The Formal Specification Column

by

Hartmut Ehrig

Technical University of Berlin, Department of Computer Science
Franklinstraße 28/29, D-10587 Berlin, Germany
ehrig@cs.tu-berlin.de

Model Transformations should be Functors

(May 3, 2007)

Don Batory[1], Oscar Diaz[2], Hartmut Ehrig[3],
Claudia Ermel[3], Ulrike Prange[3] and Gabriele Taentzer[4]

[1]Dept of Computer Sciences
University of Texas at Austin, USA
Austin, USA
batory@cs.utexas.edu

[2] Dept of Computer Sciences
Univ. of Basque Country
San Sebastian, Spain
oscar.diaz@ehu.es

[3]Fac. Electr. Eng. and Comp. Sci.
Technical University of Berlin
Berlin, Germany
ehrig,lieske,uprange@cs.tu-berlin.de

[4] Dept. of Maths and Comp. Science
University of Marburg
Marburg, Germany
taentzer@mathematik.uni-marburg.de

Abstract

The concept of model transformations is of increasing importance in different areas of Computer Science, but up to now, there is a lack of common understanding concerning the mathematical and practical point of view.

In this paper, we discuss some of the different aspects. Especially interesting is the new proposal of the POPL'07 keynote speaker Don Batory claiming that model transformations should be functors. This claim is compared with different mathematical concepts of model transformation.

Introduction

For about 35 years, various specification techniques have been developed in order to model different kinds of systems in computer science and other areas. The concept of models and model-driven development (MDD) has become very important for software development. Especially the use of UML is wide-spread in practice as visual modelling technique [10]. In order to relate different components of a system represented by different models, it is most important to have suitable model transformation techniques. Transformation of models is the key technology of MDD and serves a variety of purposes, including the refinements of models, their mapping to implementations, consistency management, and evolution. The most prominent approach to model transformation is the de facto standard *Queries, Views and Transformations* (QVT) [11] by the OMG. Model transformations have been classified in [3, 9] in *endogenous* and *exogenous* ones. While endogenous model transformations run within one modelling language, exogenous ones are used to translate models between different languages.

In this paper, we want to discuss the concept of model transformations from a mathematical and a practical point of view and analyze how far the expectations from both sides can be fulfilled in a suitable general framework. In both cases, we start with a very basic point of view to be followed by a more elaborate one.

1 A Basic Mathematical Point of View

In [2], Bezivin claims that "everything is a model". From this point of view, we aim to capture all kinds of models, including programs, by considering model classes as sets/classes of models of the same kind, such that each model becomes an element of the corresponding model class. Hence, we define a model transformation MT from model class M_1 to class M_2 as a function $MT : M_1 \to M_2$. Due to the properties of functions in set theory, this means that for each model (element) $m_1 \in M_1$ there is a well-defined model $m_2 = MT(m_1) \in M_2$. If necessary, the functional property of MT can be relaxed to be partial or nondeterministic, or even more general, that MT is a relation between M_1 and M_2.

This purely mathematical point of view allows us to apply the basic notions and results of sets and functions to model classes and model transformations, like union and products of sets and composition of functions or relations. Yet, in many cases, a semantic compatibility between models before and after the transformation is desired. Our basic mathematical point of view is useful on an abstract level, but it does not capture the behaviour of models. This means that model transformation on this level cannot be expected to preserve the behaviour of corresponding models, or to establish at least a relationship between the behaviour of corresponding models. This problem will be discussed from a more elaborate mathematical point of view in Section 4.

2 A Basic Practical Point of View

From a practical point of view, models are documents belonging to a domain-specific language which is implemented in a suitable way. This includes all kinds of documents on the level of requirements and design specifications, programs on the level of programming languages, and also code on the level of machine languages. From this point of view, a model transformation is given by a tool T which transforms documents in model class M_1 to documents in class M_2, short $T : M_1 \to M_2$. Well-known tools of this kind are compilers from programming to machine languages, code generators from modelling to programming languages, and also translation tools between different visual or textual modelling languages. In addition to these cases of exogenous model transformations, tools for program or model refactoring are examples of endogenous model transformations. From an abstract point of view, a tool $T : M_1 \to M_2$ is usually expected to define a function $MT : M_1 \to M_2$ in the sense of Section 1. But in a variety of cases, the tool actually defines only a relation from M_1 to M_2, because of non-termination or in some cases, e.g. for refactoring, even nondeterminism of the tool applied to specific input documents. Moreover, it is expected that the tool $T : M_1 \to M_2$ transforms a document $m_1 \in M_1$ to a document $m_2 = T(m_1) \in M_2$ with a well-defined semantics. In the case that both source and target languages have a semantics, we expect that the semantics of m_1 can be related to the semantics of m_2 in a suitable way. But except for some examples in the area of compiler correctness, which, of course, requires a formal semantic of the source and target languages, this expectation is not verified for most of the tools in practice.

