Approche formelle pour une ingénierie des modèles sûre [1]

Akram Idani*, Yves Ledru*, Pierre-Yves Schobbens**

* Laboratoire d'Informatique de Grenoble
681 rue de la Passerelle – BP 72
38402 Saint Martin d'Hères Cedex
{Akram.Idani, Yves.Ledru}@imag.fr

** Université de Namur, Faculté d'Informatique
21 Rue Grandgagnage
pys@info.fundp.ac.be

Résumé. Aujourd'hui les outils IDM ont atteint un bon niveau de maturité et sont de plus en plus adoptés dans le cadre d'applications complexes et critiques. Toutefois, des questions liées à la sûreté des systèmes qui en découlent restent encore ouvertes. Pour répondre à ces questions, nous proposons de ramener l'IDM dans le monde rigoureux des méthodes formelles. Nos principaux objectifs sont : (1) garantir la cohérence des correspondances entre méta-modèles au moyen d'un outil de preuve, en l'occurrence le prouveur de l'atelier B ; (2) être capable de certifier qu'une transformation de modèles préserve la sémantique des modèles source et cible ; et (3) utiliser des outils d'animation de spécifications pour simuler le comportement des différents modèles mis en jeu dans un cadre IDM.

1 Introduction

L'Ingénierie Dirigée par les Modèles (ou IDM) est une démarche de développement qui conçoit l'intégralité du cycle de développement du logiciel comme un processus de production, de raffinement itératif et d'intégration de modèles. Plus précisément elle fait la différence, entre modèles indépendants (PIM pour Platform Independent Model) et modèles dépendants des plate-formes d'exécution (PSM pour Platform Specific Model). Sur cette base le processus de développement est vu comme étant une transformation progressive d'un modèle PIM, qui spécifie la solution d'un système indépendamment des technologies de programmation, vers un modèle PSM qui décrit comment cette solution peut être implémentée dans une technologie particulière. En ce sens, l'IDM met en œuvre des "règles" pour définir la cohérence entre les différents modèles utilisés ainsi que la manière de les transformer (ou de les composer) au cours du développement. Les environnements qui supportent cette approche (ATL (Jouault

1. Ce travail est financé par le projet SELKIS : A development method of secure health care networks information systems : from requirements engineering to implementation – (ANR-08-SEGI-018).

et al., 2008) par exemple) nécessitent une description des modèles manipulés par des méta-modèles ; ce qui permet de spécifier les règles de transformation PIM/PSM par des correspondances d'un méta-modèle source vers un méta-modèle cible. Les efforts de l'OMG (Object Management Group) dans ce contexte ont donné lieu aux standards MOF (Meta-Object Facility) pour la description des méta-modèles, et QVT pour la spécification des projections entre méta-modèles.

Aujourd'hui, bien qu'on constate un grand savoir-faire méthodologique de l'IDM, des questions liées à la *sûreté/fiabilité* des systèmes qui en découlent sont encore ouvertes. En effet, bien que les environnements IDM existants permettent de spécifier des méta-modèles, ils ne permettent pas de prouver la correction des règles de transformation. Ce n'est donc qu'après l'exécution de la transformation qu'il devient possible de vérifier si le modèle PSM produit est conforme à son méta-modèle. Ceci est dû au fait que pour démontrer qu'une règle de transformation préserve la sémantique des modèles source et cible avant son exécution, des langages de modélisation, clairement définis avec une sémantique précise, doivent être utilisés. Ce qui n'est pas le cas aujourd'hui du standard QVT car il se base en grande partie sur des notations qui n'offrent pas des outils de preuve.

Dans ce contexte, nous proposons de ramener l'IDM dans un monde plus rigoureux, fondé sur la méthode formelle B. Nous ciblons par là un double objectif :

1. Effectuer des vérifications automatisées avant la transformation PIM/PSM, et ce au moyen du prouveur de l'atelier B. Ceci permettra de certifier qu'une transformation est correcte.
2. Utiliser un animateur supportant la méthode B (ProB (Leuschel et Butler, 2003), BZ-TT (Legeard et al., 2002),. . .) pour exécuter la (les) transformation(s). Ceci permettra de générer automatiquement le modèle cible.

