A Framework for Defining the Dynamic Semantics of DSLs

Ulyana Tikhonova
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
u.tikhonova@tue.nl

ABSTRACT

In this research abstract we describe our project on a common reference framework for defining domain specific languages (DSLs). The framework is meant for defining the dynamic semantics of DSLs and allows for mapping the DSL definition to the various platforms, such as verification, validation and simulation. The objectives of the project are to make a DSL dynamic semantics definition explicit and to use this definition for bridging technological diversity of various platforms, used in the DSLs development.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Languages*; D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Semantics*

General Terms

Design, Languages

Keywords

Domain Specific Languages, dynamic semantics, model transformations, validation and verification

1. RESEARCH PROBLEM

Domain-Specific Languages (DSLs) are considered to be very effective in software development and are being widely adopt-ed by industry nowadays. On one hand, a DSL captures domain knowledge and supports its reuse via domain notions and notation. It raises the abstraction level of solving problems in a domain. On the other hand, a DSL implementation captures software design solutions, which implement domain concepts and their behavior. The latter supports reuse of these design solutions and, thus, raises efficiency of the software development process.

In the context of Model Driven Engineering (MDE), the development of a DSL usually includes its design via metamodeling and its implementation via model transformations. A DSL metamodel captures language concepts, their compositional hierarchy, classification and cross references between them. Model transformations implement a semantics mapping of the DSL metamodel to its execution behavior(s). This means that dynamic (behavioral) semantics of the language is specified via a *semantic mapping* to a *semantic domain*.

In practice, a DSL implementation can include a number of semantic mappings which target different semantic domains in purpose to achieve diverse technological goals. These target semantic domains can be: source code for execution of the DSL programs, various formalisms for validation and/or verification of the DSL, graphical notations for visualization and/or simulation of the DSL programs. This means that several semantic mappings, which target different semantic domains, are defined for a single DSL (as shown in Figure 1). As a consequence, there is no guarantee that the different mappings for execution, validation, verification, etc. implement the same semantics. This potential lack of consistency causes maintenance problems and reduces the usefulness of targeting different semantic domains. For example, there is no confidence that a verified model is coherent to the corresponding executed source code.

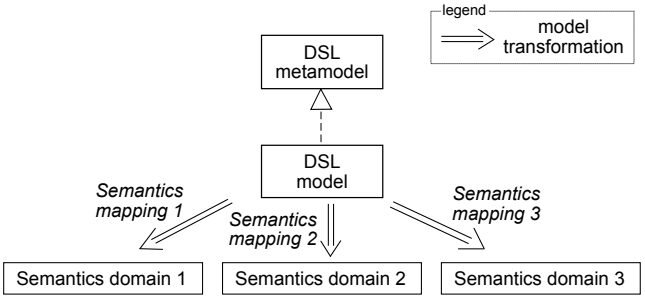

Figure 1: The semantics for a DSL comprised by multiple transformations

A way to overcome these issues is to use an intermediate step for the explicit definition of the dynamic semantics of a DSL. The DSL semantics is defined via a semantic mapping from the DSL metamodel to a *common semantic domain*, which is then used as a source for all other (predefined, implemented once and DSL independent) mappings (Figure 2). Thus, the incoherence among different DSL implementations is avoided, as the semantics is specified once and this specification is used as a source artifact for execution, validation, verification, simulation, etc. Another important advantage of this approach is an explicit definition of the dynamic seman-

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
Copyright 2013 ACM 978-1-4503-2237-9/13/08...$15.00
http://dx.doi.org/10.1145/2491411.2492404

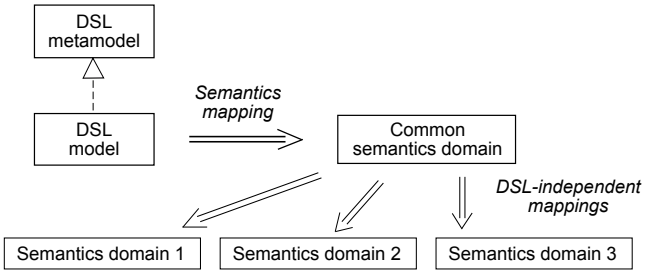

Figure 2: The semantics for a DSL is specified in a single transformation

tics of a DSL, which contributes to the design development, understanding and maintenance of the DSL.

The research problems that we target in this project are the following:

- investigation of the feasibility of defining a common semantic domain;
- bridging the technological diversity of existing semantic domains via this common semantic domain;
- development of a language and of supporting framework, that implement the common semantic domain and the bridging.

The scope of the project is restricted to the domain of system engineering. The expected outcome and a scientific contribution, that we aim at in this project, is a language, that allows for creation of new DSLs within the area of system engineering.

