Virtual Verification of System Designs against System Requirements

Wladimir Schamai, Philipp Helle, Peter Fritzson, and Christiaan J.J. Paredis

[1] EADS Innovation Works, Germany wladimir.schamai@eads.net
[2] EADS Innovation Works, UK philipp.helle@airbus.com
[3] Department of Computer and Information Science, Linköping University, Sweden petfr@ida.liu.se
[4] Georgia Institute of Technology, Atlanta, USA chris.paredis@me.gatech.edu

Abstract. System development and integration with a sufficient maturity at entry into service is a competitive challenge in the aerospace sector. With the ever-increasing complexity of products, this can only be achieved using efficient model-based techniques for system design as well as for system testing. However, natural language requirements engineering is an established technique that cannot be completely replaced for a number of reasons. This is a fact that has to be considered by any new approach. Building on the general idea of model-based systems engineering, we aim at building an integrated virtual verification environment for modeling systems, requirements, and test cases, so that system designs can be simulated and verified against the requirements in the early stages of system development. This paper provides a description of the virtual verification of system designs against system requirements methodology and exemplifies its application in a ModelicaML modeling environment.

Keywords: Requirements, Verification, ModelicaML, Modelica, MBSE, Model-based testing

1 Introduction

The ever-increasing complexity of products has had a strong impact on time to market, cost and quality. Products are becoming increasingly complex due to rapid technological innovations, especially with the increase in electronics and software even inside traditionally mechanical products. This is especially true for complex, high value-added systems such as aircraft and automobile that are characterized by a heterogeneous combination of mechanical and electronic components. The economic aspects of electronic subsystems (Embedded Systems) running within these products are remarkable. For example, avionics costs are about 30% of the overall cost of an aircraft and embedded systems represent about 40% of avionics cost. An important component of embedded systems is embedded software whose importance is rising almost exponentially in time: From the 1980s Airbus A300 which had a couple of thousand lines of software code on board, to the A380 whose software size is in the range of millions of

lines. For this aircraft, a single line of software code certified according to DO-178b level A is estimated to cost about 100 € thus yielding an overall cost for software of hundreds of millions of Euros. System development and integration with sufficient maturity at entry into service is a competitive challenge in the aerospace sector. Major achievements can be realized through efficient system specification and testing processes. Limitations of traditional approaches relying on textual descriptions are progressively addressed by the development of model-based systems engineering[1] (MBSE) approaches. Building on this general idea of MBSE, we aim at building a virtual verification environment for modeling systems, requirements and test cases, so that a system design can be simulated and verified against the requirements in the early system development stages.

1.1 Scope

For our methodology we assume that the requirements from the customer have been elicited[2] as requirement statements according to common standards in terms of quality, e.g. according to Hull et al.[4] stating that the individual requirements should be unique, atomic, feasible, clear, precise, verifiable, legal, and abstract, and the overall set of requirements should be complete, non-redundant, consistent, modular, structured, satisfied and qualified. The methods to achieve this have been well defined and can be considered to be established. Furthermore, the overall MBSE approach to system design, that is the development of a system design model from textual requirements, is not within the scope of this paper[3].

Paper structure: First we establish and describe the idea of virtual verification of system designs against system requirements (Section 2). Then we present background information on ModelicaML and the running example (Section 3) before we will explain the methodology in detail with the help of said running example (Section 4). Finally, we close with a summary of the current status and propose a number of ideas for future research (Sections 5 and 6).

2 Virtual Verification of System Designs Against System Requirements

This chapter provides the motivation behind our work, a general description thereof and the benefits of using the virtual verification of system design against system requirements (vVDR) approach. Furthermore, related work is discussed.

[1] The International Council on Systems Engineering (INCOSE) defines MBSE as follows: "Model-based systems engineering (MBSE) is the formalized application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases"[1].

[2] A description of the various requirement elicitation, i.e. capturing, techniques can be found in [2] and [3].

[3] The interested reader can find a detailed overview of existing solutions for that in [5] and [6].