2 Validation de transformations de modèles

La notion de transformation de modèles constitue l'élément central de l'IDM. En effet, cette notion porte sur l'automatisation d'opérations critiques pendant le cycle de développement telles que la génération de code. L'automatisation de ces opérations permet d'augmenter la réutilisation du cœur de métier d'un système à un autre, ce qui entraîne des économies en terme d'effort et de temps. Toutefois, dans le cadre d'applications critiques où sûreté et sécurité sont fondamentales, il est primordial d'éviter de produire des transformations de modèles *erronées*. Par conséquent, l'utilisation des techniques classiques de test et de preuve, devient incontournable pour certifier qu'une transformation de modèles est valide.

2.1 Approche basée sur la preuve

Dans l'objectif d'aider à la validation de transformations de modèles en IDM nous proposons une technique de preuve, qui se veut complémentaire aux techniques de test (Brottier et al., 2006). Nous nous inspirons pour ce faire des nombreux travaux de couplage des langages B et UML (Laleau et Polack, 2002; Snook et Butler, 2006). Ces travaux ont cherché à préciser la sémantique d'UML, à compléter ce langage par des annotations formelles pour augmenter son pouvoir d'expression, et à traduire les spécifications semi-formelles en B pour profiter des outils de preuve et d'animation de la méthode B. La figure 1 montre comment notre

approche se situe dans un contexte IDM classique. Les méta-modèles du PIM et du PSM ne sont autres que des diagrammes de classes (spécifiés par le MOF). Il est donc aisé de réutiliser les approches existantes de génération de spécifications B à partir de diagrammes de classes UML en vue de traduire ces méta-modèles en B. Le constat des travaux de couplage de UML et B nous permet de délimiter le sous-ensemble des constructions engendrées par le MOF sur lesquelles ces travaux peuvent être appliqués.

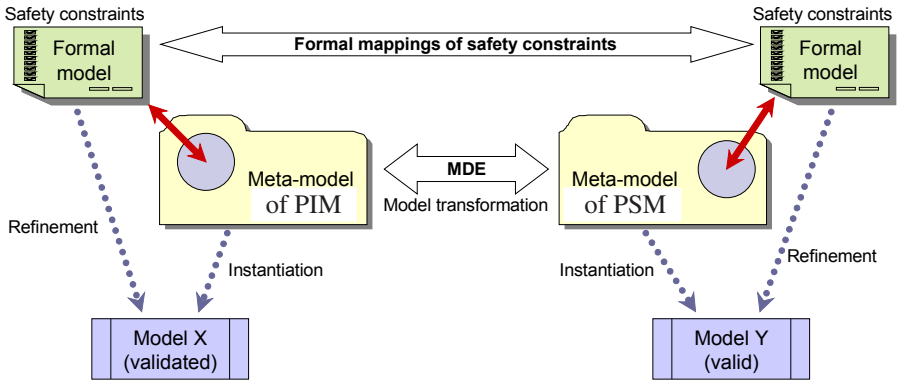


FIG. 1 – *Usage de méthodes formelles pour l'IDM*

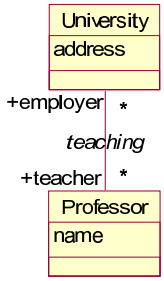
Les spécifications B résultantes peuvent être raffinées par la prise en compte, sous forme d'invariants, de contraintes de sûreté. Par exemple, si on cherche à générer du code Java sûr à partir d'un modèle objet en entrée, on pourrait être amené à interdire l'utilisation de l'héritage multiple. Une telle contrainte peut être exprimée en OCL sur les constructions du méta-modèle objet. Cependant, les outils de preuve de contraintes OCL sont peu nombreux (Brucker et Wolff, 2008), et nécessitent la traduction d'OCL dans des logiques mathématiques. Il est donc opportun de pouvoir exprimer ces contraintes invariantes dans la spécification B. L'avantage en est de pouvoir profiter pleinement de l'environnement de preuve de B. Pour ce faire, il existe deux stratégies : (1) explorer les travaux de traduction d'OCL en B (Marcano-Kamenoff et Levy, 2001) ; ou (2) exprimer les contraintes de sûreté par des annotations B au niveau des méta-modèles du PIM et du PSM. Dans le cadre de nos travaux actuels, nous explorons cette deuxième stratégie.

