Métamodèles et Points de Variation Sémantique

Arnaud Cuccuru – Chokri Mraidha – François Terrier – Sébastien Gérard

CEA, LIST, Gif-sur-Yvette, F-91191, France

{arnaud.cuccuru, chokri.mraidha, francois.terrier, sebastien.gerard}@cea.fr

RÉSUMÉ. *Les points de variation sémantique constituent une information essentielle dans le domaine de l'ingénierie des langages. Souvent liés à la sémantique dynamique des systèmes, les identifier et les comprendre est indispensable pour toute activité relative à la conception ou à l'exploitation d'un modèle (simulation, test, vérification formelle, etc.). La plupart du temps, les points de variation sont seulement identifiés de façon informelle dans une description associée au métamodèle décrivant le langage. Il n'existe actuellement pas de moyens pour les identifier et les fixer explicitement au niveau du métamodèle. Pour combler ce manque, nous proposons une notation à base de templates. Nous illustrons l'utilisation de cette notation de façon pragmatique par une version templatisée du métamodèle des machines à états d'UML 2.* [1]

ABSTRACT. *In the field of Domain Languages Engineering, Semantic Variation Points are an important issue. This crucial information is often related to the dynamic semantics of systems. Identifying and understanding it is a requisite for all model-based activities (design, simulation, test, formal verification, etc.). Most of the time, semantic variation points are only informally identified in a documentation associated with the metamodel describing the language. There is currently no mechanism to capture and fix them explicitly within a metamodel. We propose a template-based notation to solve this probleme. We illustrate our proposal in a pragmatic way with a templated version of the UML 2 state machine metamodel.*

MOTS-CLÉS : *Métamodèle, Template, Point de variation sémantique.*

KEYWORDS: *Metamodel, Template, Semantic Variation Point.*

[1] Ces travaux ont été réalisés dans le contexte du projet Usine Logicielle (www.usinelogicielle.org) du pôle System@tic Paris-Région avec le soutien de la Direction Générale des entreprises, du Conseil Régional d'Île de France, du Conseil Général des Yvelines, du Conseil Général de l'Essonne et du Conseil Général des Hauts de Seine.

1. Introduction

La prédominance des métamodèles dans les approches orientées modèle pose la question de leur réutilisation potentielle. La capacité à réutiliser/spécialiser tout ou partie de métamodèles existants est essentielle afin d'éviter le redéveloppement systématique et complet de nouveaux métamodèles. Pour traiter cette problématique, le MOF (OMG, 2004) (standard OMG pour la métamodélisation) définit des constructions inspirées par les mécanismes du paradigme Objet, tels que l'import ou l'héritage. Depuis Simula-67, les langages orientés objet ont évolués, allant au-delà de ce que l'on considère usuellement comme les fondations de ce paradigme : Objets, classes, encapsulation, héritage et polymorphisme. Principalement motivés par une volonté de réutiliser et de capitaliser davantage, certains langages orientés objet (tels que C++ ou Java) ont intégré la Généricité -- un mécanisme apparu dans Ada 83 -- permettant de rendre les éléments plus facilement réutilisables et spécialisables, par des définitions génériques basées sur des paramètres templates. Des comportements génériques peuvent ainsi être exprimés (quasi-) indépendamment du type final de ces paramètres génériques. Nous estimons que le MOF tirerait profit d'une évolution similaire, en particulier à une époque où la communauté de l'IDM s'intéresse de plus en plus aux spécifications comportementales exécutables au niveau de la métamodélisation (Clark *et al.*, 2004). Cette affirmation se vérifie particulièrement dans le domaine de l'ingénierie des langages, une activité fondamentale de l'IDM.

Si le MOF supportait les descriptions comportementales, le métamodèle d'un DSML (Domain Specific Modeling Language) embarquerait typiquement une sémantique opérationnelle, et un modèle donné respectant ce métamodèle pourrait être exécuté/simulé. La sémantique opérationnelle potentielle des DSMLs accentue néanmoins la problématique des points de variation sémantique, souvent liés à la sémantique de la dynamique des systèmes. Les identifier et les comprendre est indispensable pour toute activité relative à la conception ou à l'exploitation d'un modèle (simulation, test, vérification formelle, etc.). Indépendamment d'un quelconque formalisme pour la description de la sémantique opérationnelle, le MOF ne fournit pas de moyen pour déclarer et fixer explicitement les points de variation sémantiques. Pour les utilisateurs du métamodèle, l'information est uniquement accessible dans une documentation associée au métamodèle (généralement décrite de façon informelle en langage naturel, comme c'est par exemple le cas pour le métamodèle d'UML 2), ou plongée dans une description comportementale si le métamodèle embarque une sémantique opérationnelle (Chauvel *et al.*, 2005).

