Bridging the Chasm between Executable Metamodeling and Models of Computation*

Benoît Combemale\textsuperscript{1}, Cécile Hardebolle\textsuperscript{2}, Christophe Jacquet\textsuperscript{2}, Frédéric Boulanger\textsuperscript{2}, and Benoit Baudry\textsuperscript{3}

\textsuperscript{1} University of Rennes 1, IRISA
\textsuperscript{2} Supélec E3S – Computer Science Department
\textsuperscript{3} Inria Rennes - Bretagne Atlantique

Abstract. The complete and executable definition of a Domain Specific Language (DSL) includes the specification of two essential facets: a model of the domain-specific concepts with actions and their semantics; and a scheduling model that orchestrates the actions of a domain-specific model. Metamodels can capture the former facet, while Models of Computation (MoCs) capture the latter facet. Unfortunately, theories and tools for metamodeling and MoCs have evolved independently, creating a cultural and technical chasm between the two communities. Consequently, there is currently no framework to explicitly model and compose both facets of a DSL. This paper introduces a new framework to combine a metamodel and a MoC in a modular fashion. This allows (i) the complete and executable definition of a DSL, (ii) the reuse of a given MoC for different domain-specific metamodels, and (iii) the use of different MoCs for a given metamodel, to account for variants of a DSL.

1 Introduction

Domain-specific languages (DSLs) offer a limited, dedicated set of concepts to domain experts to let them express their concerns about a system. Previous studies have shown that the limited expressiveness of DSLs, combined with dedicated tools, can increase the productivity in the construction of software-intensive systems, while reducing the number of errors \cite{1}. A recent study by Hutchinson \textit{et al.} has even demonstrated that DSLs are one of the main motors for an industrial adoption of model-driven engineering \cite{2}.

Defining a DSL completely and precisely is difficult, in particular when it comes to the formal definition of its semantics. However, Bryant \textit{et al.} \cite{3} point out that the formal definition of DSL semantics is the foundation for the major expected benefits of DSLs: the automatic generation of the DSL tooling (\textit{e.g.}, editor and compiler), the formal analysis of model behavior, or the rigorous composition of multiple concerns modeled with different languages.

\* This work has been partially supported by VaryMDE, a collaboration between Inria and Thales Research and Technology.
As described in the left of Figure 1, Harel et al. synthesizes the construction of a DSL as the definition of a triple: abstract syntax, concrete syntax and semantic domain \[ \text{AS}, \text{CS}, \text{SD} \]. This work focuses on the definition of the abstract syntax (AS), the semantic domain (SD) and the respective mapping between them \[ M_{\text{AS-SD}} \]. Several techniques can be used to define those three elements. This paper focuses on executable metamodeling techniques, which allow one to associate operational semantics to a metamodel. In this context, we argue that the formal definition of the semantic domain must rely on two essential assets: the semantics of domain-specific actions and the scheduling policy that orchestrates these actions. It is currently possible to capture the former in a metamodel and the latter in a Model of Computation (MoC), but the supporting tools and methods are such that it is very difficult to connect both to form a whole semantic domain (see right of Figure 1).

We propose to model domain-specific actions and MoCs in a modular and composable manner, resulting in a complete and executable definition of a DSL. We experiment this proposal by leveraging two state-of-the-art modeling frameworks developed in both communities: the Kermeta workbench \[ 5 \] that supports the investigation of innovative concepts for metamodeling, and the ModHel’X environment \[ 6 \] that supports the definition of MoCs. We foresee two major benefits for this composition: the ability to reuse a MoC in different DSLs, and the ability to reuse domain-specific actions with different MoCs to implement semantic variation points of a DSL. Saving the verification effort on MoCs and domain-specific actions also reduces the risk of errors when defining and validating new DSLs and their variants. We illustrate this approach and the reuse capacities through the actual composition of the fUML DSL with a sequential and then a concurrent version of the discrete event MoC.

The rest of the paper proceeds as follows: Section 2 introduces fUML, our case study throughout the paper. Then we describe how to design the domain-specific actions of a DSL and the MoC, respectively using Kermeta (Section 3) and ModHel’X (Section 4). We propose in Section 5 a tool-supported approach to combine them to implement the complete behavioral semantics of a DSL in a modular and reusable fashion. Finally, we present in Section 6 the application of our approach to vary the MoC of fUML. Section 7 presents related work, and Section 8 concludes and proposes directions of future work.