Bridging The Formal Techniques and Model-Driven Engineering Divide

Robert France
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
france@cs.colostate.edu

October 30, 2010

1 Introduction

Model driven engineering (MDE) approaches attempt to reduce the accidental complexities associated with labor-intensive development of large, complex software systems, through the use of (1) models that describe complex systems at multiple levels of abstraction and from a variety of perspectives, and (2) automated support for transforming and analyzing models [5]. In the MDE vision of software development, models are the primary artifacts of development and developers rely on computer-based technologies to transform models to running systems.

To a casual observer looking out from the formal methods (FM) community it may seem that MDE research is subsumed by research in the formal specification and verification area. A closer examination may suggest that this is not the case. The formal specification languages that have been developed in the FM community thus far use languages that allow developers to describe systems from a very small number of viewpoints. It is well known that the more expressive a modeling language is, the more intractable the problem of developing mechanical semantic analysis techniques becomes. It should therefore not be surprising then that formal specification languages restrict their viewpoints. In MDE, a model of a complex system consists of many views created using a wide variety of viewpoints. For example, the UML provides modelers with 13 diagram types, each can be used to richly describe a system from a different perspective.

The differences in research scopes suggest that MDE provides a context in which formal specification and verification techniques can be applied. There is

evidence that this is already taking place (e.g., see [3, 6, 7, 8, 9]). With respect to the UML, in the late nineties the precise UML (pUML) group helped raise awareness of the need for more formal descriptions of UML semantics to enable rigorous analysis of structural and functional properties of systems captured in UML models. Over the last decade we have seen a significant number of papers on using relatively mature formal verification techniques to analyze properties described in particular UML models (e.g., there has been significant work on using model-checking techniques to analyze UML statemachine models, and petri net variants to analyze activity models).

Despite the focused attempts there are very few UML-based verification tools that can be described as usable by practitioners. In the following I discuss some of the opportunities for applying verification techniques in MDE and discuss some of the challenges. For the most part, the opportunities and challenges are presented in terms of UML modeling issues, primarily because this is one of the more widely-used (and misused) MDE languages out there, and there is a dire need for practical UML-based verification tools.

1.1 Reaching for Lower Hanging Fruit

The UML has reached a level of maturity that now allows us to reach for some of the lower hanging fruit (not necessarily the same as low-hanging fruit!) where application of rigorous verification techniques are concerned. One of the frustrating experiences that a modeling student or practitioner learning a language such as the UML goes through is determining if his/her model is, in some sense, a valid description. In the case of students, the only feedback that they often receive is the instructor's grade of their work. There is a need to provide modelers, in particular, UML modelers, with some means of checking the validity of their model.

An obvious approach is to provide some support for executing or animating models. At Colorado State University (CSU) we developed the UMLAnT (UML Animation and Testing) tool as a means for dynamically analyzing (testing) UML design models. A UMLAnT design model consists of class diagrams with operations specified in a Java-like action language called JAL [4]. UMLAnT is an Eclipse plugin that provides support for (1) generating test inputs that satisfy criteria based on coverage of elements in a sequence diagram that describes the scenarios that will be exercised in a test, (2) executing the design model using test inputs (a test input is an operation with parameter values), and (3) showing execution progress in terms of sequence diagrams and changes to object configurations. We are currently updating the tool to the latest version of Eclipse and making it as robust as we can.

We are also developing lightweight scenario-based analysis techniques that al-

low developers to check whether a scenario describing a desired or undesired behavior is supported by a model [10]. The technique provides a less expensive way of analyzing a system in the cases where exhaustive formal analysis is not possible or cost-effective. In the approach we are developing, a behavior is described as a sequence of snapshots, where a snapshot is an object configuration that conforms to a class diagram. A class model with operations specified in the Object Constraint language (OCL) is transformed to a class model, called a Snapshot Model, that characterizes all possible behaviors (sequences of snapshots). A verifier then provides scenarios (expressed as sequence diagrams) and the analysis tool we are developing checks whether these scenarios conform to the Snapshot Model.

One of the problems that our analysis approaches and those developed by other researchers face is that they do not handle incomplete models well. This is one of the challenges that we are currently tackling in our analysis work.