2.2 Bénéfices du modèle formel

Le modèle formel issu du méta-modèle du PIM peut être utilisé pour vérifier si le PIM est conforme à son méta-modèle et respecte ses contraintes de sûreté. Il suffit pour cela de générer des opérateurs de construction d'instances de méta-modèles. La figure 2 donne un exemple d'opération de construction permettant d'instancier une portion d'un méta-modèle objet.

Quant aux règles de transformation, elles sont injectées dans la spécification formelle via des opérations B. Les opérations **`Rule1`** et **`Rule2`** présentées dans la figure 3 sont des exemples de règles de production de tables relationnelles à partir d'un modèle objet. La règle 1 indique qu'une table, spécifiée en B par « $Relation \in Class \cup Association$ », est produite pour chaque classe et pour chaque association disposant d'une multiplicité "`0..*`". La règle 2 permet de générer les attributs de ces tables. Ces attributs correspondent à la relation

relation_attribute définie comme suit : $relation\_attribute \in Relation \leftrightarrow (UMLAttribute \cup AssociationEnd)$.



build_source_model =
BEGIN
|| $Class := \{University, Professor\}$
|| $UMLAttribute := \{name, address\}$
|| $AssociationEnd := \{teacher, employer\}$
|| $ClassifierStFeature := \{(name \mapsto Professor), (address \mapsto University)\}$
|| $Association := \{teaching\}$
|| $multiplicity1 := \{(teacher \mapsto zero),(employer \mapsto zero)\}$
|| $multiplicity2 := \{(teacher \mapsto star),(employer \mapsto star)\}$
|| $ClassifierAssocEnd := \{(teacher \mapsto Professor), (employer \mapsto University)\}$
|| $AssociationAssocEnd := \{(teacher \mapsto teaching), (employer \mapsto teaching)\}$
END

FIG. 2 – *Instanciation du méta-modèle source*

Rule1 =
PRE $Relation = \emptyset$ **THEN**
$Relation := Class \cup \{aa \mid aa \in Association$
$\wedge\ multiplicity1[AssociationAssocEnd^{-1}[\{aa\}]] = \{zero\}$
$\wedge\ multiplicity2[AssociationAssocEnd^{-1}[\{aa\}]] = \{star\}\}$
END;

Rule2 $\hat{=}$
PRE $relation\_attribute = \emptyset$ **THEN**
$relation\_attribute := \{cc, at \mid cc \in Relation \wedge at \in UMLAttribute \cup AssociationEnd \wedge$
$((at \mapsto cc) \in ClassifierStFeature \vee (at \mapsto cc) \in AssociationAssocEnd)\}$
END

FIG. 3 – *Règles de transformation exprimées en B.*

L'utilisation d'un animateur B, tel que ProB, pour animer la séquence :

`build_source_model ; Rule1 ; Rule2 ;`

transforme le diagramme de classes de la figure 2 telle que : *Relation* = {*University*, *Professor*, *teaching*}, et *relation_attribute* = {(*Professor* $\mapsto$ *name*), (*University* $\mapsto$ *address*), (*teaching* $\mapsto$ *teacher*), (*teaching* $\mapsto$ *employer*)}. Les autres règles de transformation d'un modèle objet en un modèle relationnel (*e.g.* génération des clés primaires et étrangères, etc) peuvent être exprimées de manière similaire.