2.1 Objectives

A number of studies have demonstrated that the cost of fixing problems increases as the lifecycle of the system under development progresses. As an example Davis[7] reports the following well-known relative cost of repairs for software[4]:

Lifecycle phase	*Relative cost of repair*
Requirements	0.1-0.2
Design	0.5
Coding	1.0
Unit test	2.0
Acceptance test	5.0
Maintenance	20.0

Fig. 1. Relative cost of repair for fixing defects in different lifecycle phases [7]

Thus, the business case for detecting defects early in the life cycle is a strong one. Testing thus needs to be applied as early as possible in the lifecycle to keep the relative cost of repair for fixing a discovered problem to a minimum. This means that testing should be integrated into the system design phase so that the system design can be verified against the requirements early on. To enable an automatic verification of a design model against a given set of requirements, the requirements have to be understood and processed by a computer. MBSE typically relies on building models that substitute or complement the textual requirements. Links between the model elements and the textual requirements are usually kept at the requirements' granularity level, meaning that one or more model elements are linked to one requirement. This granularity is good enough for basic traceability and coverage analysis but fails when an interpretation of a requirement's content by a computer is necessary. There is research concerning the automatic translation of natural language requirements into behavioral models to support the automation of system and acceptance testing (see e.g. [8]) but it is not widely adopted in industrial practice[9]. Formal mathematical methods may be used to express requirements, but their application requires high expertise and, hence, they are not very common in industrial practice. A recent survey came to the conclusion that "in spite of their successes, verification technology and formal methods have not seen widespread adoption as a routine part of systems development practice, except, arguably, in the development of critical systems in certain domains."[10]. The bottom line is that natural language is still the most common approach to express requirements in practice[9]. We want to provide a solution to the question of how to formalize requirements so that they can be processed and evaluated during system simulations in order

[4] Other researchers have found different absolute values but they have all found the same trend.

to detect errors or inconsistencies in a way that is easy to understand and to apply.

2.2 vVDR Concept

Figure 2 depicts the relationship of the various engineering artifacts in the frame of vVDR.

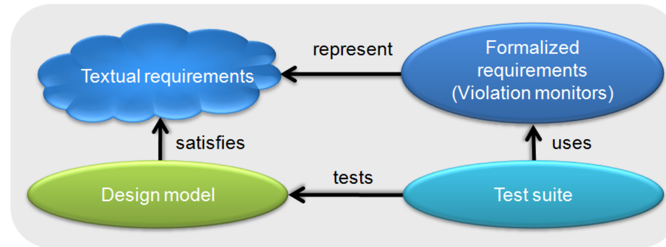

Fig. 2. Engineering data relations overview

A subset of a given set of textual requirements is selected and formalized into so-called requirement violation monitors by identifying measurable properties addressed in the requirement statement. A requirement violation monitor is basically an executable model for monitoring if the constraints expressed by the requirement statement are adhered to. To test a given design model, the requirement violation monitors are linked to the design model using explicit assignment statements. Furthermore, a test suite consisting of a test context and a number of test cases has to be built manually. The test suite uses the formalized requirements as test oracles for the test cases, i.e., if a requirement is violated during a test, the test case is deemed failed. The separation of requirement and system design modeling provides a degree of independence that ensures a high fidelity in the testing results. The test cases, requirement violation monitors and the design model can be instantiated and run automatically. Visual graphs (e.g. plots) allow the monitoring of the requirement violation monitors during run-time to see if the design model fails to implement a requirement.

2.3 Benefits

Our approach contributes to three main steps in the system development lifecycle: requirements analysis, system design and system testing. Experience shows that the main benefit of modeling in general is a contribution to the identification of ambiguities and incompleteness in the input material. Even though we assume that the textual requirements that are provided as an input to the process adhere to a high quality standard, vVDR enables the requirements analyst to further improve the quality by modeling the requirements in a formal

representation as this forces a detailed analysis of the requirements. The main contribution of vVDR is to the quality of the system design. The automatic verification of a design model based on the formalized requirements allows the detection of errors in the system design. The separation of requirements modeling and design modeling allow a reuse of the requirements for the verification of several alternative system designs. Furthermore, even for one design model the same requirements violation monitors can be instantiated several times. As described in [11], the benefits of using a model-based testing approach during the system design phase facilitates error tracing and impact assessment in the later integration and testing stages by providing a seamless traceability from the initial requirements to test cases and test results. Furthermore, it allows reusing the artifacts from the engineering stage at the testing stage of the development cycle which results in a significant decrease in overall testing effort. By integrating the requirements model in a test bench the test models can also be reused for hardware-in-the-loop test setups.