Pour combler ce manque, nous proposons d'introduire au niveau méta la capacité à définir des paramètres templates, où les paramètres peuvent être liés aussi bien au niveau méta qu'au niveau modèle. Nous illustrons de façon pragmatique la façon dont les templates peuvent adresser la problématique des points de variation sémantique, par la définition d'une version paramétrée du métamodèle des machines à états d'UML 2.

2. Les machines à états d'UML 2

Une machine à états possède une ou plusieurs régions, elles-mêmes composées de sommets (états ou pseudo-états) et de transitions reliant ces sommets. Les transitions sont gardées par une contrainte, et déclenchées par un trigger référençant un évènement déclencheur. Tirer une transition provoque l'exécution du comportement (potentiellement) associé, et la modification de l'état courant de la région, de l'état source à l'état cible de la transition. Pour simplifier notre présentation, nous considérerons que les machines à états ne possèdent qu'une seule région. Pour une description plus détaillée, nous invitons le lecteur à consulter le chapitre relatif aux machines à états, dans la superstructure d'UML 2.

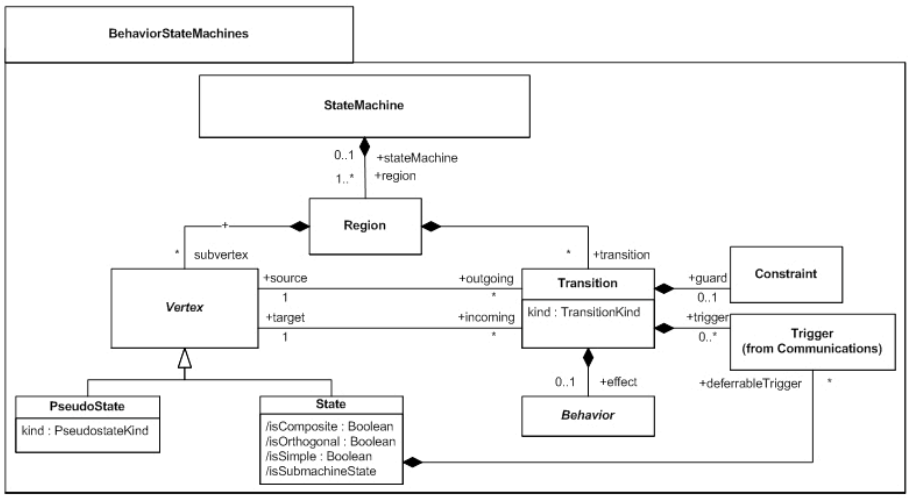

Figure 1 Métamodèle simplifié des machines à états d'UML 2

Pour notre présentation, nous n'allons nous attarder que sur deux des nombreux points de variation sémantique des machines à états : les politiques de sélection des évènements et de sélection des transitions. Lors d'une exécution, une machine à états accède à un pool d'évènements géré par l'objet contexte de la machine. En fonction de l'état courant de la machine et de l'ensemble d'évènements pertinents contenus dans le pool (c'est-à-dire ceux pouvant déclencher une transition à partir de l'état courant), la politique de sélection des évènements détermine un ordre pour l'extraction des évènements du pool, et offre la possibilité de mettre en œuvre différentes politiques de gestion des priorités. Dans les paragraphes suivants, nous considérons les politiques LIFO et FIFO comme politiques concrètes de sélection des évènements. Une fois que les évènements ayant la priorité la plus élevée ont été sélectionnés, la politique de sélection des transitions détermine la transition à tirer dans le cas où plusieurs évènements ont le même niveau de priorité. Nous considérons les politiques RANDOM et STOCHASTIC comme sémantiques concrètes pour la politique de sélection des transitions.