2. RELATED WORK

A lot of work has been done on specifying the dynamic (operational) semantics of general purpose languages (GPLs). There exist formalisms, e.g. Action Semantics [9], Structural Operational Semantics (SOS) [10], and various frameworks and tools, e.g. K-Maude,[1] Topcased,[2] Kermeta.[3] To make a DSL semantics definition executable using existing formalisms and tools, one need to manage a wide semantic gap between the high-level DSL concepts and low-level concepts of general purpose languages (GPLs). In our project we aim for decreasing this gap by introducing an intermediate language of common semantic domain.

An implementation of this idea is proposed in the work by Chen et.al [7]. They introduce high-level *semantic units* as an intermediate common language for defining the dynamic semantics of DSMLs (domain-specific modeling languages). To make it possible to use different semantic units for a definition, they explore a technique for composition of semantic units [6]. Semantic units are mapped to the Abstract State Machine (ASM) formalism[4] and the specified behavior is executed by means of the AsmL simulator tool. This approach is explored using the following case study: the semantics of SEFSM (simplified Extended Finite State Machine) is defined as the composition of two semantic units, FSM (Finite State Machine) and SDF (Synchronous Dataflow), which are specified in ASM.

Other areas, which can be related to our project, are *aspect oriented programming* and *design patterns*. Aspects and design patterns are the constructs of an intermediate common language used in software design. The techniques of composition of these constructs can be reused in our work.

3. APPROACH

To manage the research problems introduced in Section 1 we take a bottom-up iterative approach. We iterate over real-life (industrial) DSLs, define their dynamic semantics and generalize these definitions in the form of an intermediate common language. For each DSL, that we consider, we define its semantics on top of its existing infrastructure, trying to aim for the practical application of the definition in the industrial use cases (bottom-up approach). The DSLs we define in our case studies are provided by our industrial partner.

An outcome of this process is the COREF language (Common Reference Framework for Executable DSLs), that allows for definition of the dynamic semantics of DSLs in terms of the common semantic domain. The tool support of the COREF language is a framework, which implements mappings of the COREF language constructs to the various verification, validation, simulation, etc. languages and tools.

In order to design and review the COREF language, in every iteration (for each DSL case study) we perform the following steps:

1. define the dynamic semantics of a DSL using COREF language and/or some existing formalism;
2. while reuse of the COREF constructs review and adjust them if necessary;
3. if COREF language is not enough to define the DSL, extend it with the necessary constructs;
4. map the COREF constructs to the formalisms employed for the semantics definition, so that the supporting tools are applicable on the next iterations.

The core characteristic of this iterative process, that allows to assess its convergence, is an amount of reuse of the COREF constructs. Reuse of the COREF language implies reuse of the tools bridged by COREF and possibility to reapply use cases supported by these tools for different DSLs.

One of the challenges of the described approach is its starting point (first iteration), when the COREF language does not exist yet. Therefore, we employ an existing formalism for defining the DSL semantics and use this experience to generalize and design a prototype of COREF. There exists quite a number of formalisms for specifying behavior and tools for analyzing these specifications using different verification and validation techniques, e.g.: Z, B, the Abstract State Machines (ASM) language, Structural Operational Semantics (SOS). We chose the Event-B formalism [1].

Event-B is a specification language which employs formal mathematical notation for modeling software and/or hardware. The main reason we have chosen Event-B is that the Rodin platform [2] offers various supporting tools: editors, generator of so-called proof obligations, automatic provers, animators, etc. Using Rodin means that (1) we can easily

[1] `k-framework.org/index.php/Main_Page`
[2] `www.topcased.org`
[3] `www.kermeta.org`
[4] `research.microsoft.com/en-us/projects/asml`

prove consistency of the DSL semantics specifications with (interactive and automatic) provers, (2) we can find deadlocks and termination problems using model checkers, (3) we can use animators to validate the specified semantics with the help of domain experts, and (4) we can provide graphical visualization of the specification to help DSL users to understand the execution of their programs. All these tools are available in Rodin. Another benefit of Event-B is that it has an active community of users and developers.[5]

There exist a number of studies in which Event-B has been applied to define the semantics of a DSL. Ait-Sadoune and Ait-Ameur have used Rodin to describe the semantics of BPEL processes [4]. Hoang et al. have used Event-B and Rodin to automate analysis of Shadow refinement [8]. The overview of our implementation of the first iteration by applying Event-B and the results achieved and conclusions made in this experiment are described in Section 4.