Another aspect that requires attention is ensuring consistency of behavioral and structural concepts across different modeling views. This is a particularly challenging problem in the UML, and is sometimes one of the reasons practitioners limit their use to one or two UML diagram types (typically class diagrams, sequence diagrams or statemachine diagrams). One of the problems that hinders research in this area is the size of the UML language (as reflected in its metamodel) - this makes it very difficult to determine precisely the consistency relationships that must hold across elements in different diagrams. Furthermore, it has not been verified that the UML metamodel is indeed a valid description that can be relied upon correctly define these relationships. A good usability challenge problem for verification tools is finding an answer to the question "Is the UML metamodel correct?".

1.2 Transformations, Semantc Variations, and Models@Run.Time

The pevious subsection identified some obvious opportunties for applying verification techniques in the MDE context. That was just the tip of the iceberg; there other more challenging verification problems that should be tackled in MDE. A challenging problem concerns verification of model transformations. In a recently published paper on testing model transformations we highlighted some of these challenges [1]. One of the major problems concerns generating an adequate set of test models. Generating test inputs for programs that use inputs with simple structures is challenging in itself; when the inputs are models with complex structures the challenges are greater.

Another problem that must be dealt with is the variety of semantics that can be associated with languages such as the UML. In the UML some parts of the semantics are intentionally left undefined to allow users to tailor semantics to their needs. While formal methods purists may argue for defining a single semantics for

the UML, the practical reality is that different groups use the UML differently, and this need must be supported. It is highly unlikely that a singe verification approach would meet all structural, functional and behavioral analysis needs. To tackle this problem we have started a research initiative called GeMoC (Generic Model of Computation) with the goal of developing a verification framework that can be used in a modeling environment that supports a variety of semantics (or models of computation). This is a challenging task that is best tackled by collaborating teams of researchers with expertise in MDE and verification techniqueds. The initiative was started by Benoit Combemale from INRIA in France, and involves researchers from France and my group at CSU.

In closing, I'll mention an emerging MDE research area that attempts to extend the use of model to run time management: model@run.time [2]. There has been significant work on using models to support runtime adaptation of software. Verifying adaptations at runtime is a particularl challenging problem that groups working in this area are currently tackling.

References

[1] B. Baudry, T. Dinh-Trong, J.-M. Mottu, D. Simmonds, R. France, S. Ghosh, F. Fleurey, and Y. Le Traon. Challenges for model transformation testing. In *IMDT workshop in conjunction with ECMDA'06*, 2006.

[2] Gordon Blair, Nelly Bencomo, and Robert B. France. Models@ run.time. *Computer*, 42:22–27, 2009.

[3] Dan Chiorean, Mihai Pasca, Adrian Crcu, Cristian Botiza, and Sorin Moldovan. Ensuring UML Models Consistency Using the OCL Environment. *Electronic Notes in Theoretical Computer Science*, 102:99 – 110, 2004. Proceedings of the Workshop, OCL 2.0 - Industry Standard or Scientific Playground?

[4] T.T. Dinh-Trong, S. Ghosh, and R.B. France. A systematic approach to generate inputs to test UML design models. *Software Reliability Engineering, 2006. ISSRE '06. 17th International Symposium on*, pages 95–104, Nov. 2006.

[5] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*. IEEE-CS Press, 2007.

[6] Martin Gogolla, Jrn Bohling, and Mark Richters. Validating UML and OCL models in USE by automatic snapshot generation. *Journal on Software and System Modeling*, 4:2005, 2005.

[7] Alexander Knapp. A formal semantics for UML Interactions. In *Proceedings of UML'99, Springer-Verlag LNCS 1723*, pages 116–130, 1999.

[8] Johan Lilius and Ivan Porres Paltor. Formalising UML State Machines for Model Checking. In *Proceedings of UML'99, Springer-Verlag LNCS 1723*, pages 430–445, 1999.

[9] Seyyed M. A. Shah, Kyriakos Anastasakis, and Behzad Bordbar. From UML to Alloy and back again. In *MoDeVVa '09: Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*, pages 1–10, New York, NY, USA, 2009. ACM.

[10] Lijun Yu, Robert B. France, and Indrakshi Ray. Scenario-Based Static Analysis of UML Class Models. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *MoDELS*, volume 5301 of *Lecture Notes in Computer Science*, pages 234–248. Springer, 2008.