System Models vs. Test Models - Distinguishing the Undistinguishable?

Stephan Weißleder and Hartmut Lackner

{stephan.weissleder|hartmut.lackner}@first.fraunhofer.de

Abstract: Models are purposeful abstractions used in todays software engineering processes. Experience shows that the application of models reduces engineering costs, allows for easy reuse, and improves product quality. Models can be used for several purposes at various levels of abstraction. In this paper, we discuss the differences of system models and test models. For that, we consider several aspects of models and investigate if they can be used as distinctive features of system models and test models.

1 Introduction

Models are the essence of many engineering processes: People create models to understand, analyze, transform, and develop systems. There are several corresponding engineering processes, e.g., model-driven engineering [Ken02] or the Model-Driven Architecture (MDA) [Obj09] defined by the Object Management Group (OMG). The general idea is to create models at an abstract level, to refine them in a step-wise manner using model transformations, and to generate (a part of) the whole system.

Models are used, e.g., to describe systems and tests. Correspondingly, they are referred to as *system models* and *test models*. The best possible use of these models is the automatic generation of the system or the test suite. In this paper, we investigate several aspects of models with the aim to identify distinctive features of system models and test models.

This paper is structured as follows. We present the related work in Section 2. In Section 3, we describe several objective aspects and investigate if they can be used to distinguish system models and test models. We conclude and discuss our findings in Section 4.

2 Related Work

Models for software engineering have been investigated for decades [Har87, Obj07]. They can be used for system creation [Küs06] or testing [UPL06, UL06]. There are only a few comparisons of system models and test models. For instance, Malik et al. [MJV+10] state that test models can only be used for testing. In contrast, one of our findings is that test models can also be used for implementation. Furthermore, Utting and Legeard [UL06] state that test models are usually more abstract than system models. There are, however, no urgent grounds for it (see Section 3.6). In [GNRS09], the authors state that system models and test models are two different views on the same system - both models can be related using a holistic model. In addition, we describe that both models can be used for the same actions. Utting et al. [UPL06] present a taxonomy of model-based testing, in which they consider several aspects of test models. In contrast, our approach is focused on comparing system models and test models. Furthermore, we also consider a broader range of model aspects and do not restrict our considerations to model-based testing.

3 Aspects of Models

In this section, we present aspects of models and investigate them with respect to their use as a distinctive feature for system models and test models. Figure 3 shows the investigated aspects: On the left side, known characteristics are shown as presented in [UPL06]. On the right side, we present additionally identified aspects.

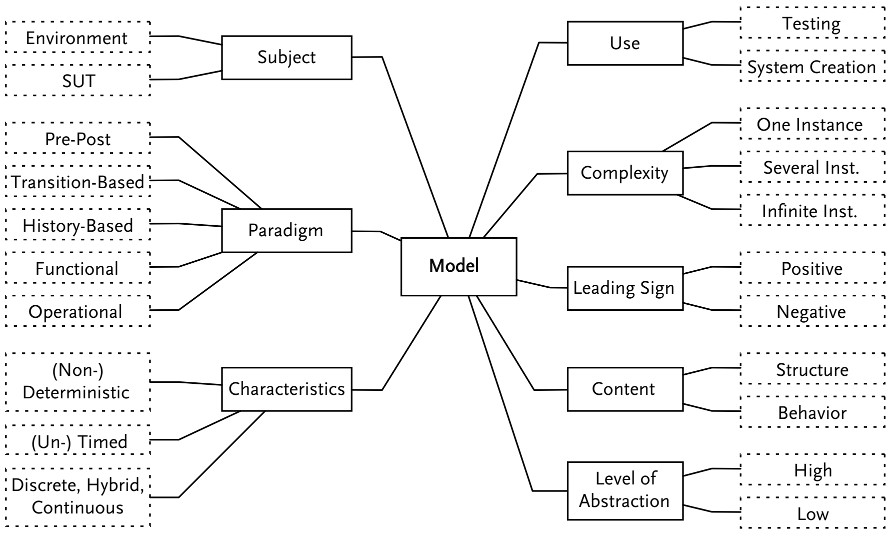

Figure 1: Model aspects.

In this paper, we focus on subject in Section 3.1, use in Section 3.2, complexity in Section 3.3, leading sign in Section 3.4, content in Section 3.5, and level of abstraction in Section 3.6. We leave the remaining aspects to future work.