2.4 Related work

In [12] an approach to the incremental consistency checking of dynamically definable and modifiable design constraints is presented. Apart from focusing on design constraints instead of design requirements which can be argued as being a marginal issue, the main difference to vVDR is that the constraints are expressed using the design model variables whereas our approach is based on a separation of the requirements and the design model. Only for a specific test context are they connected using explicit assignment statements. Additionally, the monitoring of model changes and the evaluation of the defined constraints is done by a separate "Model Analyzer Tool" whereas our approach relies on out-of-the-box modeling capabilities. The Behavior Modeling Language (BML) or more specifically the Requirement Behavior Tree technique that is a vital part of the BML is another method for formalizing requirements into a form that can be processed by computers[13][14]. But whereas vVDR relies on a separation between the set of independent requirements that are used to verify a design model and the building of a design model by the system designer, the BML methodology merges the behavior trees that each represent single requirements into an overall design behavior tree (DBT). In other words, the transition from the requirements space to the solution space is based on the formalized requirements.

3 Background

This chapter provides background information on the graphical modeling notation ModelicaML [15] and its underlying language Modelica [16] which was used to implement our approach, and introduces the running example that will be used to illustrate the vVDR methodology in Section 4.

3.1 Technical Background

Modelica is an object-oriented equation-based modeling language primarily aimed at physical systems. The model behavior is based on ordinary and differential algebraic equation (OAE and DAE) systems combined with difference equations/discrete events, so-called hybrid DAEs. Such models are ideally suited for representing physical behavior and the exchange of energy, signals, or other continuous-time or discrete-time interactions between system components.

The Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering and the Systems Modeling Language (SysML) is an adaptation of the UML aimed at systems engineering applications. Both are open standards, managed and created by the Object Management Group (OMG), a consortium focused on modeling and model-based standards.

The Modelica Graphical Modeling Language is a UML profile, a language extension, for Modelica. The main purpose of ModelicaML is to enable an efficient and effective way to create, visualize and maintain combined UML and Modelica models. ModelicaML is defined as a graphical notation that facilitates different views (e.g., composition, inheritance, behavior) on system models. It is based on a subset of UML and reuses some concepts from SysML. ModelicaML is designed to generate Modelica code from graphical models. Since the ModelicaML profile is an extension of the UML meta-model it can be used as an extension for both UML and SysML[5].

3.2 Running Example: Automated Train Protection System

In this section we introduce an example, which will be used in the remainder of this paper to demonstrate the vVDR approach. It is based on the example from[13]. Most railway systems have some form of train protection system that uses track-side signals to indicate potentially dangerous situations to the driver. Accidents still occur despite a train protection system when a driver fails to notice or respond correctly to a signal. To reduce the risk of these accidents, Automated Train Protection (ATP) systems are used that automate the train's response to the track-side signals. The ATP system in our example design model has three track-side signals: proceed, caution and danger. When the ATP system receives a caution signal, it monitors the driver's behavior to ensure the train's speed is being reduced. If the driver fails to decrease the train's speed after a caution signal or the ATP system receives a danger signal then the train's brakes are applied. The textual requirements for the ATP can be found in Appendix A. Figure 3 shows the top-level architecture of the system consisting of a driver, a train and train tracks, and the internal architecture of the train consisting of an HMI system, a control system, an engine and a braking system. The behavior of each of the components is modeled in ModelicaML.

[5] SysML itself is also a UML Profile. All ModelicaML stereotypes that extend UML meta-classes are also applicable to the corresponding SysML elements.

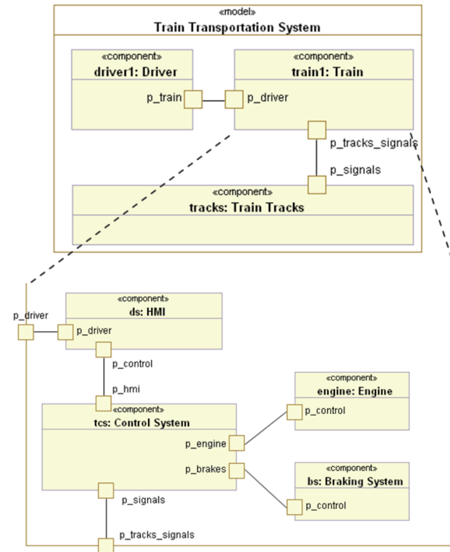

Fig. 3. Train transportation system and train architecture in ModelicaML

4 Methodology Description

Figure 4 provides an overview of the envisaged vVDR process and includes a mapping of the identified activities to the executing roles. The following subsections contain a description of the method steps and illustrate the methodology using our running example.

