Séparation des préoccupations en phase de méta-modélisation

Olivier Barais

Projet Triskell/IRISA
Campus de Beaulieu.
F - 35 042 Rennes Cedex
barais@irisa.fr

RÉSUMÉ. *Ce document présente succinctement le problème de la claire séparation de la définition des différentes formes de sémantique en phase de méta-modélisation afin de permettre l'utilisation du MOF ou d'ecore pour la définition du méta-modèle, l'utilisation d'OCL pour la définition de la sémantique statique et l'utilisation de Kermeta pour la définition de la sémantique opérationnelle. L'accent dans cette courte présentation est mis sur la justification du besoin de cette séparation et les choix techniques retenus basé sur un modèle pivot et un mécanisme de composition de modèles inspiré du merge d'UML.*

1. Motivations

La méta-modélisation dans le domaine de l'informatique se définit comme la mise en évidence d'un ensemble de concepts pour un domaine particulier. Un modèle est une abstraction d'un phénomène du monde réel tandis qu'un méta-modèle est une abstraction d'ordre supérieur mettant en évidence les concepts utilisés pour définir le modèle. Dans ce domaine, on peut parler d'un modèle conforme à son méta-modèle comme on peut parler d'un programme conforme à sa grammaire. L'OMG au travers du MOF [1] propose un langage standardisé afin de permettre la définition de nouveaux méta-modèles. Kermeta [2], développé au sein de l'équipe Triskell, est un langage de méta-modélisation exécutable construit comme une extension du MOF. Il permet de définir un méta-modèle auquel on associe une sémantique opérationnelle pour permettre la simulation des modèles. Le MOF ainsi que Kermeta permettent de définir un certain nombre de contraintes sur le modèle représenté, portant entre autres sur la cardinalité des relations entre les concepts. Ces contraintes peuvent servir à vérifier la cohérence d'un modèle par rapport à son méta-modèle. Cependant, un langage comme le MOF manque, au même titre qu'UML, d'expressivité pour représenter un certains nombre de contraintes dans les futurs méta-modèles. Un exemple consiste à regarder la définition d'un méta-modèle qui permet d'écrire des modèles de composants possédant des ports et des opérations fournies et requises au niveau de ses ports, deux composants étant reliables grâce à un concept de connecteur entre leurs ports. Il est impossible avec un langage comme le MOF o de définir la sémantique statique de ce méta-modèle ; par exemple définir dans ce méta-modèle ce qu'est un assemblage de composants valides. OCL (*Object Constraint Language*) incorpore la notion de conception par contrats [4] en UML. Largement étendu depuis la première version datant de 1999, la version 2 du langage [3] propose actuellement une syntaxe déclarative simple fondée sur une logique du premier ordre permettant à la fois d'associer des contraintes à un méta-modèle et de définir des requêtes sur des modèles. Sa proximité avec UML est, en outre, moins forte autorisant son utilisation dans le cadre de la méta-modélisation. L'utilisation d'OCL dans le cadre de la méta-modélisation offre un moyen simple d'exprimer la sémantique statique d'un méta-modèle dans le cas où celle-ci est exprimable avec une logique du premier ordre. Kermeta supporte nativement la notion de conception par contrats « à la Eiffel ». La définition des invariants et des pré-post conditions se fait alors à l'aide de la syntaxe concrète Kermeta (voir exemple de la Figure 1). Le support de la syntaxe concrète d'OCL en fournissant une transformation de modèles entre l'AST d'OCL et l'AST de Kermeta. Cette transformation est écrite en Kermeta. Le modèle résultant de la transformation est automatiquement associé avec le méta-modèle auquel il s'applique grâce à un mécanisme de composition appelé inspiré du *merge* d'UML. Le schéma de la figure 2 illustre l'architecture générale de la transformation. C'est ce mécanisme de compsoition que nous souhaitons présenter au cours de cette présentation. Nous montrerons en outre, d'autres cas d'utilisation possibles de ce mécanisme.

```
operation foo(param : String) : String
   pre myprecondition is param != ""
   post apostcondition is result.size > param.size is do
       result := param + " world"
   end
```

Figure 1 : Syntaxe Kermeta pour la définition des pré et des post-conditions

D'un point de vue méthodologique pour le concepteur de méta-modèle, cette architecture permet une bonne séparation des préoccupations entre la description du méta-modèle, la description de la sémantique statique et la description de la sémantique opérationnelle. Chacune de ces sémantiques étant en effet définies à l'aide d'une syntaxe dédiée dans une unité dédiée : avec un fichier *ecore*[1] par exemple pour le méta-modèle, un fichier Kermeta pour la sémantique opérationnelle et un fichier OCL pour la sémantique statique.

[1] Du framework Eclipse EMF (http://www.eclipse.org/emf/)

