Research Questions for Validation and Verification in the Context of Model-Based Engineering

Catherine Dubois[1], Michalis Famelis[2], Martin Gogolla[3], Leonel Nobrega[4], Ileana Ober[5], Martina Seidl[6], and Markus Völter[7]

[1] ENSIIE, Évry, France
[2] University of Toronto, Canada
[3] University of Bremen, Germany
[4] University of Madeira, Funchal, Portugal
[5] University of Toulouse, France
[6] Johannes Kepler University Linz, Austria
[7] Völter Ingenieurbüro, Heidenheim, Germany

Abstract. In model-based engineering (MBE), the abstraction power of models is used to deal with the ever increasing complexity of modern software systems. As models play a central role in MBE-based development processes, for the adoption of MBE in practical projects it becomes indispensable to introduce rigorous methods for ensuring the correctness of the models. Consequently, much effort has been spent on developing and applying validation and verification (V&V) techniques for models. However, there are still many open challenges.
In this paper, we shortly review the status quo of V&V techniques in MBE and derive a catalogue of open questions whose answers would contribute to successfully putting MBE into practice.

1 Introduction

This paper is based on the discussions of a working group (whose members are mainly the authors of this paper) of the 2013 Dagstuhl seminar number 13182 on *Meta-Modeling Model-Based Engineering Tools*. The working group addressed a panel of important issues with *validation and verification* (V&V) of models and model transformations whose correctness is crucial for the successful realization of model-based engineering (MBE) projects.

Overall, models help to clarify and plan the development of software. Furthermore, they allow the use of methods to ensure quality and discover errors in conceptual designs. However, with the extensive use of models during the software development process, there is the danger of introducing defects into the models. For detecting or avoiding such defects, techniques of validation and verification (V&V) for models help.

Besides models, model transformations (MTs) are at the heart of model-based engineering. Different kinds of MTs underlie the engineering activities and their

associated tools. They can be used for example to refine models, to generate code, etc. This variety of purposes of MTs emphasizes the need for effective and dedicated V&V techniques for animating, testing, and proving them.

In a model-based development process, V&V techniques help to better understand models and to assess model properties which are stated implicitly in the model. V&V serves to inspect models and to explore modeling alternatives. Model validation answers the question 'Are we building the right product?' whereas model verification answers 'Are we building the product right?', similar as V&V does for code. Validation is mainly an activity done by the developer demonstrating model properties to the client, whereas in verification the developer uncovers properties relevant within the development process. Verification is closely related to testing and covers automatic and semi-automatic (static) analysis techniques like theorem proving and model-checking.

In this paper, we shortly review the status quo of V&V techniques in MBE and derive research questions whose answers will push the whole field forward. The methodology behind this paper is as follows. First, we started to list objectives, methods, tools, use cases and then highlighted difficulties, bottlenecks and pitfalls. On this basis, we derived some key research questions. This paper sums up these discussions and lists the different research questions.[8]

The main conclusion of our analysis of the actual situation is that specific V&V methods for models and MTs are required. Although numerous tools exist to verify and validate hardware and software, they do not apply straightforwardly to models and MTs. In [8], E. M. Clarke, E. A. Emerson and J. Sifakis mentioned abstraction techniques as one of the promising paths for the future advances in the field of verification. The use of V&V techniques in the context of MBE follows the direction of abstraction. However V&V for models is concerned by a more complete abstraction mechanism that covers the entire system, whereas in [8], abstraction is considered mostly at mathematical level and targets formal semantics.

The rest of the paper is organized as follows: In Section 2, we reflect on the situation and describe some challenges. Subsequently, the research questions which emerged are listed in Section 3. We conclude in Section 4.

2 Status and Challenges

In this section we describe the main observations regarding the state of theory and practice that informed the discussions of our working group and list proposed solutions found in the literature. The observations are organized thematically, based on the main areas in which V&V and MBE intersect.

2.1 Gap Existing between Models and V&V Formalisms

In the context of model-based development, V&V faces new challenges. The origin of most of them stays in the gap existing between the (mostly high-level)

[8] We are aware that for some of them proposals already exist. Because of lack of space, we are not able to reference them all, and so we cannot reach exhaustiveness.

specification mechanisms, used at design time both for specifying the model and the properties that are subject to V&V, and the underlying V&V engines that use *low-level* formalisms.

The gap between the two specification mechanisms can be (at least partially) filled by using model transformations and traceability mechanisms [11], which are definitely needed in this context. Yet it is only part of the answer as behavioral semantics often leads to non-bijective correspondences between design time and runtime artifacts.

