Semantic Framework for DSLs

Zekai Demirezen
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294-1170
zekzek@uab.edu

ABSTRACT
Domain-Specific Languages (DSLs) enable domain experts to participate in software development tasks and to specify their own programs using domain abstractions. To define programs using domain concepts, rather than programming language concepts, model-based syntax and semantic specification techniques may offer advantages over current approaches. The purpose of the research described in this paper is to provide a semantic framework that can be used visually by DSL designers, yet has formal underpinnings such that interoperation with verification tools is possible to realize model checking tasks. This research is focused on a visual technique based on activity diagrams and graph transformation rules to define the semantics of DSLs.

Categories and Subject Descriptors D.2.4 [**Software Engineering**]: Software/Program Verification– *Model checking.* D.3.1 [**Programming Languages**]: Formal Definitions and Theory – *Semantics, Syntax.*

General Terms Algorithms, Languages, Verification

Keywords domain-specific languages, operational semantics, graph transformation systems, activity diagram, model checking

1. INTRODUCTION
Model-Driven Engineering (MDE) has been shown to increase productivity and reduce development costs [1]. The concepts advocated by MDE focus on abstractions tied to a specific domain that provides tailored modeling languages for domain experts. Domain-Specific Languages (DSLs), used within the MDE context, enable end-users who are domain experts to participate in software development tasks and to specify their own programs using domain concepts in the problem space, rather than programming language concepts in the technical solution space. However, there remain several challenges that drive new research in DSLs. For example, simulation, code generation, model checking and different kinds of analysis require a precise definition of the semantics of a DSL. Most modeling toolsets do not allow the semantics of DSLs to be defined in a way that would support such desirable analysis and generation tasks.

DSLs, like any other language, consist of definitions that specify the abstract syntax, concrete syntax, static semantics and dynamic semantics of the language. Specification of abstract syntax includes the concepts that are represented in the language, and the relationships between those concepts. Concrete syntax definition provides a mapping between meta-elements and their textual or graphical representations. Well-formedness rules, which represent the static semantics of a language, can be defined to check model consistency. The runtime behavior of each syntactical meta-element defined in the DSL represents the dynamic semantics of the language, which is often more challenging to specify.

The research described in this paper represents an investigation into the design of a semantic framework that enables DSL designers to define semantic specifications using visual models. The proposed framework also addresses issues of model verification and model analysis by defining the verification tasks that are specific to a particular domain.

2. RELATED WORK
Current platforms and toolsets that have provided a means for specifying the behavioral semantics of a modeling language often rely on some formalism based on operational semantics. A common approach is to map the metamodel concepts of a DSL to a mature and well-known existing target semantic domain (e.g., Abstract State Machines (ASM) [2], and Petri Nets [3]). In this context, Agrawal et al [4] and Chen et al [5] utilize what they call a semantic anchoring technique to map abstract syntax models to existing ASM semantic domains in the GME platform. Ruscio et al [6] propose a similar technique, except the ASM mapping is integrated within the AMMA platform. In these approaches, the dynamic behavior of a specific DSL element is modeled as a sequence of ASM state transitions. Although these kinds of definitions enable the adoption of model checking and simulation activities using the target semantic domain, it is challenging for DSL designers to use such approaches (because of unfamiliar formalisms in the target model concepts). Muller et al [7] extended an abstract syntax metalayer with an action language to weave a semantic definition within a metamodel. However, the necessity of defining the behavior of each concept in an imperative way results in code that is written in the style of a general-purpose programming language. Engels [8] provides operational semantics of diagrams by means of collaboration and graph transformations. Knapp [9] uses temporal logic; Overgaard [10] advocates the π-calculus to define semantics. Although the formal structures of these related works are suitable for usage with model verification and simulation tools, the specific approaches require expertise in notations and formalisms that are not generally within the skillsets of most designers.

3. RESEARCH GOALS
The research described in this paper proposes a semantic framework that can be used by DSL designers, yet has a formal foundation that will permit interoperation with model verification tools. A key research question addresses the feasibility of designing a general visual language that can be used to define the dynamic semantics of a modeling language, which can interoperate with analysis tools to

allow designers to verify the correctness of models within domain-specific verification tasks.

4. APPROACH AND METHODOLOGY

Existing approaches for defining the formal semantics of programming languages can be used to specify the semantics of DSLs. However, a critical point of this proposed work is an investigation of the benefits that visual models offer to DSL designers in terms of specifying semantics of a new language.

A first step of this project is to investigate a technique for representing state transitions. Behavioral semantics of DSLs can be represented by a sequence of state transition rules. This approach divides all semantic concerns into discrete states and transition relations. In particular, in-place model transformations [11] represent an approach for designing state transitions. One of the main characteristics of in-place model transformation is that target and source models are always instances of the same metamodel. Graph grammars [3] provide visual rules to specify in-place transformations based on precondition actions and postcondition steps. The notation proposed by AGG [12] to model graph transformations can be used to define these rules visually. AGG is a rule-based visual language supporting an algebraic approach to graph transformation. Available tools associated with AGG (e.g., Graph Transformation Engine, Graph Pattern Matching, and AGG's analysis techniques for consistency checking) make AGG an attractive candidate for the definition of state translation rules.

