Bridging the Gap in Model-based V&V Methodology

Position Paper

Mass Soldal Lund

SINTEF ICT, Oslo, Norway
mass.s.lund@sintef.no

Abstract. In this paper we suggest that one of the big challenges in development of model-based V&V methodology is bridging the gap between the low-level user guides of V&V tools and the high-level text-book descriptions of model-driven development methodologies. As a step towards mitigating this we suggest a structure for V&V methodology with three layers: a bottom level with usage of specific V&V techniques, a middle level with V&V method patterns, and a top level with overall development methodology. Within this structure we suggest that the middle level is often missing in presentations of model-based V&V and in presentations of model-driven development methodologies, and is where the need for development is the greatest.

Keywords: Model-driven development, verification and validation, methodology.

1 Introduction

The development of methods for verification and validation (V&V) can be divided into three parts: development of theory, development of tools implementing the theory, and development of the pragmatics concerning the theory and tools. However, there is a tendency among developers of methods for V&V to have much emphasis on the development of theory and tools, while the pragmatics of the methods are often reduced to user guides of the tools. This is unfortunate, as pragmatics is about placing the theory and tools in context and describing how they should be applied beyond just operating the tools.

In fact, an investigation into the state-of-the-art with respect to model-driven system development methodologies and with respect to model-based V&V processes reveals a gap between how V&V is treated in high-level descriptions of system development methodologies and how it is treated by the providers of model-based V&V methods. Model-driven development methodologies do not characterize well how model-based V&V methods should be integrated in the model-driven approach, and model-based V&V methods do to a very little degree characterize how they should be integrated into a broader model-driven methodology. The result is that the V&V methods are poorly integrated with the

overall methodology, and none get the full benefits of the model-driven/model-based approach. We believe that bridging this gap is a main challenge in the development of model-based V&V methods and methodology. As a contribution to how this gap can be bridged we propose a way of describing model-based V&V methodology in a three layered structure.

The reminder of this paper is structured as follows: In Sect. 2 we take the top-down approach and investigate how V&V methodology is treated in model-driven system development, and in Sect. 3 we take the bottom-up approach and look at how it is treated by providers of model-based V&V. Then in Sect. 4 we describe our proposed structure of model-based V&V methodology. Finally, in Sect. 5 we provide conclusions.

2 Top-down: Model-driven System Developments Methodologies

There has not been much focus on V&V processes in model-driven system development. This may be illustrated by the almost complete absence of V&V in special issues of *IEEE Computer* [18] and *IEEE Software* [17] on model-driven development. The only exceptions are the observations made by Mellor et al. [13] that

> [a] model is a coherent set of formal elements describing something [...] built for some purpose that is amendable to a particular form of analysis, such as [...] [t]est case generation

and by Selic [16] that

> we can attain MDD's full benefits only when we exploit its potential for automation. This includes [...] [a]utomatically verifying models on a computer (for example, by executing them)

and hence that models should be executable. However, these observations are not pursued. A reason for this absence might be that the focus on model transformation and code generation leads to the view that there is less need for V&V as implementations are automatically generated from models; for example Schmidt [15]:

> The ability to synthesize artefacts from models helps ensure the consistency between application implementations and analysis information associated with functional and QoS requirements captured by models. This automated transformation process is often referred to as "correct-by-construction," as opposed to conventional handcrafted "construct-by-correction" software development processes that are tedious and error prone.

Several model-driven methodologies, like COMET [5], KobrA [2] and Catalysis [9], barely mention testing, and with no real relation to what in other respects make them model-driven methodologies. The Rational Unified Process

(RUP) [11] has a high emphasis on testing and prescribes use cases as the basis for testing. Still, it describes testing only at a general level and with no reference to the use of models. Much the same is the case with Agile Model-Driven Development (AMDD) [1] – general discussions on testing, but without references to the models assumed by the development methodology. An exception is the TIMe method [6], which discusses the application of different V&V techniques in model-driven development in some detail.

Still, this must be considered a shortcoming in the present state of model-driven development, and also one of its great challenges; how to really integrate *model-based* V&V in *model-driven* development, and how to get V&V to benefit from having a model-driven approach.

3 Bottom-up: Model-based V&V Techniques

In the previous sections we took the top-down approach and looked at how V&V are treated in different model-driven development methodologies. In this section we take the bottom-up approach and look at how V&V processes are presented in the context of model-based V&V methods and tools, more specifically performance simulation, formal verification and testing, and how these processes are seen in relation to overall system development.

3.1 Performance Simulation

Balsamo et al. [4] report from a survey of model-based performance prediction methods, and make the observation that:

> Although several of [the] approaches have been successfully applied, we are still far from seeing performance prediction integrated into ordinary software development.

In the survey several methods were analyzed with respect to their potential for automation, and to where in the system development lifecycle they can be integrated. Fig. 1 shows a generic performance simulation process that is common for all the methods in the survey.