The V&V frameworks, which are typically using back-end tools using specific formalisms need to be complemented with mechanisms allowing the user to interact with the back-end tools through their own modeling languages. They do offer the right environment to reason about the system under modeling. In this context we can mention existing initiatives that go on this direction. In [6] a high-level change-driven transformation language is proposed in order to capture changes, even those that concern low level transformations (as it may be in the case of some feedback (e.g. counter-examples) produced by V&V tools. Supporting the user during the error diagnosis phases can be done e.g. by allowing customizable simulation trace visualization [1].

In the same spirit of hiding the technicalities related to the V&V engine, we can mention here the issue of (semi-)automatically managing the V&V machine configuration. Although not necessarily a research challenge, this is a typical functionality required in order to improve the accessibility to V&V tools.

2.2 Need to Refine Existing Methodologies

V&V tools have the potential of helping the user during model development, by allowing early V&V, similar to debugging facilities offered by programming development environments. Most of the generic methodologies identify some possible points where V&V could be used. In a realistic setting, the model-based development methodology should be refined in order to better identify which V&V activities are meaningful at the various phases of design, in order to take full advantage of the existing V&V engines. Such a methodology would most likely vary depending on the nature of the project and on the application domain, but one can expect that methodology definitions can be capitalized within the same business domain.

2.3 Importance of Snapshots

In the context of V&V, it is important to be able to clearly and completely describe a dynamic model configuration (often called model snapshot, or global state of the system) - in terms of existing objects, values of their respective attributes, existing links, active states in the state machine (if any), content of the input queue, etc. Such a snapshot could correspond to the state of the dynamic model before an error or otherwise identified as meaningful. A particular case of model snapshot is the one specifying the model instantiation, arising in the context of describing the initial state of a model execution. Each commercial

tool that offers model simulation functionalities uses some (more or less documented) mechanism for describing model instantiation, however this snapshot description does not cover arbitrary snapshots.

There are some approaches that tackle the idea of model snapshot, such as for instance, the USE tool [12] which allows the user to generate/verify UML model snapshots. Yet, no modelling tool uses such elaborate techniques.

2.4 Properties

V&V focuses on verifying some properties with different objectives. Properties to be verified differ according to the nature of models (e.g. static or dynamic) and to the stage in the development process (e.g. specification or code generation time). Furthermore there are also limitations and synergies depending on the applied V&V techniques (static analysis, testing, model checking, runtime verification, formal proof). However there is no one-to-one association between properties and techniques, the latter being for the most part complementary.

More generally, we can distinguish structural properties (composition of models, consistency, redundancy), behavioral properties (ensure liveness of a model or safety properties) and also quantitative properties (such as Worst Case Execution Time (WCET) or schedulability). Properties are related with models but also to model transformations. Here the classification is different, it is discussed in more detail in the next subsection. We can also distinguish between static and dynamic properties and in both categories refine into language inherent, model specific, generic or user-defined properties.

We can find in the literature many approaches and tools to verify models. However it is quite confusing and it is difficult to draw a classification allowing the answer to the following question: *What kind of property to verify on which model at what stage with what kind of technique?* Some partial answers exist, e.g. using feature models [10] or ontologies [15].

2.5 Model Transformations

Model transformations (MTs) are an integral part of model-based software engineering. MTs are used for a broad variety of purposes, ranging from applications as diverse as maintaining inter-model consistency to defining the translational semantics of a domain-specific language. This ubiquity of model transformations emphasizes the need for effective V&V techniques, that focus specifically on MTs. This is especially true in contexts of use where MTs are critical, such as code generation. At the same time, V&V for MTs differs from traditional program verification, since MTs are defined at a higher level of abstraction than programs, thus creating opportunities for more effective application of formal techniques. These factors have generated considerable interest in the research community, with special-interest events such as the VOLT workshop ("Verification Of ModeL Transformations") being organized since 2012.

An important challenge in studying the verification of MTs is to understand how verifying MTs is different from traditional program verification. There has

been some work mapping the challenges that are specific to MTs; for example, [4, 5] discuss the challenges of testing MTs, whereas [16] focuses specifically on bidirectional transformations.

In the same vein, it is necessary to understand how to best reuse existing verification tools, techniques and methods. For example, in [2], the autors propose a tri-dimensional approach that takes into account (a) the kind of transformation involved, (b) the properties of interest, and (c) the available verification techniques. The purpose is to identify what verification method is most appropriate based on the transformation and desired properties. Similarly, [10] classifies approaches for verifying transformations based on five criteria: (a) verification goal (e.g. consistency, correctness), (b) representation of the domain, (c) representation used for the verification task, (d) specification language used, and (e) verification technique (theorem proving, static analysis, model checking).

