Une sémantique opérationnelle pour une meilleure méta-modélisation

Ali Koudri* † — **Joël Champeau**† — **Denis Aulagnier***

** Thales Systèmes Aéroportés*
10 avenue de la 1ère DFL - CS 93801 - 29238 Brest cedex 3
{ali.koudri,denis.aulagnier}@fr.thalesgroup.com
† ENSIETA
2 rue François Verny
29806 Brest Cedex 9
joel.champeau@ensieta.fr

RÉSUMÉ. *Dans les domaines contraints par les normes de certification, l'utilisation de méta-modèles semi-formels ne peut pas être envisagée car elle rendrait difficile un certain nombre d'opérations (preuves, consistance, transformations, etc). Nous nous attachons dans cet article à soulever les problèmes conceptuels et techniques dans la formalisation des méta-modèles de l'infrastructure d'UML2 et à proposer une implantation en KerMeta qui améliore cette situation.*

ABSTRACT. *The use of semi-formal meta-models is in some cases, such as fields constrained by certification norms, forbidden. The reason is that models would then not provide the ability to make useful operations such as proof, consistency checking, transformations, etc. In this paper, we have identified, through an implementation of UML2 infrastructure in KerMeta, conceptual and technical problems.*

MOTS-CLÉS : *IDM, méta-modèle, formalisation, KerMeta*

KEYWORDS: *IDM, meta-model, formalization, KerMeta*

1. Introduction

Dans les domaines embarqués critiques tels que l'avionique, le développement d'applications complexes est un processus soumis à des contraintes de certification imposant un cadre strict. Ces développements sont menés via un ensemble d'activités, parallèles ou non, gérées par différentes équipes ayant leur propre domaine d'expertise et utilisant une terminologie propre à leur métier qui conduit généralement à l'utilisation de DSL (Domain Specific Language). Dans l'état actuel des technologies, il existe deux possibilités pour définir des DSLs basés sur des méta-modèles : par l'utilisation de "DSL factories"[1], ou par la création de profiles UML quand le méta-modèle est projeté sur le celui d'UML.

Le MOF(Meta Object Facility) (OMG, 2006a) permet une unification de l'expression des méta-modèles, qu'ils soient ensuite utilisés comme profils UML ou non. Dans cet article, nous considérons l'espace conceptuel, représenté par la spécification du MOF, et l'espace technique ou domaine du réalisable, qui semble être majoritairement vu à travers ECore de l'environnement Eclipse[2]. Prenons par exemple le cas du méta-modèle UML2(OMG, 2006b) : en considérant ce méta-modèle comme complètement formalisé via sa spécification, ce qui peut prêter à discussion (Kobryn, 1999, Alanen *et al.,* 2006, Diskin *et al.,* 2005), l'espace technique choisi doit conserver toute la sémantique décrite par ce méta-modèle, sans introduire de distorsions ou de lacunes.

Lié au cycle de développement, le méta-modèle doit nous aider à garantir la conformité des modèles à travers toutes les étapes du processus et tous les points de vue présents dans un modèle. La garantie de cette conformité est généralement assurée par des artefacts extérieurs au méta-modèle (outils), bien que la définition même de celui-ci possède de nombreuses propriétés garantissant une bonne cohérence des points de vue issus du méta-modèle.

Dans le même ordre d'idée, l'interopérabilité entre les outils UML reste un problème clairement identifié par les utilisateurs. En effet, le faible niveau de formalisation du méta-modèle UML, comme nous le verrons dans les paragraphes suivants, conduit à une interopérabilité somme toute relative, pour ne pas dire inexistante. Là encore, cette situation laisse la part à l'interprétation des outilleurs qui sont obligés d'implanter une partie de la sémantique du langage dans l'outil plutôt que de s'appuyer sur un méta-modèle formalisé.

Dans la suite de cet article, nous présentons un travail montrant qu'actuellement il est difficile d'exprimer techniquement la sémantique d'un méta-modèle (mapping entre le monde conceptuel et le monde technique) à l'aide des plateformes couramment utilisées comme EMF (Eclipse Modeling Framework). À travers cette étude, nous faisons ressortir les manques au niveau conceptuel et technique pour la bonne

---

1. microsoft, http ://www.microsoft.com
topcased, http ://www.topcased.org
GMF, http ://www.eclipse.org/gmf
2. EMF, http ://www.eclipse.org/emf

expression des méta-modèles. Nous avons choisi, pour illustrer notre propos, une implantation de l'infrastructure d'UML2, constituant le coeur d'un bon nombre de méta-modèles.

2. Motivations

Comme nous l'avons souligné en introduction, notre domaine d'application est fortement contraint par les besoins de certification où l'utilisation de méta-modèles semi-formels n'est pas envisageable.

Pour décrire des méta-modèles, nous disposons au niveau conceptuel du couple MOF+OCL que l'on supposera parfait dans le reste de cet article, ce qui, comme nous l'avons vu en introduction, est discutable. Dans l'espace technique, nous disposons de ECore qui est une implantation d'un sous-ensemble du MOF, EMOF (Essential MOF) et qui a été depuis peu étendu pour supporter les contraintes OCL[3] (invariants et pré/post-conditions pour les opérations).