La preuve de correction des règles de transformation permet de montrer, d'une part, qu'en partant d'un PIM valide, une transformation produit un PSM conforme à son méta-modèle et respectant les contraintes de sûreté de ce dernier ; et d'autre part, le respect des invariants de liaison des méta-modèles du PIM et du PSM. Par exemple, l'invariant de liaison ci-dessous, contraint l'ensemble des associations traduisibles en tables relationnelles. Il indique que seules les associations dont les multiplicités, sur les deux extrémités, sont égales à "**`0..*`**" ou "**`1..*`**" peuvent donner lieu à des tables dans le modèle relationnel. Le prouveur de l'atelier B permet ainsi de vérifier que la règle **`Rule1`** satisfait cette contrainte.

$\forall rr \cdot (rr \in Relation \wedge rr \in Association \Rightarrow$
$multiplicity1[AssociationAssocEnd^{-1}[\{rr\}]] \subseteq \{zero, one\}$
$\wedge\ multiplicity2[AssociationAssocEnd^{-1}[\{rr\}]] = \{star\})$

3 Perspectives

Une grande majorité des travaux autour de l'IDM s'orientent vers les langages et techniques pour le développement de transformations de modèles mais, parmi ces travaux, très peu s'intéressent aux problèmes de la validation/certification. Les techniques de test et de preuve, dans leur dualité, permettent d'aider à cette validation. Cet article présente les premiers pas d'une IDM fondée sur la preuve et l'animation, et ce en explorant une approche dite *interprétée* (Facon et Laleau, 1995) ou *deep encoding*. Ce travail sera étendu dans les mois prochains en tenant compte des points suivants :

- Approfondir l'identification des constructions des méta-modèles traduisibles en B en définissant les correspondances entre le MOF et le langage B.
- Adapter les règles de transformation de UML vers B proposées dans la "littérature" à un contexte orienté IDM.
- Étudier les liens entre opérations B et règles initialement exprimées en QVT.

Par ailleurs, nous songeons à mener des études sur la génération automatique de modèles d'entrée pour simuler de façon pertinente les transformations au moyen d'animateurs B. En effet, dans le cas où les preuves formelles sont très complexes, l'identification de modèles d'entrée pertinents peut aider à la validation des transformations. Des outils comme BZ-TT (Legeard et al., 2002) peuvent servir pour générer automatiquement ces modèles d'entrée à partir de la spécification B.

Références

Brottier, E., F. Fleurey, J. Steel, B. Baudry, et Y. L. Traon (2006). Metamodel-based test generation for model transformations : an algorithm and a tool. In *ISSRE*, pp. 85–94. IEEE Computer Society.

Brucker, A. D. et B. Wolff (2008). HOL-OCL : A Formal Proof Environment for UML/OCL. In J. L. Fiadeiro et P. Inverardi (Eds.), *FASE*, Volume 4961 of *LNCS*, pp. 97–100. Springer.

Facon, P. et R. Laleau (1995). Des spécifications informelles aux spécifications formelles : compilation ou interprétation ? In *Actes du 13ème congrès INFORSID*.

Jouault, F., F. Allilaire, J. Bézivin, et I. Kurtev (2008). ATL : A model transformation tool. *Science of Computer Programming 72*(1-2), 31–39.

Laleau, R. et F. Polack (2002). Coming and going from uml to b : A proposal to support traceability in rigorous is development. In *2nd International Conference of B and Z Users*, Volume 2272 of *LNCS*, pp. 517–534. Springer.

Legeard, B., F. Peureux, et M. Utting (2002). Automated boundary testing from Z and B. In *FME'02, Formal Methods Europe*, Volume 2391 of *LNCS*. Springer-Verlag.

Leuschel, M. et M. Butler (2003). ProB : A Model Checker for B. In *FME 2003 : Formal Methods Europe*, Volume 2805 of *LNCS*, pp. 855–874. Springer-Verlag.

Marcano-Kamenoff, R. et N. Levy (2001). Transformation d'annotations OCL en expressions B. In *Approches Formelles dans l'Assistance au Développement de Logiciels*.

Snook, C. et M. Butler (2006). UML-B : Formal modeling and design aided by UML. *ACM Transactions on Software Engineering and Methodology (TOSEM) 15*(1), 92–122.