Nouvelles fonctionnalités de Kermeta :

Olivier Barais – Franck Fleurey– Pierre-Alain Muller – Didier Vojtisek - Jean-Marc Jézéquel

Projet Triskell/IRISA
Campus de Beaulieu.
F - 35 042 Rennes Cedex
{prenom.nom}@irisa.fr

RÉSUMÉ. Ce document présente succinctement les développements en cours de Kermeta. L'accent dans cette courte présentation est mis sur la présentation de trois nouvelles fonctionnalités et leurs cas d'utilisation possibles. La démonstration de Kermeta se propose d'illustrer ces trois nouvelles fonctionnalités sur des exemples simples.

1. Introduction

Kermeta [2], développé au sein de l'équipe Triskell, est un langage de méta-modélisation exécutable construit comme une extension du MOF [1]. Il permet de définir un méta-modèle auquel on associe une sémantique opérationnelle pour permettre la simulation des modèles. Dans le cadre de cette démonstration, nous nous proposons de présenter trois développements en cours pour respectivement :

- supporter la syntaxe concrète d'OCL 2.0,
- améliorer la séparation des préoccupations entre la description du méta-modèle, la spécification de sa sémantique statique et la spécification de sa sémantique opérationnelle
- et finalement proposer un outillage autour du méta-modèle Java afin de pouvoir considérer tout programme Java comme un modèle instance de son méta-modèle. L'objectif est alors d'utiliser Kermeta pour faire des transformations sur ces modèles.

