Behavior, Time and Viewpoint Consistency: Three Challenges for MDE

José Eduardo Rivera[1], José Raul Romero[2], and Antonio Vallecillo[1]

[1]Universidad de Málaga Spain
[2]Universidad de Córdoba Spain
{rivera,av}@lcc.uma.es, jrromero@uco.es

Abstract. Although Model Driven Software Development (MDSD) is achieving significant progress, it is still far from becoming a real Engineering discipline. In fact, many of the difficult problems of the engineering of complex software systems are still unresolved, or simplistically addressed by many of the current MDSD approaches. In this position paper we outline three of the outstanding problems that we think MDSD should tackle in order to be useful in industrial environments.

1 Introduction

Although both MDSD and MDA have experienced significant advances during the past 8 years, some of the key difficult issues still remain unresolved. In fact, the number of engineering practices and tools that have been developed for the industrial design, implementation and maintenance of large-scale, enterprise-wide software systems is still low — i.e. there are very few real *Model-Driven Engineering* (MDE) practices and tools. Firstly, many of the MDSD processes, notations and tools fall apart when dealing with large-scale systems composed of hundred of thousands of highly interconnected elements; secondly, MDE should go beyond conceptual modeling and generative programming: it should count on mature tool-support for automating the design, development and analysis of systems, as well as on measurable engineering processes and methodologies to drive the effective use of all these artifacts towards the predictable construction of software systems. In particular, engineering activities such as simulation, analysis, validation, quality evaluation, etc., should be fully supported.

We are currently in a situation where the industry is interested in MDE, but we can easily fail again if we do not deliver (promptly) anything really useful to them. There are still many challenges ahead, which we should soon address in order not to lose the current momentum of MDE.

In this position paper we focus on three of these challenges. Firstly, the specification of the behavioral semantics of metamodels (beyond their basic structure), so that different kinds of analysis can be conducted, e.g., simulation, validation and model checking. A second challenge is the support of the notion of time in these behavioral descriptions, another key issue to allow industrial systems to be realistically simulated and properly analyzed — to be able to conduct, e.g.,

M.R.V. Chaudron (Ed.): MODELS 2008 Workshops, LNCS 5421, pp. 60–65, 2009.

performance and reliability analysis. Finally, we need not only to tackle the *accidental* complexity involved building software systems, but we should also try to deal with their *essential* complexity. In this sense, the effective use of independent but complementary viewpoints to model large-scale systems, and the specification of correspondences between them to reason about the consistency of the global specifications, is the third of our identified challenges.

2 Adding Behavioral Semantics to DSLs

Domain Specific Languages (DSLs) are usually defined only by their abstract and concrete syntaxes. The abstract syntax of a DSL is normally specified by a metamodel, which describes the concepts of the language, the relationships among them, and the structuring rules that constrain the model elements and their combinations in order to respect the domain rules.

The concrete syntax of a DSL provides a realization of the abstract syntax of a metamodel as a mapping between the metamodel concepts and their textual or graphical representation (see Fig. 1). A language can have several concrete syntaxes. For visual languages, it is necessary to establish links between these concepts and the visual symbols that represent them — as done, e.g, with GMF. Similarly, with textual languages links are required between metamodel elements and the syntactic structures of the textual DSL.

Current DSM approaches have mainly focused on the structural aspects of DSLs. Explicit and formal specification of a model semantics has not received much attention by the DSM community until recently, despite the fact that this creates a possibility for semantic mismatch between design models and modeling languages of analysis tools [1]. While this problem exists in virtually every domain where DSLs are used, it is more common in domains in which behavior needs to be explicitly represented, as it happens in most industrial applications of a certain complexity. This issue is particularly important in safety-critical real-time and embedded system domains, where precision is required and where semantic ambiguities may produce conflicting results across different tools. Furthermore, the lack of explicit behavioral semantics strongly hampers the

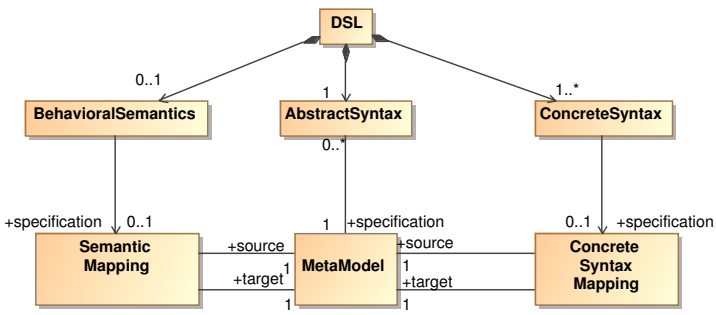

Fig. 1. Specification of a Domain Specific Language

development of formal analysis and simulation tools, relegating models to their current common role of simple illustrations.

The definition of the semantics of a language can be accomplished through the definition of a mapping between the language itself and another language with well-defined semantics (see Fig. 1). These *semantic mappings* [2] are very useful not only to provide precise semantics to DSLs, but also to be able to simulate, analyze or reason about them using the logical and semantical framework available in the target domain. In our opinion, in MDE these mappings can be defined in terms of model transformations.

