xDSLs dirigés par les Modèles Formels

Tour d'horizon de l'outil Meeduse

Akram Idani, German Vega

Univ. Grenoble Alpes, Grenoble INP, CNRS, LIG
F-38000 Grenoble France

{nom.prenom}@univ-grenoble-alpes.fr

Résumé

Meeduse est un atelier de conception formelle de langages dédiés domaine (DSLs). Il permet de travailler sur la correction d'un DSL au moyen d'outils de raisonnements automatisés. La force de l'outil provient de ProB (un animateur et un model-checker de la méthode B) et d'EMF (un environnement IDM bien établi). Cet article passe en revue deux usages utiles de Meeduse : (*i*) exécution et déboggage, et (*ii*) transformation de modèles.

Mots clés : DSLs, Méthode B, Animation, Transformation de modèles.

1 Introduction

Plusieurs langages de modélisation dédiés domaine (DSLs), tels que les automates de D. Harel, les réseaux de Pétri ou les diagrammes de séquences d'UML décrivent les aspects comportementaux d'un système. Ils disposent ainsi d'une sémantique dite opérationnelle qui leur confère un caractère exécutable − d'où l'appellation xDSLs[1]. Les travaux IDM qui se sont intéressés à ce type de DSLs, ont donné lieu à des langages d'action (*e.g.* Kermeta, fUML) ainsi que des outils divers (*e.g.* xMOF, USE). Grâce à son "exécutabilité", un xDSL permet de simuler le comportement du système bien avant les activités de codage. L'objectif en est d'aborder les activités de vérification et validation (V&V) tôt dans le processus de développement. Dans [3] nous avons montré qu'une approche formelle peut s'avérer incontournable lors de la définition de la sémantique du xDSL. En effet, si celle-ci n'est pas rigoureusement définie, les activités de V&V peuvent être grandement impactées. Aussi, une approche formelle permet-elle de neutraliser les risques d'erreur émanant du langage et ré-hausser de fait son intérêt.

Malheureusement, les ateliers de conception de DSLs (appelés Language Workbenches − LWBs) ne supportent pas les méthodes formelles. Cette constatation repose sur les études de Kosar et al., [7] et Iung et al., [6] qui ont finement étudié et analysé les LWBs de 2006 jusqu'à 2019. La première étude (2006-2012) indique clairement qu'il y a un besoin urgent d'identifier les raisons et trouver les solutions. La deuxième étude (2012-2019) ne se réfère qu'au test comme activité de vérification et montre que sur les 59 outils analysés seuls 9 proposent un support au test. Dans nos travaux, nous avons cherché à donner une réponse pragmatique à cette problématique grâce à l'usage de la méthode B [1] et ses outils de preuve, d'animation et de model-checking. Nous avons développé Meeduse[2], un LWB qui permet d'instrumenter formellement un xDSL en décrivant sa sémantique en B et en fournissant l'outillage de vérification nécessaire. Une première preuve de concept a été présentée à AFADL'2018 − l'outil n'était qu'à l'état embryonnaire. Depuis, il a évolué, a gagné en maturité et a fait l'objet de plusieurs publications [2, 3, 4]. Il a aussi été appliqué avec succès sur plusieurs études de cas et a remporté deux prix au challenge TTC'19 (best verification et audience award). Dans cet article, nous passons en revue deux usages utiles de Meeduse : (*i*) exécution et déboggage, et (*ii*) transformation de modèles.

[1] xDSL: Executable Domain-Specific Language.

[2] `http://vasco.imag.fr/tools/meeduse/`

2 Exécution et déboggage

L'exécution et le déboggage – termes communément utilisés dans le jargon IDM – servent à observer le comportement d'un modèle dans le but d'identifier et comprendre des erreurs éventuelles. Dans le domaine des méthodes formelles on évoque souvent le terme animation interactive avec un objectif de vérification plus large que le déboggage ; cela peut couvrir la décomposition de prédicats, la résolution de contraintes à la volée, l'abstraction d'états, etc. Il existe deux approches pour assurer l'exécution et le déboggage d'un DSL : les approches interprétées, et les approches compilées. Dans le cadre des approches interprétées, l'état d'un modèle ainsi que les étapes d'exécution qui modifient cet état, sont définis au niveau du modèle au travers d'un langage spécifique. C'est donc le modèle lui-même qui est directement exécuté, analysé et déboggé. Cependant, les fonctionnalités de vérification de ces techniques sont très limitées. Dans le cadre des approches compilées, le modèle est traduit dans un formalisme cible exécutable dans le but de bénéficier des outils sous-jacents. La limitation majeure de ce deuxième type d'approches est que l'exécution se fait dans l'espace technologique du formalisme cible, sans aucune rétro-action au niveau du modèle source.