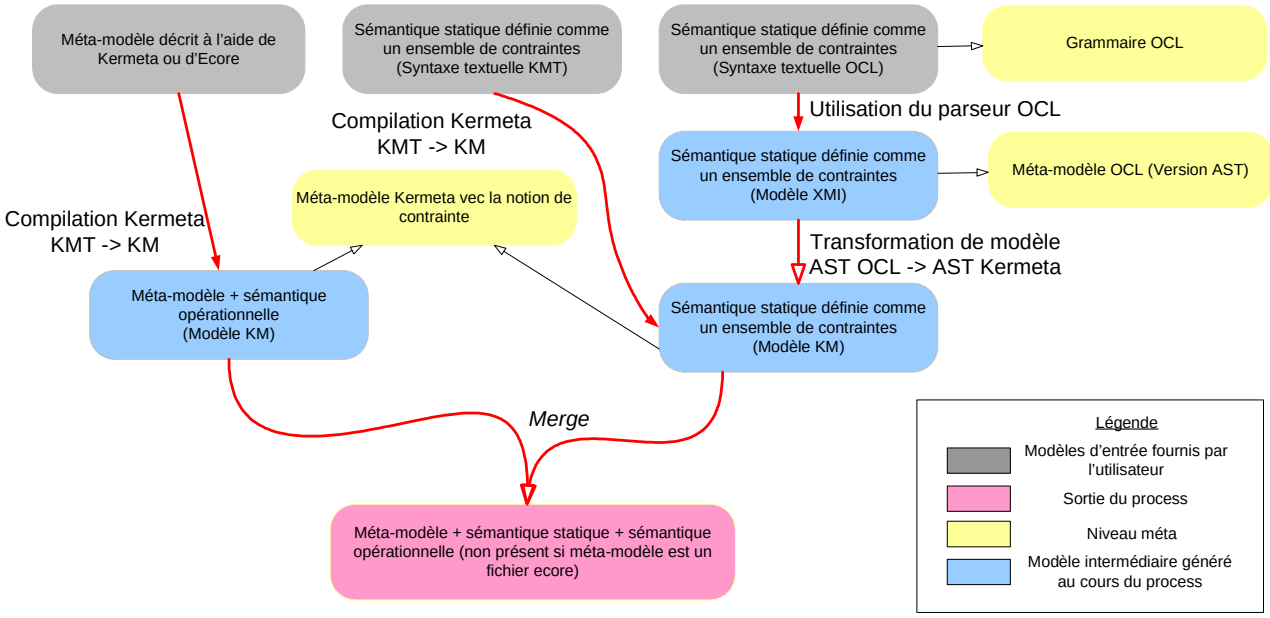

Figure 2 : Vue globale de l'architecture

2. Mécanisme de composition

Le mécanisme de composition est intégré nativement à Kermeta. Le système de type n'est alors vérifié que sur le résultat de cette composition. Ce mécanisme est construit pas une extension de la sémantique de la primitive `require` de Kermeta. Cette dernière devient une primitive de composition à part entière. Son utilisation est illustrée dans la figure 3. Comme dans l'exemple pour un modèle de composant appelé Speeds, il est possible de décrire dans un programme Kermeta, une dépendance vers le un fichier EMF dont la sémantique opérationnelle est définie dans un fichier KMT et dont la sémantique statique est définie dans un fichier OCL. Le méta-modèle résultant est obtenu après une transformation d'EMF vers Kermeta et une transformation OCL vers Kermeta. La composition de ces trois entités produit un méta-modèle avec une sémantique opérationnelle et une sémantique statique. Pour éviter que ce mécanisme de composition soit utilisé par défaut dès qu'il y a un conflit de nom. Une classe « *composable* » doit être annotée avec le mot-clé @Aspect. Pour le moment, le mécanisme de composition ne peut composer que des méta-modèles sans conflits. Il est, par exemple, possible de composer une classe avec des méthodes abstraites avec une classe ayant des méthodes concrètes possédant les mêmes signatures mais il n'est pas possible de composer ces deux classes si deux méthodes concrètes possèdent la même signature. Dans le cas d'utilisation initial qui concerne la séparation de la spécification de la sémantique statique et de la sémantique opérationnelle, cette limitation n'est pas gênante. Par contre il est nécessaire d'étendre cette opération de composition afin de pouvoir lui passer en paramètres des directives de composition précisant la sémantique de la composition. En effet, il ne nous semble difficile de définir une unique sémantique de composition de méta-modèles exécutables [5]. L'opérateur de composition doit donc devenir paramétrable afin d'adapter sa sémantique à son contexte d'utilisation.

```
require  "speeds.ecore"
require  "speeds_forSimulation.kmt"
require  "speeds_Constraints.ocl"
```

Figure 3 : Utilisation des primitives de composition

3. Intérêt d'un mécanisme de composition natif dans un langage de méta-modélisation

Le support de ce mécanisme de composition dans Kermeta apporte au moins une autre perspective pour un concepteur de méta-modèle. Dans deux nombreux cas, les opérations ajoutées au niveau des méta-classes sont utilisées que dans le cas d'une transformation précise et n'on pas de lien avec la sémantique opérationnelle. Ces méthodes ont alors tendance à venir polluer le méta-modèle. Ce mécanisme de composition permet d'enrichir la définition du méta-modèle dans un contexte précis tout en gardant un méta-modèle propre pour l'illustration des concepts du domaine auquel il est attaché.

[1] OMG, *MOF 2.0 Core Final Adopted Specification.* 2004.
[2] P-A. Muller, F. Fleurey, and J-M Jézéquel. -- Weaving executability into object-oriented meta-languages. -- In S. Kent L. Briand, editor, *Proceedings of MODELS/UML'2005*, volume 3713 of *LNCS*, pages 264--278, Montego Bay, Jamaica, October 2005. Springer.
[3] OMG, UML 2.0 Object Constraint Language (OCL) Final Adopted specification. 2003.
[4] B. Meyer. Applying "Design by Contract". *Computer* 25, 10 (Oct. 1992), 40-51.
[5] Andrew Jackson, Olivier Barais, Jean-Marc Jézéquel, and Siobhán Clarke. -- Toward a generic and extensible merge. -- In *Models and Aspects workshop, at ECOOP 2006*, Nantes, France, July 2006.