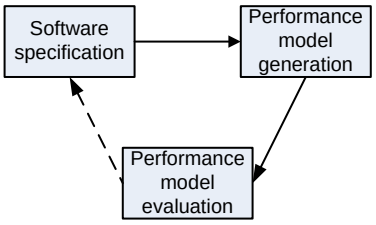

Fig. 1. Performance simulation process

The survey shows that most of the available methods have high potential for automation both in the performance model generation and the performance model evaluation, though the full potential is not realised in all of the tools. The dashed line represents feedback from the evaluation back to the specification, a feature considered by the authors to be a key success factor for the practical application of performance simulation. This is however a feature that is supported by few of the methods and tools in the survey.

The survey further reports that, when seen in relation to system development, most of the methods apply to design models. A clear trend is noted in the application of UML as the specification language used for design models. While some of the methods also apply to earlier phases of the development process (software specification and software architecture), analysis at these levels is more complicated because of lack of information and hence more variables must be considered. An evident challenge in model-based simulation is in defining methods and processes that may be applied over larger parts of the system development.

3.2 Formal Verification

Most verification techniques are based on the construction of formal models of the system under development, and one would therefore believe that verification was closely related to model-based or model-driven development. As Vaandrager [20] observes:

> Model driven development provides a great opportunity to improve the verification and validation process through the introduction of formal techniques. The systematic, structured construction of models by itself already supports validation and verification. In addition, the fact that models are available in machine readable form enables the application of a whole range of (mathematical) techniques for analysis such as theorem proving, model checking, model based testing and runtime verification.

Or in the words of Broy [7]:

> If developers understand how to support the entire development lifecycle with models, they can develop model-driven quality assurance, ensuring the system's quality by analyzing, validating, and verifying models. They can use the models to generate additional information that lets them check certain properties of development artefacts.

However, verification literature seldom puts much emphasis on the relation between the process of verification and the system development process. An example may be the book *Model Checking* [8] where the "process of model checking" is given less than a page. The process is described as follows:

1. *Modelling*: Create a design model of the system.
2. *Specification*: Express the desired properties of the design in a suitable language, usually temporal logic.

3. *Verification*: Analyze the model using a model checker. If the model does not conform to the specification, the model must be adjusted and analyzed again.

Primitive processes like this, and even less elaborated processes, are often found. There are exceptions; e.g. in [14] quite detailed guidelines for application of a method for step-wise refinement is given. Still, the only reference to system development is a statement in the introduction that it

> is not intended to be a complete methodology for system development, but should rather be seen as a supplement to methodologies like e.g. RUP.

Symptomatic of this is also that the "Verified Software Grand Challenge" has "unified theory of program construction", "integrated tool suite that support verification" and "repository of formal specifications and verified codes" on its agenda [21], but not a process or methodology for the application of these (see also Hoare & Misra [10]).

Because formal verification is not applied widely in industry, and cannot, like testing, be seen as a "standard part of standard development". Formal verification is also seldom treated in system development literature.

3.3 Testing

For model-based testing, the situation is somewhat better. In particular the books *Practical Model-Based Testing* [19] and *Model-Driven Testing* [3] are examples of approaches that aspire to place model-based testing in a wider perspective. The former has a bottom-up perspective. If first describes model-based testing methods in detail and then at the end discusses how these methods should be utilized in practice in the context of different system development methodologies (in particular agile methods and Unified Process).

In contrast, the latter of these approaches adopts a top-down perspective. It starts out by situating model-based testing in the context of a system development methodology (the W-model), and then goes on to describe different model-based testing methods and how model-based testing is applied at the different stages of the development process.

Despite these approaches, it is still the case that it is difficult to find in the literature on model-based testing treatments on how it relates to model-driven development.

4 Three Levels of V&V Methodology

In the preceding sections we have looked at how the relation between model-based V&V methods and model-driven development is treated both from a top-down and a bottom-up perspective. In a way, what we can see that there is something missing in between; the top-down approach does not reach sufficiently low and the bottom-up approach not sufficiently high in order for them to meet.

As a way of mitigating this situation we propose a structure for describing and reasoning about model-based V&V methodology where this is made explicit. In particular we propose model-based V&V methodology should be described in three layers or levels:

- *A bottom level* that describes processes for the application of specific V&V methods and tools.
- *A middle level* that describes method patterns for different kinds of model-based V&V and how that may be parameterized or instantiated with the specific methods of the bottom layer.
- *A top level* that describes general methodology for model-based V&V, which is to say a characterization of how the method patterns integrate in the overall model-driven development process.

Not surprisingly, the top level corresponds roughly to the high-level text-book descriptions of development methodologies, while the bottom level corresponds roughly to the low-level user guides of V&V tools. The middle level is what we believe is often missing; descriptions of how model-based V&V methods are to be operationalized in the context of system development, and in particular a tighter connection to the "model flow" of model-driven development.

