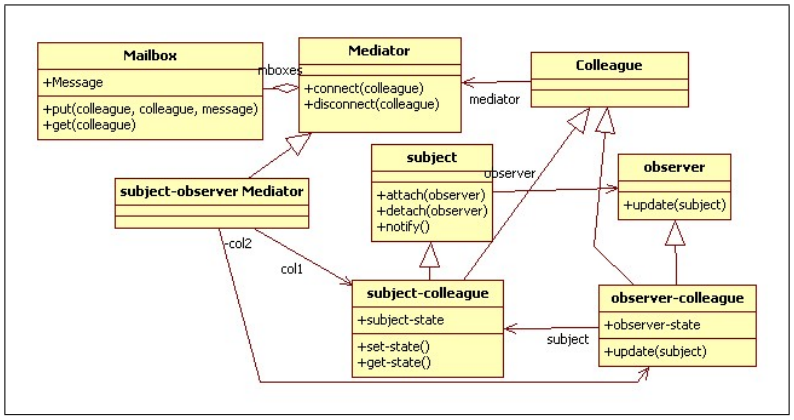

FIGURE 3 – Structure de la composition des patrons " Observateur-Médiateur "

Le résultat de la composition des deux patrons est un module Maude appelé `observer-mediator` défini par l'opération de sommation ci-dessous. Nous appliquons l'opération de *renommage* pour les classes concernées par le recouvrement :

```
(mod observer-mediator is
...
pr observer-DP *
( op 'concrete-subject-class :
-> 'D-class to
'Subject-Colleague_,
op 'concrete-observer-class :
-> 'D-class
to 'observer-Colleague_)
+
 pr mediator-DP *
 ( op 'concrete-colleague1-class : -> 'D-class
 to 'Subject-Colleague_,
 op 'concrete-colleague2-class : -> 'D-class
 to 'observer-Colleague_,op
 'concrete-mediator-class: -> 'D-class
 to subject-observer-Mediator_  ).
endom )
. endm )
```

Dans le module représentant la composition, nous retrouvons tout ce qui est défini dans les

deux modules initiaux, sauf les classes renommées par le même nom, elles sont regroupées dans une seule classe tout en conservant les informations des deux premières. Nous effectuons aussi un renommage de la classe `concrete-Mediator` (*Mediator*), dont le nom devient `subject-observer-Mediator`.

4 Conclusion

Le manque de description formelle des patrons rend leur utilisation et leur vérification difficiles. En effet, le catalogue de Gamma et al. ne peut en aucun cas être exhaustif, et les liens entre patrons seront de plus en plus complexes avec la découverte de nouveaux patrons, ce qui rendra l'identification et la composition des patrons encore plus difficiles. Nous avons contribué dans ce papier à la spécification formelle des patrons de conception en exploitant leur méta modélisation constituée souvent de diagrammes de classes UML. Nous avons défini les deux aspects structurel et comportemental d'un patron au moyen des théories *orientées objets* de la logique de réécriture. Le caractère de réflectivité (" *reflection* ") de cette logique a permis d'implémenter la spécification proposée dans le méta niveau de Maude, et d'obtenir ainsi un modèle exécutable. De plus, nous avons pu tirer profit de cette approche de formalisation pour définir la sémantique des opérations qui peuvent unir deux patrons de conception. Nous nous sommes intéressés particulièrement à l'opération de composition. Nous projetons d'étendre cette étude à la définition d'autres types d'opérations sur les patrons et le raisonnement sur les modèles résultants.

Références

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns : elements of reusable object-toriented systems*. Addison-Wesley, 1995.

[2] Epameinondas Gasparis, Jonathan Nicholson, and Amnon H. Eden. Lepus3 : An object-oriented design description language. In *Diagrams*, pages 364–367, 2008.

[3] Dae kyoo Kim, Robert France, Sudipto Ghosh, and Eunjee Song. A uml-based metamodeling language to specify design patterns. In *Patterns, Proc. Workshop Software Model Eng. (WiSME) with Unified Modeling Language Conf. 2003*, 2003.

[4] David Mapelsden, John Hosking, and John Grundy. Design pattern modelling and instantiation using dpml. In *CRPIT '02 : Proceedings of the Fortieth International Conference on Tools Pacific*, pages 3–11, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[5] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. *Electr. Notes Theor. Comput. Sci.*, 4 :190–225, 1996.

[6] José Meseguer. A logical theory of concurrent objects and its realization in the maude language. In *Research Directions in Object.MIT Press*, 1992.

[7] Tommi Mikkonen. Formalizing design patterns. In *ICSE*, pages 115–124, 1998.

[8] Motoshi Saeki. Behavioral specification of gof design patterns with lotos. In *APSEC*, pages 408–415, 2000.

[9] Toufik Taibi. Formal specification of design patterns' relationships. In *ACST*, pages 310–315, 2006.

[10] Toufik Taibi and David Ngo Chek Ling. Formal specification of design patterns - a balanced approach. *Journal of Object Technology*, 2(4) :127–140, 2003.

[11] Walter Zimmer. Relationships between design patterns. In *Pattern Languages of Program Design*, chapter 18, pages 345–364. Addison-Wesley, 1995.