2. Support de la syntaxe concrète d'OCL

Des langages comme le MOF ou UML, manque intrinsèquement d'expressivité pour représenter un certain nombre de contraintes dans les futurs méta-modèles. Un exemple consiste à regarder la définition d'un méta-modèle qui permet de décrire des modèles de composants possédant des ports et des opérations fournies et requises au niveau de ses ports ; deux composants étant liables grâce à un concept de connecteur entre leurs ports. Il est impossible avec un langage comme le MOF ou UML de définir la sémantique statique de ce méta-modèle, entre autres de préciser 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, depuis la version 2, 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 un mécanisme 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). L'intégration d'OCL dans Kermeta est faite à l'aide d'une transformation de modèle 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 aussi *merge* de modèles (voir Section 3). Le schéma de la figure 2 illustre l'architecture générale de la transformation.

```
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

Le support d'OCL dans Kermeta apporte, au moins, deux principaux avantages pour un concepteur de méta-modèle. Pour les concepteurs de méta-modèle qui utilisent déjà Kermeta, ce support améliore la compatibilité avec les standards existants et permet l'intégration de contraintes définies en OCL. Ce mécanisme permet alors dans tous les cas de définir simplement la sémantique statique d'un méta-modèle. Kermeta offre ainsi un moyen de vérifier à la demande la correction d'un modèle avec sa sémantique statique au cours d'une simulation. De plus, pour les utilisateurs d'OCL, l'implémentation d'OCL en Kermeta offre un moyen très simple de mettre en œuvre une extension d'OCL. En effet, OCL offre la possibilité d'appeler

des méthodes définies dans des méta-classes du méta-modèle à partir du moment où ces méthodes n'ont pas d'effet de bord sur le modèle. La définition en Kermeta de la sémantique opérationnelle de ces méthodes permet une mise en œuvre et une évaluation de ces nouvelles primitives du langage.

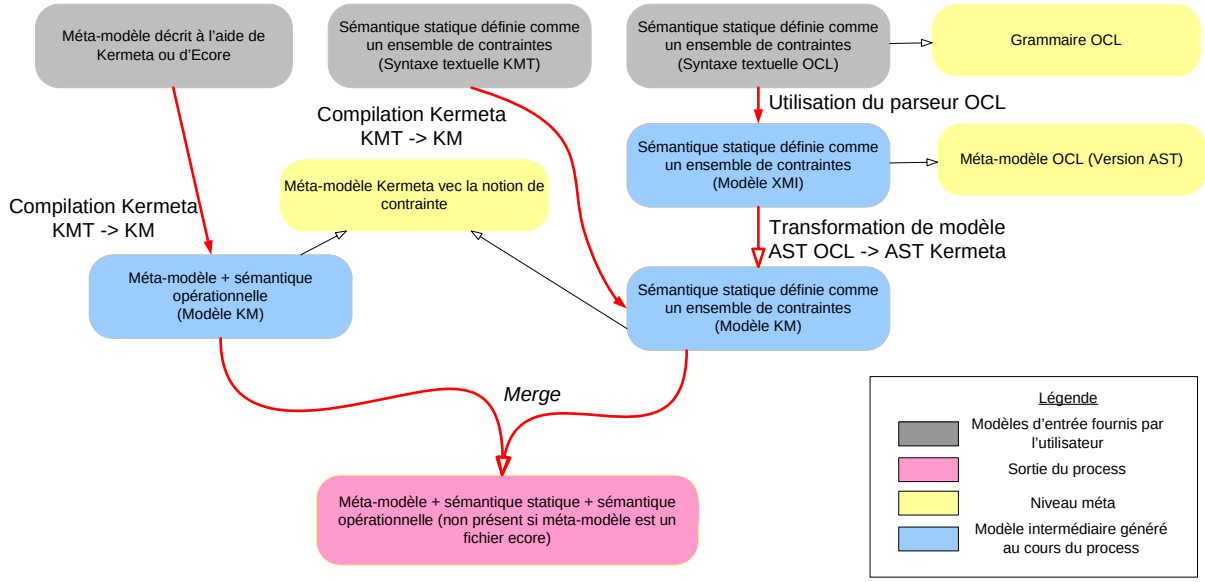

Figure 2 : Vue globale de l'architecture

3. Séparation des préoccupations lors des phases de méta-modélisation

D'un point de vue méthodologique pour le concepteur de méta-modèle, il existe différente manière de décrire la sémantique du domaine lié au méta-modèle. Kermeta supporte la définition de la sémantique opérationnelle et la définition de la sémantique statique du méta-modèle. Grâce à un nouveau mécanisme de composition inspiré du *merge* d'UML, il est maintenant possible de séparer la définition du méta-modèle, de la sémantique statique et de la sémantique opérationnelle. Chacune de ces sémantiques est alors définie 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. Nous proposons dans cette démonstration de présenter le fonctionnement de ce mécanisme de composition.

4. Java comme modèle

Enfin, Kermeta dispose maintenant d'un ensemble de MDKs (*Model Development Kit*). Un MDK est un méta-modèle associé à un ensemble d'outils (plugins eclipse) pour faciliter l'utilisation de ce méta-modèle. Le MDK Java de Kermeta, appelé SpoonEMF est basé sur le projet Spoon [5]. Le principal intérêt de *SpoonEMF* est de pouvoir considérer tout programme Java comme un modèle instance de son méta-modèle. Dès lors, *Spoon* considérant toutes les informations d'un programme java, il est, par exemple, possible d'exploiter les annotations contenues dans un programme Java pour piloter une transformation de ce code source. Deuxièmement, par rapport à *Spoon* lui-même, le fait de proposer une implémentation utilisant EMF permet d'utiliser ce méta-modèle dans le cadre de nombreux projets existants comme *Kermeta* ou ATL. Finalement, quand *Spoon* propose une représentation du méta-modèle sous forme d'un ensemble d'interfaces Java, *SpoonEMF* propose la représentation du méta-modèle avec un formalisme adapté à savoir *ecore*. Ainsi, les attributs des concepts du méta-modèle sont décrits sous forme d'attributs (non pas sous forme d'accesseurs), les associations entre les concepts du méta-modèle sont identifiées et décrites. Certains patrons de conception comme le patron visiteur sont retirés du méta-modèle et uniquement mis en œuvre au niveau de l'arbre de syntaxe abstraite. Ces points permettent de simplifier la compréhension du méta-modèle comparé à une description du méta-modèle sous forme d'interfaces Java. Nous proposons dans cette démonstration de présenter le plugin Eclipse SpoonEMF pour l'utilisation du MDK Java au travers d'une transformation pour faire des métriques sur les programmes Java.

[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] R. Pawlak, C. Noguera and N. Petitprez. Spoon: Program Analysis and Transformation in Java. INRIA Research Report #5901, May 2006.

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