3 An Elaborate Practical Point of View

A more elaborate practical point of view concerning model transformation is taken by Batory et al. in [12]. Their approach of *Feature Oriented Model Driven Development* (FOMDD) is a combination of Model Driven Development (MDD) and Feature Oriented Programming (FOP). [12] presents a case study of FOMDD, a product-line of portlets which are building blocks of web portals. A portlet is specified as a set of models which are refined and transformed, leading to an implementation. Combining model transformation and model refinement exposes a fundamental commuting relationship that should – as claimed in [12] – arise in all examples of FOMDD: the transformation of a refined model equals the refinement of the transformed models. This is expressed by the following commuting diagram in [12], where M_1, M_2, D_1 and D_2 are model classes, $\triangle M$ and $\triangle D$ are model refinements, where $m_2 = \triangle M(m_1)$ can be considered as a refined model composed of m_1 with some $\triangle$, and $f : (M_1 \cup M_2) \to (D_1 \cup D_2)$ is a model transformation.

$$\begin{tikzcd} M_1 \arrow[r, "\triangle M"] \arrow[d, "f"'] & M_2 \arrow[d, "f"] \\ D_1 \arrow[r, "\triangle D"] & D_2 \end{tikzcd}$$

In the case study of [12], it turned out that for a specific instance $m_1 \in M_1$ both paths in this diagram could be implemented, leading to $d_2 = f(\triangle M(m_1))$ and to $d'_2 = \triangle D(f(m_1))$. In the beginning, both implementations were unequal, i.e. $d_2 \neq d'_2$, but after removing some bugs in the implementation of $\triangle M, \triangle D$ and f, they were equal, i.e. $d_2 = d'_2$. The main claim of [12] is that, as a general requirement within FOMDD, the commutativity of the diagram should be valid. This general requirement allows to validate the correctness of the abstractions, tools and portlet specifications, as well as optimize portlet synthesis. In [1], Batory has generalized this idea, on the one hand, to the synthesis of all kinds of (software) products, and, on the other hand, to a categorical framework in the following sense: The function $\triangle M : M_1 \to M_2$ is replaced by a category $\mathbf{M}$ with objects $M_1 \cup M_2$ and generating morphisms $m_1 \xrightarrow{\triangle M} m_2$ for each $m_1 \in M_1$ with $\triangle M(m_1) = m_2$. Commutativity of the diagram above means now that the function f becomes a functor $F : \mathbf{M} \to \mathbf{D}$ with $F(m_1 \xrightarrow{\triangle M} m_2) = d_1 \xrightarrow{\triangle D} d_2$, where $d_1 = F(m_1)$, $d_2 = F(m_2)$ and $\triangle D = F(\triangle M)$. His general claim is that model classes M and D should be replaced by model categories $\mathbf{M}$ and $\mathbf{D}$, where the objects are models, and the morphisms are suitable model refinements (called *features*). Moreover, model transformations should be functors

$$F : \mathbf{M} \to \mathbf{D},$$

where the functor properties are important requirements for the compatibility of model transformations and refinements. Actually, he claims that some compilers from Java to bytecode and javadoc representation, respectively, can be considered as functors in this sense.

4 An Elaborate Mathematical Point of View

In Section 1, we have pointed out that on a very basic abstract level, model transformations should be functions $MT : M_1 \to M_2$. But this is not really satisfactory because it does not capture the behaviour of models and corresponding preservation requirements by model transformations. As discussed in Section 3, it is advocated by practitioners like Batory that a model transformation should be a functor $F : \mathbf{M} \to \mathbf{D}$. Of course, categories and functors are very important mathematical concepts, which have been applied successfully to formal software development (see e.g. text books [5, 8, 6]). In [6], for each parameterized specification $PSP = (SP, SP_1)$, the parameter specification SP is a sub-specification of the target specification SP_1, and the free functor $F : Cat(SP) \to Cat(SP_1)$ from the category of SP-algebras/data types to that of SP_1-algebras/data types is a model transformation. The same is true for the semantics $SEM : Cat(IMP) \to Cat(EXP)$ of an algebraic module specification in [7] with import and export specifications IMP and EXP. In [8], coordinated functors may be considered as model transformations.