A goal of describing V&V methodology should be to ease the integration of V&V processes in existing system development methodology. These three levels should therefore also be understood as three levels of application and three levels of configurability. The three levels of adaption can be illustrated as follows.

For an organization that already has a well-defined process for conducing V&V, processes at the bottom level could take the form of a repository of tool specific procedures for V&V that may be employed to fill some specific needs in the system development.

An organization that has a well-defined model-driven development process, but wishes to introduce one or more methods of model-based V&V can apply one or more of the method patterns processes from the middle level. The patterns should have variation points where suitable, so that adaptation to specific needs and existing processes may be made.

The top level will be for organizations that want to introduce a V&V methodology without the need or wish to retain existing processes. As with the middle level, the general methodology of the top level should have variation points that allow configuration for the specific needs of an organization.

5 Conclusions

As we have seen in this paper there is a gap between how V&V processes are presented in model-driven development methodologies and how V&V processes are presented in context of model-based V&V methods and tools; in a blunt formulation we say that the model-driven development methodologies do not pay sufficient attention to how V&V actually should be carried out, and model-based V&V methods do not pay sufficient attention to how they fit into model-driven

development. This is a gap in a double sense. It is an actual gap in that it is hard to find the relation described in the literature, but also a gap between the current state of model-based V&V and where it ideally should be.

As a contribution towards bridging this gap, we suggest describing model-based V&V methodology in a three layered structure: a *bottom level* describing application of specific V&V techniques, a *top level* describing high-level methodology, and a *middle level* describing V&V method patterns that tie the top and bottom levels together. Obviously, the middle level is where the gap is and where the need for development is the greatest.

The insights presented in this paper might come across as trivial. But no matter how trivial it may appear we believe the gap is real and that it is a factor holding model-based V&V back from reaching its full potential. Further, we are convinced that filling the gap is a real challenge, and believe that way of thinking presented in this paper is a good starting point for contributing to this challenge. In our own experience the approach is useful; our first attempt at defining V&V methodology along these lines can be found in [12].

Acknowledgements. The work on which this paper reports has been funded by the European Commission through the projects MODELPLEX (Contract no. 034081) and NESSoS (Contract no. 256980).

References

1. Ambler, S.W.: The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press, 3rd edn. (2004)
2. Atkinson, C., et al.: Component-based product line engineering with UML. Addison-Wesley (2002)
3. Baker, P., et al.: Model-Driven Testing. Using the UML Testing Profile. Springer (2008)
4. Balsamo, S., Di Marco, A., Inverardi, P.: Model-based performance predictions in software development: A survey. IEEE Transactions on Software Engineering 30(5), 295–310 (2004)
5. Berre, A.J., et al.: COMET – Component and model-based development methodology, version 2.6. SINTEF ICT (2006)
6. Bræk, R., et al.: TIMe: The Integrated Method. Electronic Textbook v4.0. SINTEF (1999)
7. Broy, M.: The 'Grand Challenge' in informatics: Engineering software-intensive systems. IEEE Computer 39(10), 72–80 (2006)
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (1999)
9. D'Souza, D.F., Wills, A.C.: Objects, components and frameworks with UML. The Catalysis approach. Addison-Wesley (1999)
10. Hoare, T., Misra, J.: Verified software: theories, tools, experiments. Vision of a Grand Challenge project. In: Verified Software: Theories, Tools, Experiments (VSTTE), http://vstte.ethz.ch/papers.html (2005)
11. Kruchten, P.: The Rational Unified Process. An introduction. Addison-Wesley, 2nd edn. (2000)

12. Lund, M.S.: Model-based testing with the Escalator tool. Telektronikk 115(1), 117–125 (2009)
13. Mellor, S.J., Clark, A.N., Futagami, T.: Model-Driven Development. IEEE Software 20(5), 14–18 (2003)
14. Runde, R.K., Haugen, Ø., Stølen, K.: The pragmatics of STAIRS. In: 5th International Symposium on Formal Methods for Components and Objects (FMCO'06). Lecture Notes in Computer Science, vol. 4111, pp. 88–114. Springer (2006)
15. Schmidt, D.C.: Model-Driven Engineering. IEEE Computer 39(2), 25–31 (2006)
16. Selic, B.: The pragmatics of Model-Driven Development. IEEE Software 20(5), 19–25 (2003)
17. Special issue on Model-Driven Development. IEEE Software 20(5) (2003)
18. Special issue on model driven software development. IEEE Computer 39(2) (2006)
19. Utting, M., Legeard, B.: Practical Model-Based Testing. A Tools Approach. Morgan Kaufman (2007)
20. Vaandrager, F.: Does it pay off? Model-based verification and validation of embedded systems! In: PROGRESS White Papers 2006. pp. 43–66. STW (2006)
21. Woodcock, J.: First steps in the Verified Software Grand Challenge. IEEE Computer 39(10), 57–64 (2006)