3. Une Version Templatisée du Métamodèle des Machines à Etats d'UML 2

En figure 2, nous illustrons comment le métamodèle des machines à états d'UML 2 peut être templatisé. La notation utilisée pour les templates est celle proposée dans UML 2. Dans le package TemplatedBehaviorStateMachine, les éléments modélisés sont ceux ayant été spécialisés ou ajoutés (sur la base du métamodèle défini dans le package mergé) afin de prendre en compte l'aspect dynamique d'une exécution. Certaines associations ont ainsi été ajoutées (currentState d'une StateMachine), certaines métaclasses introduites (EventOccurence), et certaines opérations déclarées sur les métaclasses.

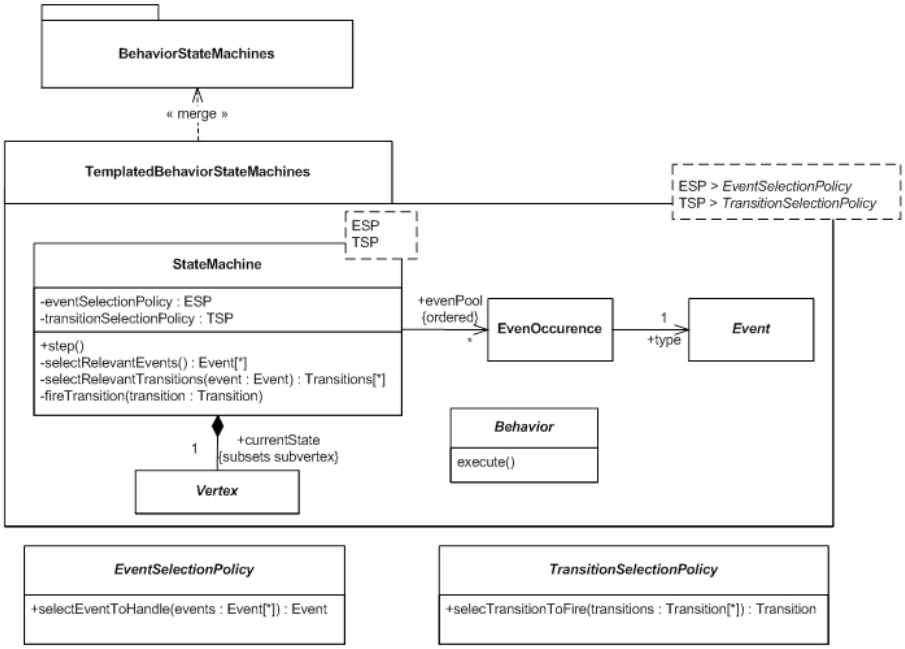

Figure 2 Métamodèle templatisé des machines à états d'UML 2

Examinons maintenant la façon dont un lecteur anonyme (mais perspicace) pourrait réagir s'il était confronté à ce diagramme. Deux paramètres templates sont identifiés au niveau du package TemplatedBehaviorStateMachine : ESP (un type qui doit être compatible avec la métaclasse abstraite EventSelectionPolicy), et TSP (un type compatible avec TransitionSelectionPolicy). Ces deux paramètres sont à leur tour référencés par la métaclasse StateMachine. Les métaclasses EventSelectionPolicy et TransitionSelectionPolicy possèdent des opérations indiquant que des comportements sont encapsulés à l'intérieur de ces métaclasses. Intuitivement, notre lecteur perspicace comprend que les comportements associés avec ces paramètres vont avoir un impact sur le comportement de la métaclasse paramétrée StateMachine. Les deux paramètres templates lui fournissent un moyen

de faire varier la sémantique opérationnelle de la StateMachine. En d'autres termes, ces paramètres identifient de façon explicite les points où la sémantique des machines à états d'UML peut varier. Cette intuition pourrait être facilement confirmée si une sémantique opérationnelle était encapsulée dans l'opération step() de la métaclasse générique StateMachine, où les comportements associés aux paramètres templates seraient explicitement référencés par des appels d'opérations sur les attributs eventSelectionPolicy:ESP et transitionSelectionPolicy :TSP de la machine.

Une fois les points de variation sémantique identifiés par des paramètres templates, la façon la plus naturelle de les fixer est de lier les paramètres avec des valeurs concrètes (c'est-à-dire les types qui seront concrètement manipulés). Dans la figure 3, nous illustrons comment les points de variation sémantiques peuvent être fixés au niveau du métamodèle ou au niveau d'un modèle. Au niveau métamodèle, un nouveau package peut être défini à partir d'un package templatisé par le biais d'une relation de type « bind ». Dans le métamodèle résultant, les références aux paramètres templates sont résolues et remplacées par les valeurs concrètes ayant été liées aux paramètres. En utilisant ce mécanisme, nous créons un package FIFO_Random_StateMachines, dans lequel les machines à état sélectionnent les évènements sur la base d'une politique de type FIFO, et les transitions par une politique de type RANDOM. Au niveau modèle, nous déclarons une machine à état avec la même sémantique, mais la liaison (binding) est réalisée directement au niveau de la déclaration de l'instance de la métaclasse.

