The Journal of Systems and Software 000 (2015) 1–25

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A three-dimensional taxonomy for bidirectional model synchronization

Zinovy Diskin [a,b], Hamid Gholizadeh [b,*], Arif Wider [c], Krzysztof Czarnecki [a]

[a] University of Waterloo, Waterloo, ON, Canada
[b] McMaster University, Hamilton, ON, Canada
[c] Humboldt-Universität zu Berlin, Berlin, Germany

ARTICLE INFO

Article history:
Received 7 May 2014
Revised 29 May 2015
Accepted 1 June 2015
Available online xxx

Keywords:
Model synchronization
Taxonomy
Formal semantics

ABSTRACT

Early model-driven engineering (MDE) assumed simple pipeline-like scenarios specified by the Model-Driven Architecture approach: platform-independent models that describe a software system at a high-level of abstraction are transformed stepwise to platform-dependent models from which executable source code is generated. Modern applications require a shift toward *networks* of models related in various ways, whose synchronization often needs to be incremental and bidirectional. This new situation demands new features from transformation tools, and a solid semantic foundation to understand and classify these features. We address the problem by presenting a taxonomy of model synchronization types, organized into a 3D-space. Each point in the space refers to a specific synchronization semantics with an underlying algebraic model and the respective requirements for the change propagation operations and their properties. The taxonomy aims to help with identifying and communicating a proper specification for the synchronization problem at hand and for the available solutions offered by tools.

1. Introduction

Early model-driven engineering (MDE) assumed simple pipeline-like scenarios as specified by the Model-Driven Architecture (MDA) approach (Miller and Mukerji, 2003): platform-independent models that describe a software system at a high-level of abstraction are transformed stepwise to platform-dependent models from which executable source code is generated. The generated code was meant to be a secondary artifact similar to assembler or byte code, which (being the result of compilation) can be discarded anytime, whereas models were the primary artifacts to be maintained. Fig. 1 illustrates this pipeline-like flow of transformations: models are always transformed unidirectionally from a higher to a lower level of abstraction.

However, it soon became clear that full code-generation, where modeling languages were meant to succeed programming languages, was difficult to achieve in practice because manual modifications of the generated code (or lower level models) were often unavoidable, or simply more practical. Hence, to keep models' role as first-class citizens, code modifications need to be propagated back to higher level models to keep them in sync. We thus have vertically bi-directional transformation or *round-tripping*.

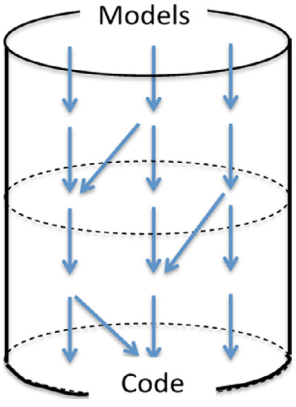

Fig. 1. MDE-pipe in MDA.

Furthermore, in early MDA scenarios models at the same level of abstraction were often considered to have no overlaps. For example, a platform-independent model was meant to have no overlap with the model of a specific platform (both at a high abstraction level), which were together used for generating a platform-specific model at a lower abstraction level. In practice however, often multiple high-level models describe different aspects of a system and do overlap. In such cases consistency between those models must be ensured. Thus, bidirectional synchronization is not only needed *between* abstraction layers ("along streams", e.g., between a UML model and Java code it

* Corresponding author. Tel.: +1 905-525-9140x23358.
E-mail addresses: zdiskin@gsd.uwaterloo.ca (Z. Diskin), hamid.gholizadeh@gmail.com, mohammh@mcmaster.ca (H. Gholizadeh), wider@informatik.hu-berlin.de (A. Wider), kczarnec@gsd.uwaterloo.ca (K. Czarnecki).

http://dx.doi.org/10.1016/j.jss.2015.06.003
0164-1212/

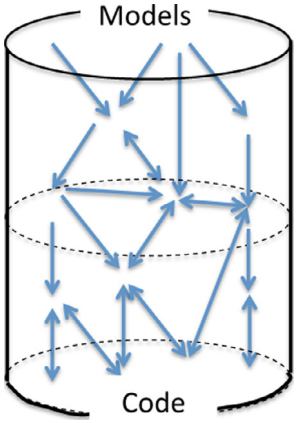

Fig. 2. Modern MDE pipe.

generates), but also *within* abstraction layers ("across streams", e.g., between a class diagram and a sequence diagram specifying an interaction of objects typed over the class diagram). These features transform the MDA pipe into a network of interacting models as shown in Fig. 2. Metaphorically, we can say that round-tripping and overlapping change the flow from "laminar" to "turbulent", as illustrated in Fig. 3.

