Model Transformations and Event-B for Specifying an Industrial DSL

Ulyana Tikhonova*
Technische Universiteit Eindhoven, The Netherlands

1 Introduction and Motivation

Domain-Specific Languages (DSLs) are a central concept of Model Driven Engineering (MDE). They are considered to be very effective in software development and are being widely adopted by industry nowadays. A DSL is a programming language specialized to a specific application domain. It captures domain knowledge and supports reuse of such knowledge via common domain notions and notation. In this way, the DSL raises the abstraction level of solving problems in the domain. On the other hand, a DSL implementation captures software design solutions, which implement domain concepts and their behavior. In this way, the DSL supports reuse of the design solutions and, thus, raises efficiency of the software development process.

A DSL is usually implemented as a translation from the domain concepts to the programming language of an execution platform. From a semantics point of view, the gap bridged by this translation can be quite wide. The DSL translation usually includes rather complicated design solutions and algorithms, which employ both high-level concepts of the DSL and low-level concepts of the execution platform. In the context of MDE, this complexity is coded in model transformations and code generators. As a result, in order to understand how a DSL program works, one needs to examine the source code generated from this DSL program. The complexity of the DSL translation, which is hidden from the DSL engineers, results in challenges when understanding, using, evolving, and verifying the DSL.

The complexity of the DSL translation can be revealed and reduced by providing its description on a higher abstraction level than model transformations, such as a thorough mathematical-based formal specification of the DSL semantics [Watt 1991]. In an industrial context it is not common to have a formal specification of the DSL semantics. However, having a formal specification of the DSL semantics can be beneficial if there are tools that support this specification. For example, the DSL semantics can be checked to be consistent and correct by analyzing its specification using verification and validation tools; DSL programs can be simulated and debugged by executing their specifications using animation tools.

The existing formalisms for specifying the semantics of general purpose languages (GPLs) (such as Action Semantics and Structural Operational Semantics) do not have practically applicable tool support. At the same time, there exists quite a number of tools that support formalisms for specifying behavior of hardware and software systems (such as B-method, Abstract State Machines). These formalisms are not designed for specifying the DSL semantics. In this work we investigate how a real-life industrial DSL can be specified using a specification formalism that has an extensive tool support; what are the practical benefits of having a formal specification of the DSL; and how the specification formalism can be adopted to realize these practical benefits by applying the MDE techniques.

2 Our Approach

To capture semantics and reveal the complexity of a mature real-life industrial DSL, we provided its formal specification using the Event-B formalism [Abrial 2010]. Event-B is supported by the Rodin platform. Rodin supports (1) checking correctness of Event-B specifications with the theorem provers and model checkers; (2) execution of Event-B specifications with the animators; (3) and wrapping Event-B specifications into graphical visualizations. However, applying Event-B for specifying the industrial DSL poses the following problems.

First, the DSL resides in two abstraction levels: DSL metamodel and DSL programs (models). The DSL is designed and implemented on the metamodel level, and it is used via instantiating DSL programs on the model level. While a generic specification of the DSL on the metamodel level can be created and analyzed once, Event-B specifications of many DSL programs need to be constructed and simulated by DSL users. We cannot expect DSL users to create Event-B specifications of their DSL programs and to check them themselves. Therefore, we applied the generic instantiation technique to concretize the generic specification of the DSL, defined on the metamodel level, into specifications of concrete DSL programs on the model level.

Second, the Event-B formalism is designed for specifying hardware or software systems, rather than DSL concepts and semantics. Therefore, specifying the DSL in Event-B is challenging and results in a big specification, which is hard to understand, maintain and verify. To tackle this problem we applied (de)composition of Event-B specifications: a system is decomposed into modules, which interact with each other; these modules are specified separately and then composed into one specification of the whole system.

We implemented all these techniques in model transformations from the DSL to Event-B. The model transformations automatically generate an Event-B specification for each concrete DSL program by composing and instantiating it from the generic specification modules. As a consequence, the DSL-to-Event-B transformations support the reuse of verification results and improve usability of Event-B and Rodin. The results of this work are described in detail in our paper [Tikhonova et al. 2013].

3 Future Work

For future work we aim to apply this approach to other DSLs. For this we need to generalize the model transformations by distinguishing reusable specification patterns. The patterns found while generalizing will form a library of reusable concepts for specifying semantics of DSLs. And the corresponding model transformations will realize a supporting framework for analyzing and executing the DSLs specifications.

References