Nous voulons montrer par notre étude que pour augmenter la formalisation des méta-modèles, l'utilisation des couples MOF/OCL et ECore/OCL n'est pas suffisante et qu'il faut enrichir le niveau méta-modèle pour garantir la conformité des modèles au regard de leur définition.

Notre propos est de souligner, par cette implantation, les différents points non formalisés dans la spécification, de montrer les limitations des outils utilisés pour décrire les méta-modèles et de proposer des améliorations afin de préciser la sémantique des méta-modèles pour diminuer la part d'interprétation des outilleurs. À cette fin, nous nous appuierons sur un langage de méta-modélisation comme KerMeta (Pierre-Alain *et al.*, 2005) qui permet de renforcer la définition classique d'un méta-modèle (MOF + OCL) par des opérations et des propriétés qui dénotent de sa sémantique opérationnelle. Ce renforcement sémantique permet d'une part la mise à jour de la structure du modèle au regard des contraintes, et d'autre part l'exécutabilité des modèles.

3. Expérimentation

L'infrastructure d'UML2 est constituée de deux paquetages qui sont le paquetage *core* et le paquetage *profiles*. Nous nous intéressons ici au premier car il constitue réellement le coeur de tout méta-modèle à l'OMG. Celui-ci est constitué de 4 sous-paquetages qui sont les paquetages *PrimitiveTypes*, contenant la définition des types primitifs de bases pour la définition de méta-modèles, *Abstractions*, contenant principalement des méta-classes abstraites devant être raffinées, *Constructs*, contenant principalement des méta-classes concrétisant les concepts du paquetage *Abstractions* et enfin *Basic*, contenant un sous ensemble du paquetage *Constructs* servant à la défi-

3. eclipse ocl, http ://www.eclipse.org/modeling/mdt/ ?project=ocl

nition du format XMI. Nous nous attachons ici à décrire l'implantation du paquetage Constructs.

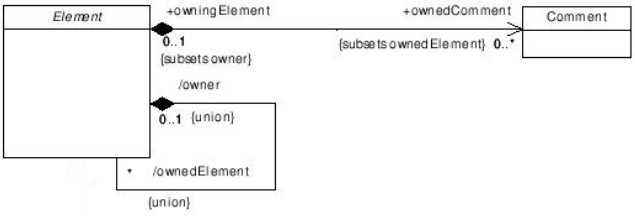

Figure 1. *Extrait du méta-modèle avec la méta-classe* Element

La classe abstraite *Element* est le constituant de base de tout modèle (cf fig 1). Dans la spécification, il est stipulé qu'un élément peut contenir toute une hiérarchie d'éléments en veillant à ce qu'il ne se contienne pas lui même, directement ou indirectement. Il est également indiqué que son ensemble *ownedElement* est une union dérivée de l'ensemble des associations composites qui ont pour source l'élément courant. Cela signifie que l'ensemble des éléments contenus peut être calculé automatiquement à partir des éléments composites représentés par les associations qui ont pour source l'élément courant. La projection dans l'espace technique EMF n'est pas directement possible car la notion d'association n'y a pas été réifiée, contrairement au MOF. Par conséquent, nous ne pouvons pas accéder directement aux associations.

Cependant, une projection est possible en utilisant la notion de *EReference* et la relation *eOpposite* pour manipuler la notion de composite. La résolution partielle de ce problème peut s'implanter en KerMeta par un attribut dérivé s'exprimant par le mot clé *property* accompagné d'un getter et/ou d'un setter, mais nous n'avons pas cherché à implanter cette solution partielle que nous jugeons insuffisante puisque nous ne pouvons pas, par exemple, exprimer une association sans navigabilité à ses 2 extrémités. Ce premier exemple illustre donc le problème d'une projection dans un espace technique trop restreint, mais qui pourrait être améliorée par l'ajout de concepts et d'opérations au niveau du méta-modèle.

Pour poursuivre notre démonstration, nous avons étudié les notions d'ensemble et de sous-ensemble qui sont utilisées implicitement, en MOF, KerMeta ou en ECore. Toujours sur la figure 1, les éléments de type *Comment* sont vus comme faisant partie du sous-ensemble de *OwnedElement*. Cette propriété structurelle intéressante ne peut pas trouver d'implantation directe en ECore car elle va au delà de la simple structure du modèle. Pour l'exprimer en KerMeta, les *properties* peuvent être utilisées mais la dérivation et le sous-ensemble n'ont pas la même sémantique. Pragmatiquement, 2 techniques sont à envisager pour exprimer le sous-ensemble *ownedComment* de l'ensemble *ownedElement* :