3.1 Subject

Models are abstractions of an artifact - their subject. This subject is usually (a part of) a concrete system or its environment. Here, we investigate if system models or test models are fixed to one of these subjects.

Constructive system models are often used to create the system and, thus, describe the system. However, system models can also describe the interaction with other components by specifying the expected behavior as observed from outside. Thus, system models can describe the system as well as its environment.

Test models can also describe the system or the environment: The straight forward approach to model tests is to describe test cases, i.e. part of the system environment, at a higher level of abstraction, e.g. using activity diagrams of the Unified Modeling Language (UML). In contrast, tests can also be derived from descriptions of the expected system behavior. Several tools support test generation from system models [Sma, Con, Mic09].

As a result, both system models and test models can be (and actually are) used to describe both the system or its environment.

3.2 Use

Models are intentional abstractions. They are designed for a specific purpose. In this section, we investigate if test models are restricted to be used for testing and if system models are restricted to be used for system design.

System models describe the system and are intended to be used for system creation. They can also be used for testing. For instance, several model-based test tools use system models for automatic test generation [Sma, Con, Mic09].

Test models are designed for testing, i.e. describe tests of a system. A test contains input stimuli and expected system behavior to describe interaction sequences with the system, e.g. using sequence diagrams [Obj07]. Additionally, such sequence diagrams are often attached to use cases that are used for system design. Correspondingly, test descriptions like sequence diagrams can also be used for system design.

As a result, both system models and test models can be used for system design and system test. This is not surprising: Both kinds of models contain information. This information can be used for system implementation or testing.

3.3 Complexity

Models can describe structure or behavior. For instance, the UML [Obj07] defines several structural models, e.g., class diagrams, and behavioral models, e.g., state machines. Structural and behavioral models can allow exactly one possible instance, more than one but a finite number of instances, and an infinite number of instances. For instance, multiplicities in class diagrams describe the number of possible related objects. The number of loops, the value type of parameters, etc. determine the number of behaviors in state machines.

Both system models and test models can be of arbitrary complexity. A large number of models with a low complexity can have a similar expressiveness as a small number of models with a high complexity. Examples are the following: State machines of high complexity often describe system behavior - they are used for system design and system test. While systems are often easy to generate from state machines, the generation of tests that cover all aspects is undecidable. Less complex behavioral models are often used to describe single tests or to describe the meaning of a use case in system design. Although there may be different challenges for system design and test design based on the complexity of the used model, system models and test models can be of arbitrary complexity in general.

3.4 Leading Sign

Models usually describe the expected (positive) structure or behavior of a system. Additionally, undesired structure or behavior can be described in negative models.

System specifications are based on positive models describing the intended functionality. The opposite, i.e. designing a system solely on negative models, is cumbersome and usually impossible. As a consequence, negative models are used in addition to positive models [SO05, vL04]. For security-relevant systems like, e.g., firewalls, the description of unwanted behavior is of utmost importance for system design.

Most test generators use positive test models to generate tests cases [Sma, Con, Mic09, Wei, GNRS09]. These test generators are constrained to test the specified functionalities of a system. Testing if unwanted system functionalities are prohibited is likewise important. The authors are aware of a comparatively small number of such approaches, e.g., deriving unspecified behavior from positive test models [KL09].

Both system models and test models can have positive or negative leading sign.

3.5 Content

Models can describe behavior or structure. There are various languages for behavioral and structural models. For instance, the UML 2.1 defines six structural diagrams and seven behavioral diagrams. There are other languages like, e.g., Petri Nets, B, Z, or AMSs.

The system model is often based on structure. For instance, structural elements like classes are defined before behavior can be assigned to them, e.g., using state machines. There are, however, also functionality-focused approaches that abstract from structural details using „the system“ as the only structure. Structural as well as behavioral information is crucial for system development. Thus, system models can and should describe both.

Testing can be focused on behavior [UL06] and on structure [SW08]. Thus, test models can be focused on describing behavior or structure, too. The border between structural and behavioral models, however, is not always sharp. For instance, behavioral tests can also be derived from class diagrams that contain operations with pre- and postconditions.

We conclude that system models as well as test models contain both behavior and structure.