4.1 Method Step: Select Requirements to Be Verified

From the set of agreed input requirements the requirements analyst selects requirements that are to be verified by means of simulation. The selection criteria depend on the requirement types as well as on the system design models that are planned to be created. Generally speaking, the requirements analyst needs to decide if the vVDR approach is suitable to test a given requirement. This step requires a close collaboration between the requirements analyst and the system designer. The output of this activity is a selected subset of the input requirements. This activity contributes to the system design modeling by clarifying the level of detail that is required of the model for an automatic evaluation of the selected requirements. For example, the requirements 001, 001-2 and 002 would not be selected because appropriate models will be missing or simulation is not best suited[6] for their verification. In contrast, the requirements 003-009 are good candidates for the verification using simulations. The recommended procedure for the selection of requirements is as follows:

[6] For example, design inspection could be sufficient.

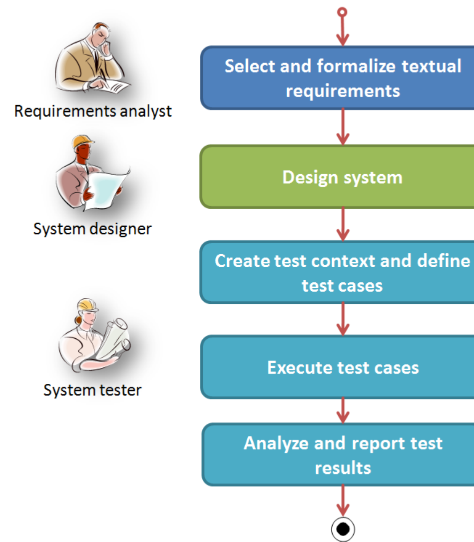

Fig. 4. Methodology overview

– Read a requirement
– Decide if this requirement can and shall be evaluated using a simulation model
– Decide if this requirement is complete, unambiguous and testable by using the kind of design models that are to be created
– If yes: Mark this requirement as selected
– If no: Skip this requirement

The selected subset of requirements will then be transferred into the modeling tool and used in the subsequent steps.

4.2 Method Step: Formalize Textual Requirements

The second step is to formalize each requirement in order to enable its automatic evaluation during simulations. Consider requirement 006-1: "If at any time the controller calculates a "caution" signal, it shall, within 0.5 seconds, enable the alarm in the driver cabin." Based on this statement we can:

– Identify measurable properties included in the requirement statement, i.e., the reception of a caution signal, the activation of the alarm and the time frame constant,
– Formalize properties as shown in Fig. 5 and define a requirement violation monitor as illustrated in Fig. 6.

In order to determine if a requirement is fulfilled the following assumption is made: A requirement is implemented in and met by a design model as long as

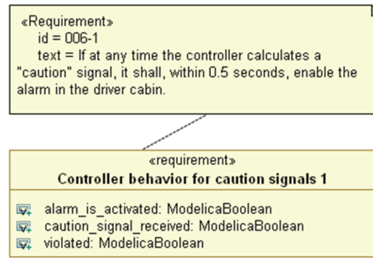

Fig. 5. Formalized requirement properties in ModelicaML

its requirement violation monitor is evaluated but not violated. Now the violation relations can be defined. This example uses a state machine[7] (as shown in Fig. 6) to specify when the requirement is violated. In general, it is recommended to

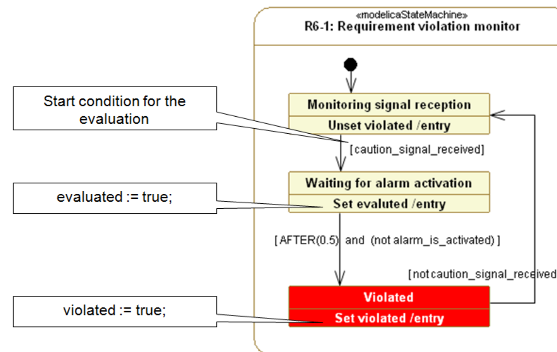

Fig. 6. Requirement violation monitor example

create the following attributes for each requirement:

- evaluated: Indicates if the requirement was evaluated at least once,
- violated: Indicates if this requirement was violated at least once.