Meeduse met en œuvre une approche compilée, avec pour espace technologique cible celui de la méthode B. D'une part, il repose sur ProB [8] et bénéficie par conséquent de la richesse de ses fonctionnalités ; et d'autre part, il apporte la brique manquante car il réalise les rétro-actions requises au niveau du modèle source. L'approche de l'outil est présentée dans la Figure 1. A partir du méta-modèle du DSL, Meeduse génère une machine B fonctionnelle (appelée Static Semantics) qui représente ses aspects structurels. La sémantique opérationnelle du langage peut alors être définie dans une machine B à part (appelée Dynamic Semantics) qui inclut la machine fonctionnelle. Ce niveau sémantique permet de travailler sur la correction du DSL au moyen d'outils de preuve comme l'AtelierB.

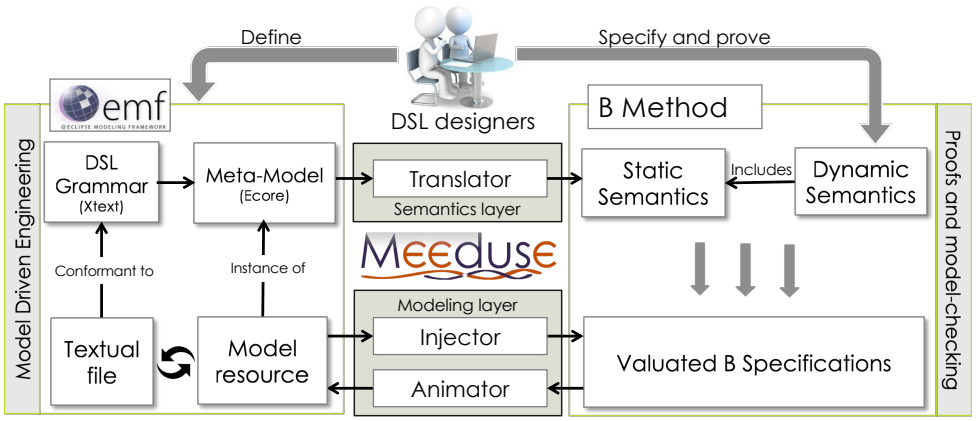

Figure 1: Concepts majeurs de Meeduse (Figure issue de [2])

Pour réaliser l'exécution de modèles, Meeduse embarque ProB et l'utilise de manière complètement transparente pour l'utilisateur final. A partir d'un modèle conforme au méta-modèle du DSL, Meeduse produit un raffinement de la spécification fonctionnelle en y introduisant les données du modèle. On obtient ainsi une spécification valuée et sémantiquement équivalente au modèle en entrée. La Figure 2 donne un exemple : (1) la machine *MeeduseTuto* est extraite à partir d'un méta-modèle contenant une classe **`Counter`** avec un attribut **`value`** de type entier ; et (2) le raffinement *MeeduseTuto_r* est extrait d'un modèle constitué d'une instance de cette classe où **`value`** est égal à 5. Dans cet exemple, on a considéré que seul l'attribut changera de valeur lors de l'animation.

Ayant cette spécification valuée, ProB entre en action pour calculer les opérations déclenchables dans l'état courant. A chaque pas de l'animation, Meeduse calcule la différence entre l'état avant et l'état après l'animation, et applique les modifications sous-jacentes au modèle source produisant ainsi son exécution.

```
MACHINE MeeduseTuto
SETS
   COUNTER
CONCRETE_CONSTANTS
   Counter
PROPERTIES
   Counter ∈ F (COUNTER)
CONCRETE_VARIABLES
   value
INVARIANT
   value ∈ Counter → NAT
INITIALISATION
   value :∈ Counter → NAT
END
```

```
REFINEMENT MeeduseTuto_r
REFINES MeeduseTuto
CONSTANTS
   COUNTER1
PROPERTIES
   COUNTER1 ∈ COUNTER ∧
   Counter = {COUNTER1}
INITIALISATION
   value := {(COUNTER1 ↦ 5)}
END
```

Figure 2: Modèles et méta-modèles dans Meeduse

3 Transformation de modèles

Dès lors qu'on parvient à exécuter un modèle EMF en entrée au moyen de ProB, il devient possible d'instrumenter en B plusieurs paradigmes de l'IDM, tels que la fusion, la composition, ou la transformation de modèles. En 2019, nous avons participé à la compétition TTC (Transformation Tool Contest) et nous avons montré la viabilité de l'outil dans ce cadre [5]. Meeduse a été utilisé non seulement pour montrer comment vérifier grâce au langage B, une transformation de modèles, mais aussi pour l'exécuter et produire les modèles cibles. L'outil n'a pas été conçu comme étant dédié à la transformation de modèles. Néanmoins, il offre la possibilité d'exécuter plusieurs modèles qui dépendent les uns des autres ; et c'est justement cette possibilité qui le rend bien adapté à la transformation de modèles.