3.6 Level of Abstraction

Models can reside at various levels of abstraction regarding detail and completeness. For instance, models can describe a wide range of artifacts from abstract ones like use cases via state machines to concrete models, e.g., containing source code or machine code.

System models can reside at almost every level of detail abstraction. Process models like the MDA [Obj09] prescribe that the process starts with abstract models that are refined in a step-wise manner until source code can be derived. Regarding completeness, system models can range from a complete description to partial descriptions that do not consider, e.g., already implemented library functions.

Like system models, test models can reside at all levels of abstraction regarding detail and completeness: Test models can also be refined in a step-wise manner. They can also be focused on a certain part of the system. Test models are often assumed to be more abstract than system models [UL06]. This, however, always depends on the intended usage: For testing general behavior, implementation details are rather unimportant. For security testing, these details are focused on, e.g., to prevent buffer overflows.

As a result, system models as well as test models can reside at any level of abstraction.

4 Conclusion and Discussion

The focus of our paper is the general distinguishability of system models and test models. We investigated several model aspects with respect to this issue. Our findings are that none of them could be used as a distinctive criterion for system models and test models, i.e. a model cannot be clearly identified as a system model or a test model.

There are several points to discuss. For instance, we investigated the impact of the considered aspects at a formal level and neglected common practices and experiences. Furthermore, the considered aspects were identified based on our experiences. There may be other aspects that provide a distinction for system models and test models. However, our investigations show at least that the presented aspects do not provide a distinction. There may be other applications like, e.g., test generation algorithms, for which it is important if the considered model describes the system or the test.

Acknowledgment. This work was supported by grants from the BMBF (Bundesministerium für Bildung und Forschung) via innovation alliance SPES2020.

References

[Con] Conformiq. Qtronic. http://www.conformiq.com/.

[GNRS09] Helmut Götz, Markus Nickolaus, Thomas Roßner, and Knut Salomon. iX-Studie: Modellbasiertes Testen. http://www.heise.de/kiosk/special/ixstudie/09/01/, 2009.

[Har87] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[Ken02] Stuart Kent. Model Driven Engineering. In *IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods*, pages 286–298, London, UK, 2002. Springer-Verlag.

[KL09] Kathrin Kaschner and Niels Lohmann. Does my service have unspecified behavior? In Oliver Kopp and Niels Lohmann, editors, *Proceedings of the 1st Central-European Workshop on Services and their Composition, ZEUS 2009, Stuttgart, Germany, March 2-3, 2009*, volume 438 of *CEUR Workshop Proceedings*, pages 22–28. CEUR-WS.org, March 2009.

[Küs06] Jochen M. Küster. Definition and Validation of Model Transformations. *Software and Systems Modeling*, V5(3):233–259, 2006.

[Mic09] Microsoft Research. SpecExplorer. http://research.microsoft. com/en-us/projects/SpecExplorer/, 2009.

[MJV$^+$10] Qaisar A. Malik, Antti Jaaskelainen, Heikki Virtanen, Mika Katara, Fredrik Abbors, Dragos Truscan, and Johan Lilius. Model-Based Testing Using System vs. Test Models - What Is the Difference? *Engineering of Computer-Based Systems, IEEE International Conference on the*, 0:291–299, 2010.

[Obj07] Object Management Group. Unified Modeling Language (UML), version 2.1. http://www.uml.org, 2007.

[Obj09] Object Management Group. Model Driven Architecture (MDA). http://www.omg.org/mda, 2009.

[Sma] Smartesting. Test Designer. http://www.smartesting.com.

[SO05] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.

[SW08] Daniel A. Sadilek and Stephan Weißleder. Testing Metamodels. In Alan Hartman Ina Schieferdecker, editor, *ECMDA'08: European Conference on Model Driven Architecture*. Springer, June 2008.

[UL06] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[UPL06] Mark Utting, Alexander Pretschner, and Bruno Legeard. A Taxonomy of Model-Based Testing. Technical Report 04/2006, Department of Computer Science, The Universiy of Waikato (New Zealand), 2006.

[vL04] Axel van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 148–157, Washington, DC, USA, 2004. IEEE Computer Society.

[Wei] Stephan Weißleder. ParTeG (Partition Test Generator). http://parteg.sourceforge.net.