The evaluated attribute is necessary, because, while a violation during a simulation provides sufficient indication that a requirement is not met, a non-violation is not enough to ensure the satisfaction of a requirement. For example, if the value of "caution_signal_received" is never true during a particular test case simulation this can mean that:

- This requirement is not addressed by the design (i.e., the caution signals are not received by the controller at all),

[7] A ModelicaML state machine is one possible means to express the violation of a requirement. It is also possible to use other formalisms, equations or statements for it.

– Or this requirement is not verified by this test case because the test case does not provide appropriate stimuli for the design model.

This method step supports the requirements analyst in improving the quality of the selected requirements by identifying ambiguities or incompleteness issues. For example, the following questions were raised when formalizing the original input requirement:

– "If a caution signal is returned to the ATP controller then the alarm is enabled within the driver's cab. Furthermore, once the alarm has been enabled, if the speed of the train is not observed to be decreasing then the ATP controller activates the train's braking system."
– What does decreasing mean, by which rate?
– The driver will need time to react, how much?
– The controller cannot activate things instantaneously. How much time is allowed at the most to pass between the stimuli and the expected result?

Any issues that are identified in this step have to be resolved with the stakeholders and all affected textual requirements have to be updated accordingly. In our example, the updated version of this requirement has been split into two separate requirements 006-1 and 006-2.

4.3 Method Step: Select or Create Design Model to Be Verified against Requirements

The actual system design is not in the scope of this paper. The system designer builds a design model for each design alternative that he comes up with[8]. Since the requirements are kept separate from the design alternatives, the same requirements can be reused to verify several designs, and the same requirement violation monitors can be reused in multiple test cases.

4.4 Method Step: Create Test Models, Instantiate Models, Link Requirement Properties to Design Model Properties

After the formalization of the requirements and the selection of one design model for verification, the system tester starts creating test models, defining test cases and linking requirement properties to values inside the design model. The recommended procedure is as follows:

– Define a test model that will contain test cases, a design model, the requirements and their requirement violation monitors.
– Define test cases for evaluating requirements. One test case can be used for evaluating one or more requirements.

[8] For ease of use, the design will normally be modelled in the same notation and the same tool as the requirements. However, it can be imagined to build interfaces to executable models that were built using different modelling notations in different tools and then subsequently use vVDR to test these models.

- Create additional models if necessary, for example, models that simulate the environment, stimulate the system or monitor specific values.
- Bind the requirements to the design model by setting the properties of a requirement to values inside the design model using explicit assignments.

Particularly the last step will require the involvement of the system designer in order to ensure that the requirement properties are linked properly, i.e. to the correct properties values inside the design model. For example, the assignment for the requirement property *caution_signal_received* is as follows:

```
caution_signal_received =
design_model.train1.pc1.tcs.controller.tracks_signals_status == 1
```

This means that the requirement property *caution_signal_received* will become true when the controller property *tracks_signals_status* is equal to one[9].

Another example is the assignment of the requirement property alarm_is_activated. Here the system tester will have to decide which design property it should be linked to. It could be accessed from the ATP controller or from the HMI system, that is between the controller and the driver (see Fig. 3), or from the driver HMI port directly. The answer will probably be: It should be accessed from the driver HMI port because failures in HMI system may also affect the evaluation result. Furthermore, it is recommended to create the following attributes and statements[10] for each test model:

- `test_passed := evaluated and not violated;`
  Indicates if the test is passed or failed.
- `evaluated := if req1.evaluated and ... and reqN.evaluated then true ...;`
  Indicates if the test case has evaluated all requirements.
- `violated := when {req1.violated,... ,reqN.violated} then true ...;`
  Indicates if any of requirements was violated.

These definitions enable an automated test case results evaluation by using the requirement violation monitors of the involved requirements as a test oracle for the test case. Figure 7 presents an example of a test case that drives the simulation.

4.5 Method Step: Test and Observe Requirement Violations

After having created the test models, the system tester can run simulations and observe the results. Hereby, the system tester will be interested in knowing if test cases have passed or failed. A test case is deemed to have failed when not all requirements were evaluated or some requirements were violated during the execution of the test case. In our approach a Modelica simulation tool (e.g. MathModelica[11]) allows a visual monitoring of the requirement violation monitors during the simulation as shown in Fig. 8.