In general, the correctness of MTs is of paramount importance and is generally the ultimate goal of verification of MTs. Correctness is tightly related with specification. The *Tracts* methodology [18] proposes a generalization of model transformation contract, based on the idea of *duck typing* MTs with OCL constraints, while also supporting test-case generation. Another approach is to reuse transformation primitives in order to build correct-by-construction transformations, typically with a trade-off in expressive power, such as the case of DSLTrans [3].

Finally, the identification of properties that are meaningful when verifying MTs is also important. Confluence and termination have been studied extensively for transformations based on graph rewriting [9, 14]. In [2], a more thorough categorization of properties is proposed, marking the distinction between properties related to transformations themselves (e.g. determinism) and properties related to the result of applying the transformation (e.g. conformance of the output). For each of these major categories, several minor sub-categories are identified.

2.6 Informal vs. Formal vs. Incomplete Modeling

Starting from informal use case sketches, which can be connected to formal model elements by trace links, there is need during V&V to concentrate on particular model parts and to switch on or off particular model inherent elements (in class diagrams, e.g., multiplicities, or aggregation and composition restrictions). Furthermore we may desire to explicitly configure constraints by negating, deactivating or activating them (in class diagrams, e.g., class invariants and operation pre- and postconditions; in state charts, e.g., state invariants and transition pre- and postconditions). Such configuration options must be offered with different types of granularity: (a) all model elements may be relaxed, (b) only a manual model element selection can be considered for relaxation, or (c) a semi-automatic element selection for relaxation may be offered (such a semi-automatic selection might depend on a user-determined, editable criteria catalogue). In the ultimate vision one might consider sliders on the user interface of the modeling tool which allows the developer to gradually go from a strict, formal model through various

intermediate levels to a totally relaxed and informal model. However, if formal test cases and test scenarios are desired, then a minimal frame for test case construction must be preserved. Such a minimal frame could consist, e.g., for class diagrams of central classes and associations and for state charts of central states and transitions, with all model elements in the minimal version without any implicit or explicit constraints. Thus, there will be a tradeoff between switching off or ignoring certain model elements (like constraints or classes) and the degree of formality: if more model elements are switched off, the model will become more informal and will accept more scenarios; however, this procedure only works to a certain degree, because one can only formulate scenarios if at least a minimum selection of formal model elements is present; for a completely informal model no formal scenario can be formulated. Thus, ignoring certain model elements or constraints and the degree of formality are closely linked.

In order to handle incomplete and partial models wrt V&V, these model configuration options must be recorded along with the various V&V test scenarios, test cases and their results with respect to the configured model. Test cases and test scenarios must be invoked repeatedly with different configuration options and test results must be recorded in a systematic way. It must be possible to query test results in order to retrieve the proper configuration settings. The mechanisms for playing around with model relaxation can also be employed to allow for model alternatives. Model relaxations and model alternatives will span up a graph of model versions. These model versions must be connected to a graph of proper model test cases and model scenarios.

2.7 Comparison and Benchmarking

In the V&V research area, not only theoretical results are important, but also the tools which implement novel approaches in order to benefit from these results. Expressive benchmarks are necessary to evaluate the maturity of the tools as well as compare the power of novel approaches to established techniques. At dedicated workshops like the Comparison and Versioning of Software Models [17] the benchmarking issue has been a specially discussed topic which is one approach to obtain commonly accepted benchmarks. Often community-organized competitions and evaluations are a valuable source for the benchmarks which serve as basis for evaluations in scientific publications illustrating the benefits of their contributions. Examples of such events are the SAT solving competition [13] or the Transformation Tool Contest (TTC) [7].

For the community the advantages of a centrally organized competition are manifold. First of all, evaluations are performed in an environment equal to all participants, thus allowing a fair comparison of the different tools. Second, the benchmarks are not selected by the tool developers, but by some impartial experts and covers the whole spectrum of interesting test cases. Third, a clear documentation of the outcome is provided such that the experiments performed in the context of a competition are repeatable. By this means the state-of-the-art of a research field becomes publicly available, it becomes clear where progress has been made and where more work has to be done. Ideally, new research questions

are derived from the results which might then be handed over to the community in order to get solutions. These research questions may be documented in terms of benchmarks which cannot be handled by current tools. Often, such benchmarks are provided from industry and allow the researchers to show that a new approach improved the state-of-the-art.

To the best of our knowledge, recently there are no community-organized evaluations and competitions for V&V approaches in the field of MBE (TTC goes in that direction), although such events would be extremely beneficial with respect to the same arguments discussed above. Their absence is somehow surprising as the research community is rather large playing a major role in all leading modeling conferences. However, there are several reasons which might explain why no V&V competitions for MBE-approaches are organized at the moment. In the following, we elaborate on three urgent problems, which have to be overcome for structured V&V research.