"Turbulency" of modern model transformation brings several theoretical and practical challenges. The necessity to sync models within abstraction layers makes model alignment (matching) an essential component of synchronization. It is important to separate concerns and distinguish between (1) alignment that needs heuristics and, perhaps, an input from the user, and (2) update propagation, which can be treated as an algebraic operation amenable to full automation after an update policy is established (see Diskin et al., 2011a). Multi-directionality of model synchronization means that changes between models are propagated in all directions in a mutually consistent way. Its implementation via separate but mutually compatible procedures would require a proof of compatibility, and be very difficult to maintain as each direction of change propagation is itself a complex model transformation. For the case of two models, a common solution nowadays is to specify a consistency relation between the models and let the update propagation procedures be inferred from this specification, so that they are always consistent by construction. Such an approach is commonly referred to as bidirectional transformation or *bx*. We are not aware of any implementation of the multi-directional case.

Semantics of such complex synchronization procedures as above is not well understood, whereas clear semantics is crucial for synchronization tools because otherwise users have no trust in automatic synchronization. Severe problems of adopting the industrial standard QVT-R (which treats the binary synchronization case) is a typical example: despite early availability of QVT-R tools on the market, its adoption could hardly be considered successful. As argued by Stevens (2010), the most probable reason for the failure is a set of major semantic issues revealed and discussed in Stevens (2010). Moreover, building semantic foundations for QVT-R turned out a challenging issue. A formal semantics for a relatively simple *check-only* mode required rather intricate mathematical constructs based on symbolic graphs (Guerra and de Lara, 2012); formalization of the *enforce* mode is still an open issue. Thus, understanding of even a binary general synchronization is challenging, not to mention the multi-ary case (which fits in our metaphor of turbulency: the latter is known to be a very complex mechanical phenomenon, whose specification requires special mathematical methods). The lack of a sound underlying theory leads to a shaky conceptual framework, ambiguous terminology and flawed communication between tool users and tool builders, and ultimately to deficient tools. Indeed, imperfect MDE tooling (particularly, for synchronization) is considered by many as a major barrier for the industrial adoption of MDE (Haan, 2008; Kuhn et al., 2012; Tomassetti et al., 2012; Whittle et al., 2013); we will discuss this issue in more detail in Section 2.

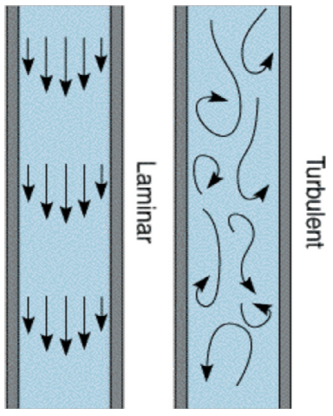

Fig. 3. Change in MDE flow.

A typical first step in approaching a complex problem in science or engineering is to build a basic ontology of the domain and classify the "species" inhabiting it. This is what we are going to do in this paper for the domain of bx—the simplest, but practically very important case of multi-directional model synchronization, when the system to be kept in sync consists of two models. Several highlights of our goals in the paper, means to achieve them, and their interpretation, are as follows.

- We only consider the binary case.
- We classify semantic possibilities for model synchronization rather than possible implementations and tools. In the context of tooling, we classify design choices—the *what* rather than the *how*.
- The classification is based on a mathematical model of bx provided by *delta lenses* (Diskin et al., 2011a; 2011b): a family of algebraic structures that specify bx in an abstract declarative way (a brief description will be given in Section 2.1, and a full account can be found in Appendix A). A central ingredient of delta lenses is the notion of inter-model correspondence, which establishes consistency or inconsistency of the models. If one of the initially consistent models is updated and consistency is broken, the other model is to be accordingly updated too to restore consistency; we say that the update (or the change) is propagated from one model to the other.
- No specific assumptions are made about what models, their updates and correspondences are, which provides a significant flexibility of the framework. One typical application is synchronization of the source and the target of a model transformation translating models from a source metamodel to a target metamodel, with correspondences given by traceability mappings. Another application is synchronization of two views of the same system, with correspondences given by model matching. In both cases, correspondences between models are implicitly typed by the respective mappings between metamodels; moreover, the latter can be seen as view definitions that actually determine transformations (Diskin, 2009; Diskin et al., 2010). Thus, metamodels and their mappings are important but implicit components of the framework. In contrast, for scenarios of model-metamodel coevolution (Ruscio et al., 2012; Mantz, 2014) the metamodel participation is quite explicit and correspondences are given by the typing mapping between a model and its metamodel rather than between two metamodels. Although the case of model coevolution can be formally subsumed by the framework, the underlying intuition is

Please cite this article as: Z. Diskin et al., A three-dimensional taxonomy for bidirectional model synchronization, The Journal of Systems and Software (2015), http://dx.doi.org/10.1016/j.jss.2015.06.003