4. PRELIMINARY RESULTS

In our first case study we specified the dynamic semantics of a mature real-life industrial DSL using the Event-B formalism. We employed Event-B as a target language for the semantics mapping and implemented this mapping using model-to-model transformation. The execution of the DSL-to-EventB transformation allows for automatic generation of Event-B specifications of the DSL programs, and thereby for their analysis using the broad spectrum of the tools provided by the Rodin platform: proving semantics coherency, model checking and animation. The goal of this case study was to investigate the benefits of having a formal specification of a DSL semantics and how these benefits can be practically achieved, applied and reused in an industrial context. The results and outcomes of this investigation are listed below.

4.1 Use Cases and User Roles

In respect to the use cases of the application of Event-B specification there can be distinguished two different roles: a DSL developer and a DSL user. These roles have different expertise and different needs. An overview of the use cases, that involve these two roles, is depicted in Figure 3 (on the right).

A DSL developer defines the dynamic semantics of the DSL to verify its design, to discover its properties and to validate that the design achieves the required behavior. The definition and its analysis are performed once for the language but require knowledge of proving and model checking techniques.

DSL users usually have a vague understanding of the DSL semantics and its implementation. They can get better understanding of the behavior of the DSL constructs and can get the insight in the DSL implementation via animation of its semantics specification. This type of analysis can be performed many times and should provide a user-friendly notation, as DSL users usually do not know formal methods techniques.

4.2 Compliance with the MOF Meta-levels

In the MDE context a DSL is defined via its *metamodel*, that captures language concepts, their compositional hierarchy, taxonomy and cross references between them. DSL programs instantiate this metamodel (Figure 3, on the left). These two standard MOF (Meta Object Facility) meta-levels can be implemented in Event-B using *generic instantiation* technique [3, 11, 5]. This technique allows for reuse of an Event-B specification by cloning a pattern (metamodel) specification and assigning concrete values to the abstract data types.

From the practical point of view the implementation of gener-ic instantiation poses a challenge to prove the conformance of a `DSL program specification` to the `DSL semantics specification` (Figure 3, on the left). An average DSL user is not expected to prove the conformance of her program specifications to the DSL specification using Rodin interactive provers, as it requires knowledge of propositional calculus and understanding of proof strategies. Therefore, this proof should be performed automatically. In practice the automated provers of Rodin fail to discharge all corresponding theorems due to the sizes of the programs. Therefore, we designed and implemented a technique how to evaluate the conformance using an Event-B animator, provided in Rodin.

4.3 Modularity of Semantics Definition

Capturing semantics of a DSL given in terms of its metamodel is rather complicated within an Event-B specification due to their different abstraction levels. To handle this complexity we employ modularity of the DSL semantics. We distinguish two types of semantics modularity: architectural and configurational. The *architectural modularity* of a DSL is determined by its implementation design: modules, components, layers of the existing DSL infrastructure can be specified separately in Event-B. These specifications are then composed together into a whole system specification using the *(de)composition* technique [3, 12]. Thus, the resulting specification is easier to create, understand and maintain.

The *configurational modularity* of a DSL is induced by different semantic features, that we distinguished in the DSL semantics. The semantic features can be also specified (and verified and validated) separately. The separate specification of the semantic features and their composition into a complete DSL specification were implemented in the transformation modules. Thus, the configurational modularity of the Event-B specification was achieved by applying the MDE techniques.

4.4 User-Friendly Visualization

To make it convenient for a DSL user to work with Event-B we developed a graphical visualization of the DSL specification using BMotion Studio plug-in.[6] This visualization runs together with the ProB animator and provides a GUI (graphical user interface) for a machine being animated. By experimenting with a DSL program in this GUI a user can get better understanding of the DSL design and improve efficiency of her programs.

4.5 Implementation

The resulting DSL-to-EventB transformation was implemented using the *Operational QVT* (Query/View/Transformation) language in the Eclipse environment.[7] The input for the transformation is provided directly by the DSL implementation software, which is compatible with the EMF (Eclipse Modeling Framework). As a target metamodel for the trans-

[5] `www.event-b.org`

[6] `www.stups.uni-duesseldorf.de/bmotionstudio/index.php/BMotion_Studio`

[7] `wiki.eclipse.org/M2M/Operational_QVT_Language_(QVTO)`

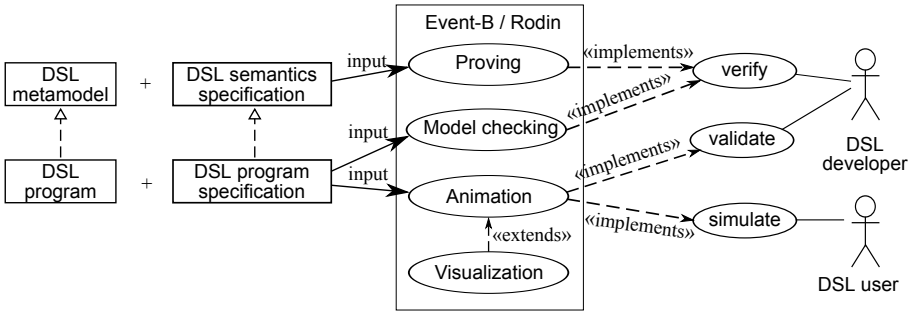

Figure 3: Use cases of the application of Event-B specification

formation we use Event-B Ecore implementation provided by the EMF framework for Event-B.[8]