(i) *No common standards.* Many approaches use their own metamodel in order to focus on the language element relevant for their purposes as for example UML is too large to be completely implemented.

(ii) *No community platform.* In order to collect and document interesting benchmarks which are required for organizing a successful competition, the community needs a forum to gather relevant data as it is for example done with the TPTP repository.

(iii) *Metrics for improvements.* In automatic theorem proving the improvements are mostly measured by runtime reduction or by compactness of proofs. For V&V approaches it is not so easy to measure progress in tools: the size of encodings or the execution time could be compared as well as the size of the error traces. However, MBE tools are also confronted with other requirements like usability and this is much harder to evaluate.

2.8 Domain-Specific Languages

Most verification tools have quite specific, sometimes archaic and hard to use input languages that are optimized for the semantic paradigm used by the tool. These languages are often alien to standard developers and hence, verification tools are often not used. On the other hand, developers could benefit from easy-to-use verification techniques; even in non safety-critical domains, verification can be an additional means of quality assurance, even when it is not required by industry standards. Development approaches based on domain-specific languages (DSLs) are becoming more and more mainstream, through code generation this approach leads to improved productivity. Models expressed with DSLs also have the potential of simplifying analysis and verification, because of the higher degree of domain semantics they express. There seems to be a potential to exploit the two approaches synergistically: from the high-level models, we can automate the generation of the input to the verification tools.

3 Research Questions

In this section we list the research questions that emerged from the discussions of our working group. The purpose of the list is to document the important issues that, we believe, research in the intersection of MBE and V&V must address.

Gap Existing between Models and V&V Formalisms: How do we express properties at the level of models in a way understandable to clients? How do we formulate models and properties in a single language transparent to clients? How do we report the V&V results and diagnostics in an appropriate form to clients? How do we bridge the gap between formally expressed and verified properties on one side and client attention on the other side? Can modeling language extensions help in making explicit the needs of V&V machines?

Need to Refine Existing Methodologies: How do we integrate V&V in the overall development and modeling process? On the technical level of tool exchange? On the methodological level of using the right technique at the right time for the right task? When are techniques like animation, execution, symbolic evaluation, testing, simulation, proving or test case generation used efficiently during development?For which model and model transformation properties can they be employed?

Design-time vs. Runtime: How do we obtain during the V&V phase an initial model instantiation on the model runtime level which is determined by the model design time description? How do we obtain large and meaningful instantiations? How do we connect design time and runtime artifacts? How do we deal with the scalability issue in the context of V&V ? How do we handle time and space concerns wrt design time and runtime artifacts? How do we automatically or semi-automatically manage the V&V machine configuration?

Properties: How do we handle model and model transformation properties relevant in V&V like consistency, reachability, dependence, minimality, conformance, safety, liveness, deadlock freeness, termination, confluence, correctness? How do we search for such properties in models and model transformations? What are the benefits and tradeoffs between expressing these properties on more abstract modeling levels in contrast to expressing them on more concrete levels? How do we find the right techniques for uncovering static and dynamic model properties? Which techniques are appropriate for uncovering static modeling language inherent properties, which for static model-specific properties? Which techniques are appropriate for uncovering dynamic generic properties, which for dynamic model-specific properties? Which high-level features are needed in the property description language in order to query and to determine modeling level concepts?

Model Transformation: What verification techniques are meaningful for verifying model transformations? How do we analyse properties like confluence and termination for transformations which are composed from transformation units? How do we analyse correctness of model transformations wrt a

transformation contract? How do we infer a transformation contract from a model transformation?

Informal vs. Formal vs. Incomplete Modeling: How do we leverage informal assumptions found in sketches for exploratory V&V? Are informal sketches close enough to V&V at all? What are appropriate relaxation mechanisms for different degrees of formality? How do we handle incomplete or partial models wrt V&V? How do we deactivate and activate model units? How do we handle the exploration of model properties and alternatives?

Comparison and Benchmarking: How do we compare existing V&V tools employed for modeling wrt functionality, coverage, scalability, expressiveness, executing system (i.e., for models at runtime)? Which criteria are appropriate for comparison? Can the broad and diverse spectrum of V&V machines (like B, Coq, HOL/Isabelle, SAT, SMT, CSP solvers, Relational logic and enumerative techniques) be globally compared in a fair way at all?