– ajouter un nouvel ensemble, auquel cas on se retrouve avec deux ensembles distincts et une solution sémantiquement fausse,

```
abstract class Element
{
    attribute _mustBeOwned : Boolean[1..1]
    operation mustBeOwned () : Boolean is
    do
        result := if self._mustBeOwned == void then true else self._mustBeOwned end
    end
    attribute ownedElement : set Element[0..*]#owner
    property ownedComment : set Comment[0..*]
        getter is
        do
            result := Set<Comment>.new
            var tmp : Comment
            self.ownedElement.select{ e | Comment.isInstance(e) }.each{ c |
                tmp ?= c
                result.add(tmp)
            }
        end
    reference owner : Element#ownedElement
    operation allOwnedElements() : Set<Element> is
    do
        result := Set<Element>.new
        if self.ownedElement.size() != 0 then
            result.addAll(self.ownedElement)
            self.ownedElement.each{ e | result.addAll(e.allOwnedElements()) }
        end
    end
    inv cannotContainsItSelf is not self.allOwnedElements().exists{ e | e == self }
    inv mustHaveOwner is (not self.mustBeOwned()) or (self.owner != void)
}
```

Figure 2. *Code KerMeta pour la méta-classe* Element

– faire de l'attribut *ownedComment* une *property*, accompagné d'un *getter*, ce qui apparaît ici comme une solution correcte mais qui, comme nous le verrons par la suite, n'est pas toujours applicable. Cette solution est illustrée dans le code KerMeta de la figure 2, via l'implantation de la *property OwnedComment*. Cette *property* permet la sélection des éléments de type *Comment* dans la collection des éléments composites de l'élément courant.

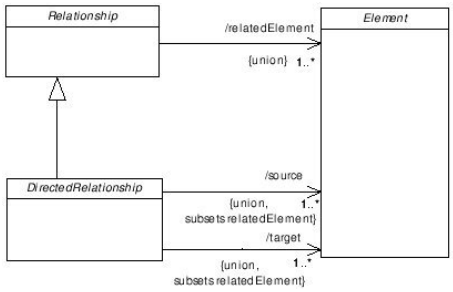

Figure 3. *Extrait de modèle avec la méta-classe* Relationship

La figure 3 présente un exemple ne pouvant pas être résolu par l'utilisation de *property*. La classe abstraite *Relationship* spécifiant une relation entre un ensemble d'éléments est raffinée en *DirectedRelationship* où l'on distingue parmi les éléments reliés un sous-ensemble *source* et un sous-ensemble *target*. L'expression de la collection *relatedElement*, comme l'union du sous-ensemble *source* et du sous-ensemble *target*, dans l'espace technique ECore est simplement impossible car la notion de sous-

ensemble n'y est pas réifiée. Une solution simple en KerMeta n'est pas envisageable pour traiter cette notion.

En revanche, de part l'exécutabilité offerte par KerMeta, cette relation bidirectionnelle entre deux ensembles d'éléments peut servir de support pour l'implantation d'une opération de mise à jour ou de propagation des modifications entre les éléments *source* et *target*. Cette opération faisant partie du méta-modèle de manière intrinsèque, elle permet la synchronisation des éléments indépendamment des vues auxquelles ils appartiennent, ceci dans le cas de l'exploitation du méta-modèle au travers d'un environnement multi-vues. La cohérence et la conformité des modèles restent alors strictement au niveau du méta-modèle.

4. Conclusion

Notre objectif est de formaliser au mieux les méta-modèles car, comme nous l'avons mentionné précédemment, une partie de la sémantique des méta-modèles (conformité, comportement) se retrouve en grande partie exprimée dans les outils. Notre but est de l'exprimer complètement dans les méta-modèles pour éviter les ambiguïtés et de minimiser les efforts lors du couplage entre la syntaxe abstraite et la syntaxe concrète. Dans cet article, nous n'avons souligné que quelques problèmes rencontrés lors de notre implantation de l'infrastructure d'UML2. Faute de place, nous avons passé sous silence les erreurs que nous avons pu déceler dans la spécification (redondance, concepts inutiles, contraintes OCL inutiles, incohérences, etc).

Notre implantation en KerMeta nous permet de considérer plusieurs degrés de formalisation : le premier est basé sur les concepts structuraux liés directement à ECore, le second intègre les règles OCL de la spécification (comme l'illustre les invariants déclarés dans la méta-classe *Element* de la figure 2), le dernier étant basé sur l'utilisation des propriétés et des opérations KerMeta pour compléter la formalisation. Ce dernier point ne s'assure pas une formalisation totale par rapport à la spécification, mais améliore grandement la qualité des méta-modèles.

5. Bibliographie

Alanen M., Porres I., « Basic Operations Over Models Containing Subset and Union Properties », *TUCS Department of Information Technologies*, 2006.

Diskin Z., Dingel J., « Mappings, maps and tables : towards formal semantics for associations in UML2 », *School of Computing, Queen's University*, 2005.

Kobryn C., « UML 2001 : A standardization Odyssey », *Communications of the ACM*, 1999.

OMG, MOF 2.0, Technical Report n° ac/03-04-08, Object Management Group, 2006a.

OMG, UML 2.1 Infrastructure, Technical Report n° ptc/06-04-03, Object Management Group, 2006b.

Pierre-Alain M., Fleurey F., Jézéquel J.-M., « Weaving Executability into Object-Oriented Meta-Languages », *Proceedings of MODELS/UML'2005*, Montego Bay, Jamaica, 2005.