Describing Dynamic Behavior. There are several ways for specifying the dynamic behavior of a DSL, from textual to graphical. We can find approaches that make use of, e.g., UML diagrams, rewrite logic, action languages or Abstract State Machines [3] for this aim. One particular way is by describing the evolution of the state of the modeled artifacts along some time model. In MDE, model transformation languages that support in-place update [4] can be perfect candidates for the job. These languages are composed of rules that prescribe the preconditions of the actions to be triggered and the effects of such actions.

There are several approaches that propose in-place model transformation to deal with the behavior of a DSL. One of the most important graphical approaches on this topic is graph grammars [5,6], in which the dynamic behavior is specified by using visual rules. These rules are visually specified as models that use the concrete syntax of the DSL. This kind of representation is quite intuitive, because it allows designers to work with domain specific concepts and their concrete syntax for describing the rules [5]. There are also other graphical approaches, most of which are in turn based on graph grammars. Among them, we can find the visual representation of QVT [7] (where QVT is given in-place semantics) or the use of different (usually extended) UML diagrams [8,9]. These approaches do not use (so far) the concrete syntax of the DSL, but an object diagram-like structure. Furthermore, most of them (including graph grammars approaches) use their own textual language to deal with complex behavior, such as Java [8] or Python [10].

Model Simulation and Analysis. Once we have specified the behavior of a DSL, the following step is to perform simulation and analysis over the produced specifications. Defining the model behavior as a model will allow us to transform them into different semantic domains. Of course, not all the transformations can always be accomplished: it depends on the expressiveness of the semantic approach. In fact, simulation and execution possibilities are available for most of the approaches in which behavior can be specified (including of course in-place transformations), but the kind of analysis they provide is normally limited. In general, each semantic domain is more appropriate to represent and reason about certain properties, and to conduct certain kinds of analysis [3].

A good example of this is Graph Transformation, which has been formalized into several semantic domains to achieve different kinds of analysis. Examples include Category theory to detect rule dependencies [11]; Petri Nets to allow termination and confluence analysis [5]; or Maude and rewrite logic to make

models amenable to reachability and model-checking analysis [12]. We have been working on the formalization of models and metamodels in equational and rewriting logic using Maude [13]. This has allowed us to specify and implement some of the most common operations on metamodels, such as subtyping or difference [14], with a very acceptable performance. This formalization has also allowed us to add behavior [15] in a very natural way to the Maude specifications, and also made metamodels amenable to other kinds of formal analysis and simulation.

3 Adding Time to Behavioral Specifications

Formal analysis and simulation are critical issues in complex and error-prone applications such as safety-critical real-time and embedded systems. In such kind of systems, timeouts, timing constraints and delays are predominant concepts [16], and thus the notion of time should be explicitly included in the specification of their behavior. Most simulation tools that enable the modeling of time require specialized knowledge and expertise, something that may hinder its usability by the average DSL designer. On the other hand, current in-place transformation techniques do not allow to model the notion of time in a quantitative way, or allow it by adding some kind of clocks to the DSL metamodel. This latter approach forces designers to modify metamodels to include time aspects, and allows them to easily design rules that lead the system to time-inconsistent states [16].

One way to avoid this problem is by extending behavioral rules with their duration, i.e., by assigning to each action the time it needs to be performed. Analysis of this kind of timed rules cannot be easily done using the common theoretical results and tools defined for graph transformations. However, other semantic domains are better suited. We are now working on the definition of a semantic mapping to real-time Maude's rewrite logic [17]. This mapping brings several advantages: (1) it allows to perform simulation, reachability and model-checking analysis on the specified real-time systems; (2) it permits decoupling time information from the structural aspects of the DSL (i.e., its metamodel); and (3) it allows to state properties over both model states and actions, easing designers in the modeling of complex systems.

4 Viewpoint Integration and Consistency

Large-scale heterogeneous distributed systems are inherently much more complex to design, specify, develop and maintain than classical, homogeneous, centralized systems. Thus, their complete specifications are so extensive that fully comprehending all their aspects is a difficult task. One way to cope with such complexity is by dividing the design activity according to several areas of concerns, or *viewpoints*, each one focusing on a specific aspect of the system, as described in IEEE Std. 1471. Following this standard, current architectural practices for designing open distributed systems define several distinct viewpoints. Examples include the viewpoints described by the growing plethora of Enterprise Architectural Frameworks (EAF): the Zachman's framework, ArchiMate, DoDAF, TOGAF, FEAF or

the RM-ODP. Each viewpoint addresses a particular concern and uses its own specific (viewpoint) *language*, which is defined in terms of the set of concepts specific that concern, their relationships and their well-formed rules.