[9] "1" denotes a caution signal in the design model
[10] These statements are written in Modelica.
[11] http://www.mathcore.com

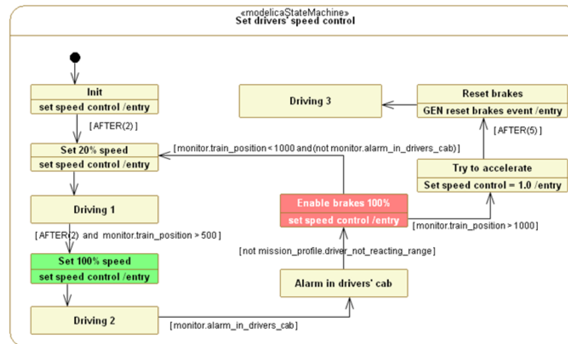

Fig. 7. Test case example

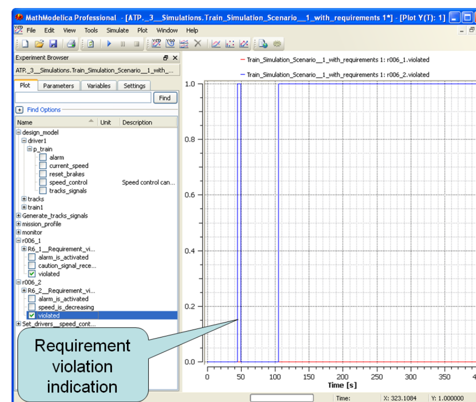

Fig. 8. Test execution and requirement violations observation

4.6 Method Step: Report and Analyze Test Results

After the execution of all test cases, the system tester creates a simulation report that should, for each test model, at least include the following information:

- Which design model, test cases and requirements were included?
- Did the test cases pass?
- If not, were all requirements evaluated?
- If yes, are there requirements violations?

This information is the basis for discussions among the involved parties and may lead to an iterative repetition of the system design and testing process described here. Furthermore, it allows the precise reproduction of test results at a later state. Additionally, these reports can be reused as a reference for later product verification activities, i.e., the physical system testing at a test bench.

5 Current Status and Future Directions

The methodology presented in this paper has been successfully applied in several case studies. However, the case studies included only a small number of requirements. In the future, a real-sized case study is planned, i.e., one that contains more than a hundred requirements to be verified using the vVDR method to determine the applicability and scalability of this approach. The traceability between requirements and design artifacts is a critical issue in the daily engineering work, particularly with regards to change impact analysis. When the system design is evolving or changing then the vVDR approach presented in this paper contributes to an efficient way of verifying the new designs against requirements by reusing the formalized requirements and test cases for quick and easy regression testing. This is enabled by the separation of requirements and test cases on the one hand and the design models on the other hand. However, it is still hard to determine the impact of a requirement change on the system design. In order to support impact analysis, a traceability of requirements to design artifacts is necessary at an appropriate level of granularity. For example, parts of a requirement statement, i.e.,. single words, should be linked to the different design model elements that they are referring to. Moreover, an effective visualization and dependencies exploration is necessary in order to enable an efficient handling of changes. A model-based development approach enables an effective and efficient reporting on and monitoring of the requirements implementation. For example, a bidirectional traceability between requirement and design allows the determination of the system development status and supports project risk management and planning. While the test cases for our running example can be easily derived directly from the input requirements, manual test case generation becomes an increasingly tedious task for real-life specifications with hundreds of requirements. Model-based testing provides methods for automated test case generation some of which already work on UML models[17] and look promising to be adapted to vVDR. Requirements traceability, a higher test automation through adaptation of model-based testing techniques as well as reporting topics are subject to our future work.

6 Conclusion

This paper presents a method for the virtual verification of system designs against system requirements by means of simulation. It provides a detailed description of all method steps and illustrates them using an example case study that was implemented using ModelicaML. It points out that this method strongly depends on the design models that are planned to be created and that not all type of requirements can be evaluated using this method. In the vVDR approach, formalized requirements, system design and test cases are defined in separate models and can be reused and combined into test setups in an efficient manner. In doing so, a continuous evaluation of requirements along the system design evolution can be done starting in the early system design stages. This approach enables an early detection of errors or inconsistencies in system design,

as well as of inconsistent, not feasible or conflicting requirements. Moreover, the created artifacts can be reused for later product verification (i.e., physical testing) activities.

