Formal Support for Defining and Analysing Domain-Specific Modelling Languages

Dorel Lucanu and Vlad Rusu

March 11, 2011

Motivation

Domain-Specific Modelling Languages (DSMLs) are languages dedicated to modelling specific application areas. Recently, the design of DSMLs has become widely accessible to engineers trained in the basics of Model-Driven Engineering (MDE): one designs a *metamodel* for the language's abstract syntax; then, the language's operational semantics is expressed using *model transformations* over the metamodel. This approach is supported by tools such as KERMETA [1].

The democratisation of DSML design catalysed by MDE is likely to give birth to numerous languages. One can also reasonably expect that there shall be numerous errors in those languages. Indeed, getting a language right (especially its operational semantics) is hard, regardless of whether the language is defined in the modern MDE framework or in more traditional ones.

Formal approaches can benefit language designers by helping them avoid or detect errors by formal verification. But, in order to be accepted by nonexpert users, formal approaches have to be "hidden" behind a familiar design process such as the MDE-based one mentioned above.

Project Outline

We propose to build on our experience of using the K formal framework [2], which has been shown effective at defining real programming languages such as C[1] and Scheme[2]. K is based in rewriting is connected to verification tools: a model checker, and a verifier for the *matching logic* [3]. K's strength for language definition lies in its modularity: one can simply build a launguage by combining "language modules" containing specific language features (imperative, functional, object-oriented, multithereading, . . .). We are planning to take the following steps:

- (re)define in K an object-oriented language, possibly by adapting the definition of KOOL, an object-oriented language used for teaching purposes[3]. This language contains some of the basic ingredients used in MDE: classes and their attributes are the building blocks of UML class diagrams, which are the metamodels in MDE terminology; whereas instances of classes can be organised as UML object diagrams, which are models in MDE terminology;
- in parallel, define in K the subset of OCL[4] called *essential* OCL. OCL is a language for model navigation, query, and constraints, and essential OCL is its subset used in metamodelling;
- merge the two definitions to obtain an "object-oriented metalanguage", say, KOOL+OCL.

[1] http://fsl.cs.uiuc.edu/index.php/A_Formal_Semantics_of_C_with_Applications
[2] http://fsl.cs.uiuc.edu/index.php/K-Scheme
[3] http://fsl.cs.uiuc.edu/index.php/KOOL
[4] http://www.omg.org/spec/OCL/2.2/

In KOOL+OCL we shall be able to define a DSML syntax as a metamodel possibly extended with OCL invariants for expressing well-formedness constraints. Regarding a DSML's operational semantics, it will only be possible at this stage of the project to "define" it using the imperative constructs of the KOOL sublanguage. This means that we shall have the same functionality as KERMETA, with the additional advantage of being formal and connected to verification tools.

However, imperative programming is sometimes too low-level for defining operational semantics ; for instance, executing finite-state machines in KERMETA takes several dozen line[5].

The same definition only takes one rewrite rule in a rewrite-based framework; see, e.g., [4].

Hence, the next step of our project is to extend to our forthcoming K definition of the OCL language to that of a rewrite-rule based language for defining DSML semantics. The rewrite rules of that language shall be mapped to K conditional rewrite rules, whose conditions shall naturally be expressed in OCL. When this is done, the KOOL+OCL language will allow us to:

- write metamodels, in particular, for defining the abstract syntax of a given DSML;
- check the conformance of models with respect to metamodels enriched with well-formedness OCL constraints; in particular, such models can represent "programs" in a given DSML;
- write model transformations by combining imperative instructions and declarative rules. Such transformations can denote operational semantics, or translations between DSMLs. We expect that transformations denoting operational semantics will use mostly rewrite rules, and transformations denoting translations will use mostly imperative instructions;
- execute, simulate, and verify model transformations with respect to expected properties.

Among the expected properties, interesting ones *safety properties*, for model transformations encoding a DSML's operational semantics; and *simulation properties*, for expressing the fact that a translation between DSML expressed as a model transformation preserves operational semantics. Such properties could be expressed as predicates written in OCL, over single models (for safety), and over pairs of models (for simulation). To perform the verifications we are planning to use, possibly after adapting them, the K model checker and matching logic verifier.

References

[1] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In *MoDELS*, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2005.

[2] Grigore Roşu and Traian Florin Şerbănuţă. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.

[3] Grigore Roşu, Chucky Ellison, and Wolfram Schulte. Matching logic: An alternative to Hoare/Floyd logic. In Michael Johnson and Dusko Pavlovic, editors, *Proceedings of the 13th International Conference on Algebraic Methodology And Software Technology (AMAST '10)*, volume 6486, pages 142–162. LNCS, 2010.

[4] Vlad Rusu. Embedding domain-specific modelling languages into Maude specifications. *ACM Software Engineering Notes*. To appear in 2011. Extended version available at `http://researchers.lille.inria.fr/~rusu/SoSym/`.

[5] http://www.kermeta.org/docs/KerMeta-How-to-add-behavior-to-a-metamodel.pdf.