Although separately specified, developed and maintained to simplify reasoning about the complete system specifications, viewpoints are not completely independent: elements in each viewpoint need to be related to elements in the other viewpoints in order to ensure the *consistency* and *completeness* of the global specifications. The questions are: how can it be assured that indeed *one* system is specified? And, how can it be assured that no views impose contradictory requirements? The first problem concerns the conceptual *integration* of viewpoints, while the second one concerns their *consistency*. There are many approaches that try to tackle the problem of consistency between viewpoints, many of them coming from the ADL community (see, e.g., [3] for a list of such works). However, many of the current viewpoint modeling approaches to system specification used in industry (including the IEEE Std. 1471 itself and the majority of the existing EAFs) do not address these problems [18].

There are several ways to check viewpoint consistency. In some approaches such as the OpenViews framework [19], two views are consistent if a design can be found that is a refinement of both views. Other approaches, such as Viewpoints [20], consistency requirements are defined in terms of rules, which are specified as queries on the database that contains the viewpoints. The database performs then the consistency checks, using first-order logic. But the most general approach to viewpoint consistency is based on the definition of *correspondences* between viewpoint elements.

Correspondences do not form part of any of the viewpoints, but provide statements that relate the various different viewpoint specifications—expressing their semantic relationships. The problem is that current proposals and EAFs do not consider correspondences between viewpoints, or assume they are trivially based on name equality between correspondent elements, and are implicitly defined. Furthermore, the majority of approaches that deal with viewpoint inconsistencies assume that we can build an underlying metamodel containing all the views, which is not normally true. For instance, should such a metamodel consist of the intersection or of the union of all viewpoints elements? Besides, the granularity and level of abstraction of the viewpoints can be arbitrarily different, and they may have very different semantics, which greatly complicates the definition of the common metamodel.

Our efforts are currently focused on the development of a generic framework and a set of tools to represent viewpoints, views and correspondences, which are able to manage and maintain viewpoint synchronization in evolution scenarios, as reported in [21], and that can be used with the most popular existing EAFs.

References

1. Kleppe, A.G.: A language description is more than a metamodel. In: Proc. of ATEM 2007 (oct 2007), `http://megaplanet.org/atem2007/ATEM2007-18.pdf`

2. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? Computer 37(10), 64–72 (2004)
3. Vallecillo, A.: A Journey through the Secret Life of Models. In: Position paper at the Dagstuhl seminar on Model Engineering of Complex Systems (MECS) (2008), http://drops.dagstuhl.de/opus/volltexte/2008/1601
4. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOPSLA 2003 Workshop on Generative Techniques in the context of MDA (2003)
5. de Lara, J., Vangheluwe, H.: Translating model simulators to analysis models. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 77–92. Springer, Heidelberg (2008)
6. Kastenberg, H., Kleppe, A.G., Rensink, A.: Defining object-oriented execution semantics using graph transformations. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 186–201. Springer, Heidelberg (2006)
7. Marković, S., Baar, T.: Semantics of OCL Specified with QVT. Software and Systems Modeling (SoSyM) (2008)
8. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: A new graph rewrite language based on the unified modeling language. In: Proc. of the VI International Workshop on Theory and Application of Graph Transformation (1998)
9. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 323–337. Springer, Heidelberg (2000)
10. de Lara, J., Vangheluwe, H.: Defining visual notations and their manipulation through meta-modelling and graph transformation. Journal of Visual Languages and Computing 15(3-4), 309–330 (2006)
11. Ehrig, H., Karsten, Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006)
12. Rivera, J.E., Guerra, E., de Lara, J., Vallecillo, A.: Analyzing rule-based behavioral semantics of visual modeling languages with maude. In: Proc. of SLE 2008, Toulouse, France. LNCS. Springer, Heidelberg (2008)
13. Romero, J.R., Rivera, J.E., Durán, F., Vallecillo, A.: Formal and tool support for model driven engineering with Maude. JOT 6(9), 187–207 (2007)
14. Rivera, J.E., Vallecillo, A.: Representing and operating with model differences. In: Proc. of TOOLS Europe 2008. LNBIP, vol. 11, pp. 141–160. Springer, Heidelberg (2008)
15. Rivera, J.E., Vallecillo, A.: Adding behavioral semantics to models. In: Proc. of EDOC 2007, pp. 169–180. IEEE Computer Society, Los Alamitos (2007)
16. Gyapay, S., Heckel, R., Varró, D.: Graph transformation with time: Causality and logical clocks. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 120–134. Springer, Heidelberg (2002)
17. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
18. Romero, J.R., Vallecillo, A.: Well-formed rules for viewpoint correspondences specification. In: Proc. of WODPEC 2008 (2008)
19. Boiten, E.A., Bowman, H., Derrick, J., Linington, P., Steen, M.W.: Viewpoint consistency in ODP. Computer Networks 34(3), 503–537 (2000)
20. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: a framework for integrating multiple prespectives in systems development. SEKE journal 2(1), 31–58 (1992)
21. Eramo, R., Pierantonio, A., Romero, J.R., Vallecillo, A.: Change management in multi-viewpoint systems using ASP. In: Proc. of WODPEC 2008 (2008)