Domain-Specific Languages: How can DSLs be defined so that they are close to the domain concepts on the one hand, but still allow the generation of meaningful input files for verification tools? How do we express the properties to be verified at the domain level in a user-friendly way? Can the property specifications be integrated with the same DSL and/or model used for describing the to-be- verified system without creating self-fulfilling prophecies? How can we lift the result of a verification (e.g. an example program execution that demonstrates the failure) back to the domain level and express it in terms of the DSL-level input? Can incremental language extensions help with making programs expressed in general-purpose languages more checkable? For example, the semantics of a specific extension construct may enable the generation of very rich inputs to the verification tool, which otherwise may have to be specified manually (program annotations or properties)?

4 Conclusion

On this paper we present some observations and list the research questions that emerged from them. This list of research questions spans a wide area of themes where MBE and V&V intersect: specification and feedback and its impact on stakeholder collaboration, development process, design-time vs runtime, properties, model transformations, informal vs formal vs incomplete modeling, comparison and benchmarking, and domain-specific languages.

Our hope is that the set of observations and questions presented here will spark further discussions and thus aid the community focus research on those areas where the synergy of MBE and V&V can yield the greatest benefits.

References

1. El Arbi Aboussoror, Ileana Ober, and Iulian Ober. Seeing Errors: Model Driven Simulation Trace Visualization. In *Proc. of the Int. Conf. on Model Driven Engineering Languages and Systems (MODELS'12)*, volume 7590 of *LNCS*, pages 480–496. Springer, 2012.

2. Moussa Amrani, Levi Lucio, Gehan Selim, Benoit Combemale, Jürgen Dingel, Hans Vangheluwe, Yves Le Traon, and James R. Cordy. A Tridimensional Approach for Studying the Formal Verification of Model Transformations. In *Proc. of the IEEE Fifth Int. Conf. on Software Testing, Verification and Validation (ICST'12)*, pages 921–928. IEEE Computer Society, 2012.
3. Bruno Barroca, Levi Lúcio, Vasco Amaral, Roberto Félix, and Vasco Sousa. DSLTrans: A Turing Incomplete Transformation Language. In *Software Language Engineering*, volume 6563 of *LNCS*, pages 296–305. Springer, 2011.
4. Benoit Baudry, Trung Dinh-trong, J.-M. Mottu, Devon Simmonds, Robert France, Sudipto Ghosh, Franck Fleurey, and Yves Le Traon. Model transformation testing challenges. In *Proc. of the IMDT workshop @ ECMDA06*, 2006.
5. Benoit Baudry, Sudipto Ghosh, Franck Fleurey, Robert France, Yves Le Traon, and Jean-Marie Mottu. Barriers to systematic model transformation testing. *Commun. ACM*, 53(6):139–143, June 2010.
6. Gábor Bergmann, István Ráth, Gergely Varró, and Dániel Varró. Change-driven model transformations - change (in) the rule to rule the change. *Software and System Modeling*, 11(3):431–461, 2012.
7. Pieter Van Grop Christian Krause, Louis Rose. Transformation tool contest 2013. `http://planet-sl.org/ttc2013`.
8. Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, 2009.
9. Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of algebraic graph transformation*, volume 373. Springer, 2006.
10. Sebastian Gabmeyer, Petra Brosch, and Martina Seidl. A Classification of Model Checking-Based Verification Approaches for Software Models. In *Proceedings of VOLT'13*, 2013.
11. Ismênia Galvão and Arda Goknil. Survey of traceability approaches in model-driven engineering. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 313–326. IEEE Computer Society, 2007.
12. Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34, 2007.
13. Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The International SAT Solver Competitions. *AI Magazine*, 33(1), 2012.
14. Jochen M. Küster. Definition and validation of model transformations. *Software and Systems Modeling*, 5(3):233–259, 2006.
15. Mounira Kezadri and Marc Pantel. First steps toward a verification and validation ontology. In *Proc. of Int. Conf. on Knowledge Engineering and Ontology Development (KEOD'10)*, pages 440–444, 2010.
16. Perdita Stevens. Generative and transformational techniques in software engineering ii. chapter A Landscape of Bidirectional Model Transformations, pages 408–424. Springer, 2008.
17. Jan Oliver Ringer Udo Kelter, Piet Pietsch. Comparison and versioning of software models. `http://pi.informatik.uni-siegen.de/CVSM2013/`.
18. Antonio Vallecillo, Martin Gogolla, Loli Burgueño, Manuel Wimmer, and Lars Hamann. Formal specification and testing of model transformations. In *Proc. of the 12th Int. Conf. on Formal Methods for the Design of Computer, Communication, and Software Systems: formal methods for model-driven engineering (SFM'12)*, pages 399–437. Springer, 2012.