On Horizontal and Vertical Relationships between Models

Martin Gogolla

University of Bremen (D), CS Department, Database Systems Group

Abstract. Detecting, modeling and managing relationships between models are central tasks within model-driven engineering. By taking a simple view on software development, we distinguish in a vertical dimension between domain-specific models, core models, and executable models. A typical example for a vertical relationship is the refinement relationship beween a core model and an executable model. In the horizontal dimension, there may be several so-called property models which have the task to validate or verify particular properties of the core model. Software development coincides in our view with model development, and therefore finding the right models and their relationships is a crucial task.

1 Models in Software Development

We follow the basic assumption expressed in [Béz05] that *Everything is a model*. Under this asumption, every artifact in software development (e.g., a requirement, a specification, executable code, or a test case) may be understood as a model.

Figure 1 shows our view on the use of models in software development. Everything within a rectangle represents a model, and an arrow stands for a relationship between models. As a special case, relationships may be uni- or bidirectional transformations. One concrete development is a path through this graph. The path may go up and down the different layers and may turn left and right as needed. Thus a path representing a development may be labelled as a *yoyo-left-right* path.

In Fig. 1 we have shown three layers for domain, core, and executable models and have indicated typical languages in which these models may be described. However, a development is not restricted to exactly these three layers. For example, there may be more than one core model layer with models being closer or farer from the executable layer. We work with UML models which incorporate OCL invariants as well as OCL pre- and postconditions for operations [GBR07]. Respective class diagrams and OCL invariants may be formulated in a loose way allowing for different implementations or may be very close to programming languages and determine a particular implementation.

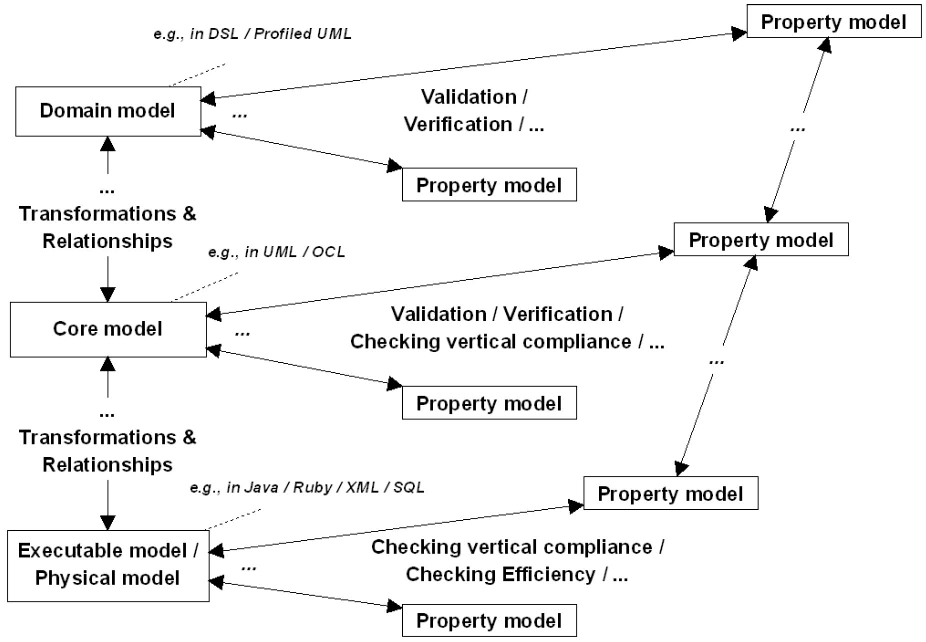

Fig. 1. Models in Software Development

2 Relationships in Software Development

Between the layers, certain relationships between the respective models have to hold in the *vertical* direction. Typical in this context are the refinement relationship or uni- or bi-directional transformations. Within a single layer there may be *horizontal* relationships. These relationships connect models to so-called property models which have the task to validate, to verify, to test or to assert particular properties of the present models. There may be relationships between property models of different layers as well. For example, a test model including test cases of a core model may be propagated to an executable model. But there may be also property models, for example verification models, which are present only on one particular layer. Property models may also be related to other property models on the same layer.

Let us now come to the main message of this contribution which is very simple: Of course, models in software development are important, but *relationships between models possess at least the same degree of importance as the models themselves.* A more systematic study and classification is needed. The kinds of relationships are manifold. They range from set-theoretic relationships and operations (like union or intersection) over theoretical relationships and notions (like refinement or equivalence) to more practical issues (like validation, verification or testing). Similar relationships have been studied in the context of information systems schemas (see, e.g., [BM07]), graph transformations (see, e.g., [HCEL96]) or algebraic specification (see, e.g., [ST06]).

From our experience with UML and OCL models the following relationships seem to be important and may hold between two given models MOD1 and MOD2. This list is not meant to be exhaustive.

- MOD1 union MOD2
- MOD1 intersection MOD2
- MOD1 difference MOD2
- MOD1 projectionOf MOD2
- MOD1 conformsTo MOD2 (MOD1 inheritsFrom MOD2)
- MOD1 representedBy MOD2 (MOD1 instanceOf MOD2)
- MOD1 refines MOD2
- MOD1 satisfies MOD2
- MOD1 equivalentTo MOD2
- MOD1 includedIn MOD2
- MOD1 consistentWith MOD2
- MOD1 validatedBy MOD2
- MOD1 verifiedBy MOD2
- MOD1 testedBy MOD2
- MOD1 implementedBy MOD2

These relationships are partly operations like *MOD1 union MOD2* which result in a new model and are partly pure relationships like *MOD1 refines MOD2* which can be established only under special given conditions. The relationships partly assume a common metamodel or on the other hand define exactly the connection between a model and its metamodel. Regarding the classification into vertical and horizontal relationships, we think that some relationships like refinement or implementation mainly belong into the vertical dimension and other relationships like testing or validation mainly into the horizontal dimension. But, for example, equivalence or inclusion may be useful both as a vertical and horizontal concept.

When we faithfully follow the priciple *Everything is a model*, then we can come to the conclusion that relationships are also models, or at least that relationships can be described by models, which is an aspect that has been already discussed in [BBG+06].

References

[BBG+06] J. Bezivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow. Model Transformations? Transformation Models! In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Proc. 9th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2006)*. LNCS 4199, Springer, Berlin, 2006.

[Béz05] J. Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.

[BM07] P.A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 1–12. ACM, 2007.

[GBR07] M. Gogolla, F. Büttner, and M. Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34, 2007.

[HCEL96] R. Heckel, A. Corradini, H. Ehrig, and M. Löwe. Horizontal and vertical structuring of typed graph transformation systems. *Mathematical Structures in Computer Science*, 6(6):613–648, 1996.

[ST06] D. Sannella and A. Tarlecki. Horizontal composability revisited. In K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, editors, *Essays Dedicated to Joseph A. Goguen*, LNCS 4060, pages 296–316. Springer, 2006.