But what about model transformation between visual languages like statecharts and Petri nets? In [5], it is shown how to define this kind of model transformation by graph transformations. It turns out that it is already quite difficult to show the functional behaviour and hence syntactical correctness of such a model transformation. Is it realistic to expect that this kind of model transformation is a functor? Actually, the models are typed attributed graphs and the model transformation can be defined by non-deleting rules and a restriction construction. In [4] (to appear), we show that these kinds of model transformations are functors between the corresponding full subcategories of typed attributed graphs. In this case, the morphisms between models are general graph morphisms, which are not necessarily refinements beween the corresponding visual models. But there are good chances to extend the results to more realistic refinements as morphisms between models. This corresponds to the proposal in Section 3, where morphisms of the corresponding categories are refinements.

Does this mean that the advanced practical point of view can be considered already as a theoretical approach? The main problem is that the models and model transformations in Section 2 and Section 3 are defined by tools which, in general, have no proper formalization up to now. There are only a few exceptions in the area of compiler construction, which have been proven to be correct. But even if there is no hope in the near future that the models and model transformations considered in [12] can be defined on a formal basis, the requirement “model transformations are functors” makes sense, because it has been validated at least for specific instances. But can this be a basis for a functorial framework of model transformations which can be applied to FOMDD as discussed in Section 3? We claim that it should be possible to develop an axiomatic functorial framework for software development approaches like FOMDD.

Although the axiomatic requirements of the framework can be validated only for specific instances, the corresponding theory should lead us to new results which are valid in practice. This would show that the theory is adequate at least for the given practical examples. In a later step it might be possible to find a suitable mathematical abstraction for the corresponding models and model transformations, which allows to prove that the axiomatic requirements are valid. In this sense, the elaborate practical and theoretical points of view could be convergent, supporting the requirement that model transformations should be functors.

5 Conclusion

In this paper, we argued that the view of model transformations as functors reconciles the mathematical and practical perspectives. The implications of the "commuting property" make sense mathematically and provide grounds for "validation on the large". Of course, this is only a proposal up to now, and especially the advantages of this view should be discussed in more detail, since also other formalizations might be useful: on the one hand, model transformations could be also considered as morphisms in a suitable category (of course, also functors are morphisms in the category of categories), and on the other hand, also features (e.g. refinements of models as in product-lines) could be considered as functors, as a dual observation to our claim "Model transformations should be functors". Features as functors would imply that we have categories of representations for a single model/program, and functors to map the objects and arrows of one representation category to another. A consolidation of the formal background for our proposal "Model transformations should be functors", as well as a further discussion of the usefulness both of the claim made in this paper and of the dual claim "Features should be functors", will be subject of future work.

References

[1] D. Batory. 2007. *From Implementation to Theory in Product Synthesis. Keynote Speech, POPL 2007.*

[2] J. Bezivin. 2005. *Model-Driven Engineering: Principles, Scope, Deployment, and Applicability.* In *Proc. GTTSE 2005.*

[3] K. Czarnecki and S. Helsen. 2003. *Classification of Model Transformation Approaches.* In *Proc. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.*

[4] H. Ehrig and C. Ermel. 2007. *Model Transformation by Graph Transformation is a Functor.* to appear.

[5] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. 2006. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer Verlag, Berlin.

[6] H. Ehrig and B. Mahr. 1985. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin.

[7] H. Ehrig and B. Mahr. 1990. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, volume 21 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin.

[8] J. F. Fiadeiro. 2005. *Categories for Software Engineering*, Springer Verlag, Berlin.

[9] T. Mens and P. Van Gorp. 2005. *A Taxonomy of Model Transformation*. In *Proc. Workshop on Graph and Model Transformation (GraMoT'05)*.

[10] Object Management Group (OMG). 2004. *Unified Modeling Language: Superstructure – Version 2.0*. `http://www.omg.org/cgi-bin/doc?ptc/2004-10-02`.

[11] QVT Merge Group. 2005. *Query/View/Transformation (QVT), Version 2.0*. `http://www.omg.org/cgi-bin/apps/doc?ad/05-03-02.pdf`.

[12] S. Trujillo, D. Batory, and O. Diaz. 2007. *Feature Oriented Model Driven Development: A Case Study for Portlets*. In *Proc. ICSE 2007*.