Generalized Sketches for Model-driven Development

Adrian Rutle

Bergen University College, p.b. 7030, 5020 Bergen, aru@hib.no

Abstract. The diversity and heterogeneity of modeling languages make the needs for formal model specifications and automatic model integration and transformation mechanisms more relevant than ever. These mechanisms are the corner stones in Model-driven Development, which is a natural evolutionary step in raising the abstraction level of programming languages. In this talk, we propose a generic formalism, Generalized Sketches, for specifying modeling languages and their transformations.

1 Model-driven Development (MDD)

MDD is a software development process in which modeling, transformations and automatization of model transformations are important issues. In MDD, an application is built by working at the model level. The process starts by specifying an abstract and formal (diagrammatic) model which is independent of the application's platform, i.e. the implementation technology, design, programming language ... etc. This kind of model is referred to as Platform Independent Model (PIM) [1]. In PIM, one can specify the business logic of the application without restriction to a special system design.

The next step in MDD consists of specifying a transformation for transforming the PIM into a (set of) Platform Specific Model(s) (PSM). PSMs are also formal models, but they are restricted to a specific implementation technology and programming language; like OO-design or relational schemes.

The last step considers transforming the PSMs to application code. There are many tools that support this step, but existing tools only allow developers to choose among a predefined set of transformation definitions, for example, transforming UML class diagrams to Java, C++, SQL code ...etc.

The challenge in MDD is in finding a formalism for specifying the models and choosing mechanisms for definition of (and automatically execution of) transformations between those models.

2 Generalized Sketches (GS)

GS is a graph-based specification format that borrows its main ideas from both categorical and first-order logic, and adapts them to software engineering needs [2]. The claim behind GS is that any diagrammatic specification technique in software engineering can be seen as a specific instance of the GS specification pattern. GS is a pattern, i.e. generic, in the sense that we can instantiate this pattern by a signature that corresponds to a specific specification technique, like UML class diagrams, ER diagrams or XML. A signature is an abstract structure consisting of a collection (or a graph) of predicate symbols with a mapping that assigns a shape (or an arity) to each predicate symbol. A $\Sigma - sketch$ is a graph with a set of diagrams labeled with predicates from the signature Σ

[3]. Diagrams drawn using a specific specification technique, will appear as a (possibly ambiguous) visualization of a sketch which is parameterized by the corresponding signature Σ.

Thus we claim that GS can be used as a standard notation for representing both the syntax and the semantics of diagrammatic specification languages, as the syntax and in most cases also the semantics of GS is mathematically well-defined and unambiguous.

3 Generalized Sketches and MDD

As mentioned above, GS can be used to specify modeling languages and transformations between them. Since GS is a generic specification format, it can be used to specify PIMs, PSMs and the transformations between them. Also by regarding programming languages as modeling languages, one can use the following generic mechanism for transformation between PIMs, PSMs and code.

Models M that are specified by a given modeling language ML must conform to the metamodel MM of ML. MM is considered as a specification technique which corresponds to a (graphical) signature Σ_{ML} in GS, i.e. $MM \cong \Sigma_{ML}$ [4]. Having abstract definitions of signatures (or metamodels,) say MM_1 and MM_2, a relationship or transformation between them can be defined as a morphism $rel : MM_2 \rightarrow MM_1$ from the target metamodel MM_2 to the source metamodel MM_1 (see the figure.) Then any model M_1 conforming to (i.e. which is an instance of) MM_1 can be transformed automatically to a model M_2 conforming to MM_2 by computing the pullback (M_2, i_2, rel^*) of the sink (MM_1, i_1, rel) [5]. The underlying category of the models and metamodels is GRAPH and thus the pullback exists. The application of the pullback construction opens for automatization of the transformation as it's needed by MDD [6].

$$\begin{array}{ccc} M_1 & \xleftarrow{rel^*} & M_2 \\ {\scriptstyle i_1}\downarrow & & \downarrow{\scriptstyle i_2} \\ MM_1 & \xleftarrow{rel} & MM_2 \end{array}$$

4 Tools

By developing tools that support GS as a generic pattern for specifying and developing diagrammatic specification techniques we can prove and exploit the practical value of GS in all aspects of (meta)modeling and MDD from transformation and integration to decomposition and code-generation.

The transformation definition in most existing transformation tools is composed of a set of transformation rules that define how elements or constructs from a source model can be transformed to a target model [1]. These rules are restricted to element-wise transformations and in the best case to binary relations between elements. While in the GS methodology, a transformation is a morphism capable of transforming structures and relationships spanning over many (meta)model elements. Other drawbacks of the methodologies used today are that transformation rules are only based on heuristics –they must be defined and hard-coded for each two metamodels, and you are not guarantied the existence of compositionality and associativity between rules.

Our tool will be used to design signatures corresponding to existing specification techniques, like UML class diagrams and ER diagrams. Designing signatures for existing modeling languages (the so-called "sketching" or "formalizing" in [3]) involves exhausting exploration of the syntax and

semantics of those languages to find the adequate set of predicates; like total, partial, jointly mono, disjoint-cover ... etc needed to express all properties that can be expressed by them. Then, preferred graphical notations for the predicates can be chosen. (This step corresponds to the specification of MM1 and MM2 in the figure.) Diagrams, i.e. visualizations of sketches, can be drawn using the signatures/specification techniques (this step corresponds to the specification of M1 and M2 in the figure.) The tool will also support definition of transformation between (meta)models and automatic construction of pullback.

References

1. Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
2. Zinovy Diskin and Uwe Wolter. A Diagrammatic Logic for Object-Oriented Visual Modeling. In *ACCAT 2007: 2^{nd} Workshop on Applied and Computational Category Theory*, volume 203 of *ENTCS*, pages 19–41, Amsterdam, The Netherlands, 2008. Elsevier Science Publishers B. V.
3. Zinovy Diskin. Generalized sketches as an algebraic graph-based framework for semantic modeling and database design. Technical Report 9701, University of Latvia, Riga, Latvia, August 1997.
4. Uwe Wolter and Zinovy Diskin. The Next Hundred Diagrammatic Specification Techniques – An Introduction to Generalized Sketches. Technical Report 358, Department of Informatics, University of Bergen, Norway, July 2007.
5. Zinovy Diskin. Model Transformation via Pull-backs: Algebra vs. Heuristics. Technical Report 521, School of Computing, Queen's University, Kingston, Canada, September 2006.
6. Zinovy Diskin and Jürgen Dingel. A metamodel Independent Framework for Model Transformation: Towards Generic Model Management Patterns in Reverse Engineering. In *ATEM 2006: 3^{rd} International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering*, 2006.