4.6 Conclusion

We have shown (1) that generic instantiation allows to put Event-B specifications in compliance with the Meta Object Facility (MOF) meta-levels, (2) that modularity of dynamic semantics supports simple and clear model-to-model transformations and allows for the reuse of already verified specification parts by means of the composition approach, and (3) that automated proving can be enhanced with generation and evaluation of auxiliary Event-B components. As a result, model-to-model transformation makes Event-B and its supporting tools applicable in the industrial use cases for two very different roles: a DSL developer and a DSL user. All listed above observations and implemented techniques are described in detail in the forthcoming paper [13].

We have observed, that the abstraction level provided by Event-B is not enough for the COREF language. As a next step we plan to generalize the DSL-to-EventB transformation by distinguishing reusable patterns in the Event-B specifications of the DSL. The patterns found in generalization will form a prototype of the COREF language. This will allow both for reuse of the developed techniques and for reuse of already verified and visualized pieces of specification.

5. ACKNOWLEDGEMENTS

I am very grateful to my supervisor Mark van den Brand (Eindhoven University of Technology) for the guidance and support he provides to me.

6. REFERENCES

[1] J.-R. Abrial. *Modeling in Event-B: system and software engineering*, volume 1. Cambridge Univ Pr, 2010.

[2] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(6):447–466, 2010.

[3] J.-R. Abrial and S. Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundam. Inform.*, 77(1-2):1–28, 2007.

[4] I. Ait-Sadoune and Y. Ait-Ameur. Stepwise Design of BPEL Web Services Compositions: An Event-B Refinement Based Approach. In R. Lee, O. Ormandjieva, A. Abran, and C. Constantinides, editors, *Software Engineering Research, Management and Applications 2010*, pages 51–68. Springer Berlin / Heidelberg, 2010.

[5] D. A. Basin, A. Fürst, T. S. Hoang, K. Miyazaki, and N. Sato. Abstract Data Types in Event-B – An Application of Generic Instantiation. In *Workshop on the experience of and advances in developing dependable systems in Event-B*, CoRR, 2012.

[6] K. Chen, J. Porter, J. Sztipanovits, and S. Neema. Compositional Specification Of Behavioral Semantics For Domain-Specific Modeling Languages. *Int. J. Semantic Computing*, 3:31–56, 2009.

[7] K. Chen, J. Sztipanovits, S. Abdelwalhed, and E. Jackson. Semantic Anchoring with Model Transformations. *European Conference on Model Driven Architecture - Foundations and Applications*, pages 115–129, 2005.

[8] T. Hoang, A. McIver, L. Meinicke, C. Morgan, A. Sloane, and E. Susatyo. Abstractions of non-interference security: probabilistic versus possibilistic. *Formal Aspects of Computing*, pages 1–26, 2012.

[9] P. Mosses. Theory and practice of action semantics. In W. Penczek and A. Szalas, editors, *Mathematical Foundations of Computer Science 1996*, volume 1113 of *Lecture Notes in Computer Science*, pages 37–61. Springer Berlin / Heidelberg, 1996.

[10] G. D. Plotkin. A Structural Approach to Operational Semantics. *The Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[11] R. Silva and M. Butler. Supporting Reuse of Event-B Developments through Generic Instantiation. In K. Breitman and A. Cavalcanti, editors, *11th International Conference on Formal Engineering Methods, ICFEM*, volume 5885 of *Lecture Notes in Computer Science*, pages 466–484. Springer, 2009.

[12] R. Silva and M. Butler. Shared Event Composition/Decomposition in Event-B. In B. K. Aichernig, F. S. de Boer, and M. M. Bonsangue, editors, *Formal Methods for Components and Objects (FMCO)*, pages 122–141. Springer, 2010.

[13] U. Tikhonova, M. Manders, M. G. J. van den Brand, S. Andova, and T. Verhoeff. Applying Model Transformation and Event-B for Specifying an Industrial DSL. 2013.

[8] `wiki.event-b.org/index.php/EMF_framework_for_Event-B`