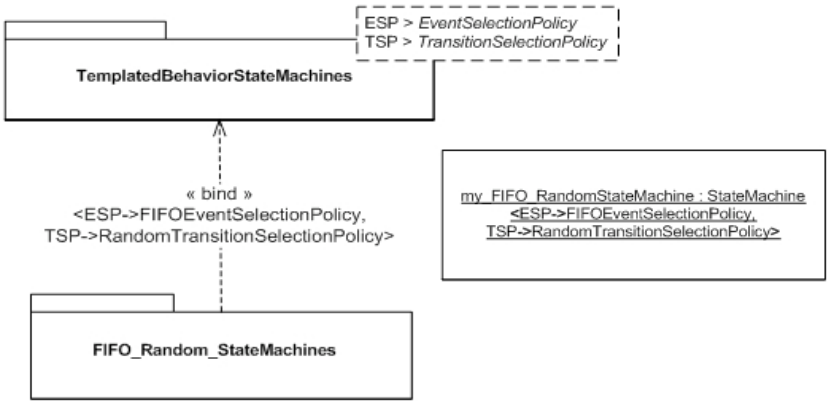

Figure 3 Points de variation sémantique résolus par liaison des paramètres

4. Conclusion

Le MOF ne fournit pas de mécanismes standards pour identifier et fixer explicitement les points de variation sémantique d'un métamodèle, et les travaux relatifs à ce sujet ne proposent pas de solution satisfaisante. Nous avons proposé une solution basée sur des définitions de templates, et montré de façon pragmatique (faute de place pour en faire une démonstration plus formelle) comment les métamodèles templatisables pouvaient nous aider à identifier et fixer les points de variation sémantique de façon explicite et intuitive. Nous n'avons cependant pas encore discuté de la manière dont les métamodèles templatisables pourraient s'intégrer dans les chaînes d'outils orientées modèles traditionnelles.

Nous travaillons actuellement sur la mise en œuvre d'un plug-in Eclipse supportant la définition et l'instanciation de métamodèles templatisés, tels que nous l'avons illustré dans cet article. Nous avons pour l'instant défini une extension du métamodèle de Ecore, inspirée par le sous-ensemble du métamodèle UML 2 relatif aux templates. Le support des templates pourrait être utilisé comme un outil utile pour résonner sur les modèles. L'identification des points de variation sémantique est le bénéfice sur lequel nous avons mis l'accent dans cet article, mais des travaux tels que (Clark *et al.*, 2002) et (Emerson *et al.*, 2006) ont également identifié d'autres bénéfices (composition de métamodèles, définition et application de patrons de conception pour la métamodélisation, etc.). Ainsi, des règles de transformation de modèles adaptées (sur lesquels nous travaillons actuellement) pourraient être appliquées, permettant le remplacement des liaisons des paramètres templates par des valeurs concrètes dans le métamodèle résultant, et donc la production d'un métamodèle standard directement exploitable par une chaîne d'outils classique.

5. Bibliographie

Chauvel F., Jezequel J., « Code generation from UML models with semantic variation points », *8th international conference on Model Driven Engineering Languages and Systems, MODELS*, Montego Bay, Jamaica, 2005.

Clark T., Evans A., Kent S., « Engineering Modelling Languages: A Precise Meta-modelling Approach », *5th International Conference on Fundamental Approaches to Software Engineering, FASE*, Grenoble, France, 2002.

Clark T., Evans A., Sammut P., Willans J., « An eXecutable Metamodeling Facility for Domain Specific Language Design », *4th OOPSLA Workshop on Domain-Specific Modeling, DSM*, Vancouver, Canada, 2004.

Emerson M., Sztipanovits J., « Techniques for metamodel composition », *6th OOPSLA Workshop on Domain-Specific Modeling, DSM,* Portland, Oregon, USA, 2006.

OMG, « Meta Object Facility (MOF) 2.0 Core Specification », 2004.