La traduction d'un méta-modèle qui utilise des concepts d'un ou plusieurs autres méta-modèles donne lieu à une machine B unique qui inclut les structures de données de tous les méta-modèles. Ce même principe est appliqué au niveau modèle. En conséquence, pour instrumenter une transformation de modèle dans Meeduse, il suffit de s'inspirer du schéma de la Figure 3. Ayant un méta-modèle source $\mathrm{MM_{Source}}$ et un méta-modèle cible $\mathrm{MM_{Cible}}$, on introduit un méta-modèle $\mathrm{MM_{transfo}}$ qui utilise $\mathrm{MM_{Source}}$ et $\mathrm{MM_{Cible}}$. Étant donnée que la spécification B qui découle de $\mathrm{MM_{transfo}}$ couvre tous les concepts utiles (ceux de $\mathrm{MM_{Source}}$ et ceux de $\mathrm{MM_{Cible}}$), alors une transformation de modèle peut être spécifiée au travers d'opérations B dans cette même spécification. L'animation est réalisée sur la base d'une instance de $\mathrm{MM_{transfo}}$ qui désigne le modèle source à transformer. Quant au modèle cible, il est produit au fur et à mesure de l'animation des opérations B calculées par ProB.

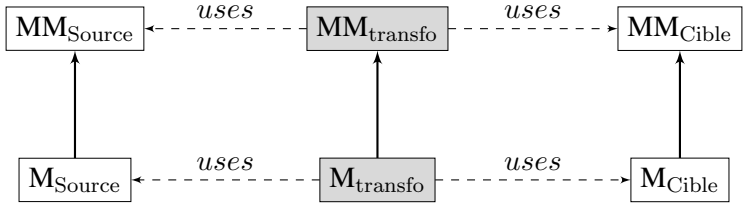

Figure 3: Architecture d'une transformation de modèles dans Meeduse

Cette approche a plusieurs bénéfices :

- Définir au moyen d'invariants B, les propriétés qu'une transformation doit préserver ;
- Spécifier les règles de transformation sous forme d'opérations et prouver leur correction vis-à-vis des propriétés invariantes ;

- Expliciter les concepts utiles à la trace de la transformation dans $\mathrm{MM}_{\mathrm{transfo}}$, et les inclure dans la spécification formelle.

4 Conclusion

Meeduse se présente aujourd'hui comme étant le seul atelier de conception formelle de DSLs. En effet, les investigations existantes [6, 7] pointent l'absence de méthodes formelles dans les LWBs. Notre approche repose sur la méthode B et ses outils, et couvre aussi bien les DSLs graphiques (conçus via Sirius ou basés sur UML comme dans le cadre de Papyrus) que les DSLs textuels (définis au moyen d'une grammaire Xtext). Outre cette contribution au monde des DSLs, Meeduse peut apporter des bénéfices immédiats aux nombreuses recherches, réalisées par la communauté B, pour la formalisation de notations semi-formelles. Nous pouvons citer, sans être exhaustifs, [10] où les auteurs proposent une formalisation de la sémantique opérationnelle des diagrammes de séquences d'UML2.x, ou encore [9] présentant une sémantique run-to-completion aux diagrammes d'états/transition, etc. Non seulement les sémantiques opérationnelles que ces travaux proposent pourraient être instrumentées et exécutées dans Meeduse mais aussi les transformations sous-jacentes. En effet, l'extraction de spécifications B à partir d'un modèle source n'est autre qu'une transformation d'un méta-modèle source vers un méta-modèle (ou une grammaire) de B. Grâce à Meeduse il devient possible d'écrire ces transformations en B lui-même, prouver leur correction, et les exécuter pour produire le modèle B escompté.

References

[1] J.-R. Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996.

[2] A. Idani. Meeduse: A tool to build and run proved DSLs. In *16th International Conference on Integrated Formal Methods (IFM)*, volume 12546 of *LNCS*, pages 349–367. Springer, 2020.

[3] A. Idani. Formal model-driven executable DSLs: Application to Petri-Nets. *International Journal on Innovations in Systems and Software Engineering (ISSE)*, 18(1), 2022.

[4] A. Idani. The B Method meets MDE: Survey, progress and future. In *16th Int. Conference on Research Challenges in Information Science (RCIS)*, volume 446, pages 495–512. Springer, 2022.

[5] A. Idani, G. Vega, and M. Leuschel. Applying formal reasoning to model transformation: The meeduse solution. In *Proceedings of the 12th Transformation Tool Contest, co-located with STAF'2019, Software Technologies: Applications and Foundations*, volume 2550, pages 33–44, 2019.

[6] A. Iung, J. Carbonell, L. Marchezan, E. M. Rodrigues, M. Bernardino, F. P. Basso, and B. Medeiros. Systematic mapping study on domain-specific language development tools. *Empirical Software Engineering*, 25(5):4205–4249, 2020.

[7] T. Kosar, S. Bohra, and M. Mernik. Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77–91, 2016.

[8] M. Leuschel and M. Butler. ProB: an automated analysis toolset for the B method. *Software Tools for Technology Transfer (STTT)*, 10(2):185–203, 2008.

[9] K. Morris, C. F. Snook, T. S. Hoang, G. C. Hulette, R. Armstrong, and M. J. Butler. Formal Verification of Run-to-Completion Style Statecharts Using Event-B. In *14th European Conference on Software Architecture*, volume 1269 of *CCIS*, pages 311–325. Springer, 2020.

[10] I. Mouakher, F. Dhaou, and J.-C. Attiogbé. Event-Based Semantics of UML 2.X Concurrent Sequence Diagrams for Formal Verification. *Journal of Computer Science and Technology*, 37(1):4–28, 2022.