Although each AGG transformation shows one of the state transitions of the runtime behavior, to give complete semantics of DSLs, sequences of state changes should be defined. These sequence definitions control what state transitions are to be fired and in what order. Therefore, the second step of this project is to develop a technique for specification of state transition sequences. An activity diagram is an appropriate state machine to define these transition sequences. It enables the design of simple and compound states, branches, forks, and joins. In the proposed framework, each state transition will be mapped with an activity in the activity diagram. Therefore, each activity diagram will depict state transition configurations.

The final step of this project is to facilitate model checking functions by interoperating existing model checking tools with the syntax and semantics of a new modeling language. To enable this capability, an instance model specified in a metamodel must be converted into the formalism expected by an underlying model checking tool. Next, the properties that the model must satisfy need to be stated by a logical formalism expressed in the format expected by the verification tool. For example, Baresi et al [13] demonstrate how the Alloy [14] tools can be used in graph transformation systems. Graph transformations (described using AGG) can be transformed into an Alloy Model [13]. Therefore, DSL programs and verification tasks, which are defined by AGG at the domain level, can be transformed into the lower level model needed by Alloy.

Evaluation. A research question will be considered in the project evaluation by using several unique domains that each have a representative DSL. Specification complexities of state transition, sequence of transitions, and verification task definitions, will be checked in detail for each domain. An additional step will be needed to compare the proposed semantic framework with other operational semantics techniques. Visual modeling, comprehensibility, ease of use, amount of time to design a new DSL, and compatibility with verification tools will be used as comparison and evaluation criteria.

5. ACKNOWLEDGMENTS

This work is supported in part by an NSF CAREER award (CCF-0643725).

6. REFERENCES

[1] Vallecillo, A. A Journey Through the Secret Life of Models, In Dagstuhl Seminar: Model Engineering of Complex Systems, 2008.

[2] Borger, E. High Level System Design and Analysis using Abstract State Machines, In FM-Trends 98, Vol. 1641, LNCS, Springer, 1999, pp. 1-43.

[3] de Lara, J., Vangheluwe, H. Translating model simulators to analysis models. In Fundamental Approaches to Software Engineering, Vol. 4961, Springer LNCS, 2008, pp. 77-92.

[4] Agrawal, A., Karsai, and G., Ledeczi, A. An end-to-end domain-driven software development framework. In Object-Oriented Programming, Systems, Languages, and Applications, Anaheim, CA, 2003, pp. 8–15.

[5] Chen, K., Sztipanovits, J., Abdelwalhed, S., and Jackson, E. Semantic Anchoring with Model Transformations. In Model Driven Architecture Foundations and Applications: First European Conference, Springer, 2005, pp. 115–129.

[6] di Ruscio, D., Jouault, F., Kurtev, I., Bezivin, J., and Pierantonio, A. Extending AMMA for Supporting Dynamic Semantics Specifications of DSLs, Technical Report 06.02, Laboratoire d'Informatique de Nantes-Atlantique, Nantes, France, April 2006.

[7] Muller, P.-A., Fleurey, F., and Jézéquel, J.-M. Weaving Executability into Object-Oriented Meta-Languages, In Proceedings of MODELS/UML, 2005, pp.264-278.

[8] Engels, G., Hausmann, J. H., Heckel, R., and Sauer, S. Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML. In The Unified Modeling Language: Advancing the Standard. York, UK, Vol. 1939, LNCS. Springer, 2000, pp. 323–337.

[9] Knapp, A. A Formal Semantics of UML Interactions, In Proceedings of the UML – Beyond the Standard, Vol. 1723, LNCS. Springer, 1999, pp. 116–130.

[10] Overgaard,G. Formal Specification of Object-Oriented Meta-Modelling, In Fundamental Approaches to Software Engineering, Berlin, Germany, Vol. 1783, LNCS, Springer, 2000, pp. 193–207.

[11] Czarnecki, K., Helsen, S.: Feature-based Survey of Model Transformation Approaches. In: IBM Systems Journal, Vol. 45, Issue 3, July, 2006, pp. 621-645.

[12] Beyer, M. AGG1.0 – Tutorial, Technical University of Berlin, Department of Computer Science, 1992.

[13] Baresi, L., and Spoletini, P. On the Use of Alloy to Analyze Graph Transformation Systems. In Proceedings of the International Conference on Graph Transformation, Vol. 4178, LNCS, Springer, 2006, pp. 306–320.

[14] Jackson, D., Shlyakhter, I., and Sridharan, M. A Micromodularity Mechanism, In European Software Engineering Conference, Vienna, Austria, 2001, pp. 62–73.