References

1. C. Haskins, Ed., *Systems Engineering Handbook: A guide for system life cycle processes and activities.* INCOSE, 2006.
2. D. Gause and G. Weinberg, *Exploring requirements: quality before design.* Dorset House Pub, 1989.
3. P. Loucopoulos and V. Karakostas, *System requirements engineering.* McGraw-Hill, Inc. New York, NY, USA, 1995.
4. E. Hull, K. Jackson, and J. Dick, *Requirements engineering.* Springer Verlag, 2005.
5. J. Estefan, "Survey of model-based systems engineering (MBSE) methodologies," *Incose MBSE Focus Group*, vol. 25, 2007.
6. P. Helle, A. Mitschke, C. Strobel, W. Schamai, A. Rivière, and L. Vincent, "Improving Systems Specifications - A Method Proposal," in *Proceedings of CSER 2008 Conference, April 4-5 2008, Los Angeles, CA*, 2010.
7. A. Davis, *Software requirements: objects, functions, and states.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1993.
8. V. A. d. Santiago Júnior, *Natural language requirements: automating model-based testing and analysis of defects.* São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2010.
9. L. Mich, M. Franch, and P. Novi Inverardi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 2, pp. 151–151, 2004.
10. J. Woodcock, P. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM Computing Surveys (CSUR)*, vol. 41, no. 4, pp. 1–36, 2009.
11. P. Helle and W. Schamai, "Specification model-based testing in the avionic domain - Current status and future directions," in *Proceedings of the Sixth Workshop on Model-Based Testing 2010, Paphos, Cyprus*, 2010.
12. I. Groher, A. Reder, and A. Egyed, "Incremental Consistency Checking of Dynamic Constraints," *Fundamental Approaches to Software Engineering*, pp. 203–217, 2010.
13. T. Myers, P. Fritzson, and R. Dromey, "Seamlessly Integrating Software & Hardware Modelling for Large-Scale Systems," in *2nd International Workshop on Equation-Based Object-Oriented Languages and Tools, Paphos, Cyprus*, 2008.
14. D. Powell, "Requirements evaluation using behavior trees-findings from industry," in *Australian Software Engineering Conference (ASWEC07)*, 2007.
15. W. Schamai, P. Fritzson, C. Paredis, and A. Pop, "Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behavior Using Graphical Notations," in *Proc. of the 7th International Modelica Conference, Como, Italy*, 2009.
16. P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1.* Wiley-IEEE Press, 2004.
17. M. Prasanna, S. Sivanandam, R. Venkatesan, and R. Sundarrajan, "A survey on automatic test case generation," *Academic Open Internet Journal*, vol. 15, 2005.

A ATP requirements

ID	**Requirement Text** (based on [13])
001	The ATP system shall be located on board the train.
001-2	The ATP system shall consist of a central controller and five boundary subsystems that manage the sensors, speedometer, brakes, alarm and a reset mechanism.
002	The sensors shall be attached to the side of the train and read information from approaching track-side signals, i.e. they detect what the signal is signaling to the train driver.
002-2	Within the driver cabin, the train control display system shall display the last track-side signal values calculated by the controller.
003	Three sensors shall generate values in the range of 0 to 3, where 0, 1 and 2 denote the danger, caution, and proceed track-side signals respectively. Each sensor shall generate the value 3 if a track-side signal that is out of the range 0..2 is detected.
004	The controller shall calculate the majority of the three sensor readings. If no majority exists then the value shall be set to "undefined" (i.e. 3).
005	If the calculated majority is "proceed" (i.e. 0) then the controller shall not take any action with respect to the activation of the braking system.
006-1	If at any time the controller calculates a "caution" signal, it shall, within 0.5 seconds, enable the alarm in the driver cabin.
006-2	If the alarm in the driver cabin has been activated due to a "caution" signal and the train speed is not decreasing by at least $0.5m/s^2$ within two seconds of the activation, then the controller shall within 0.5 seconds activate the automatic braking.
007-1	If at any time the controller calculates a "danger" signal it shall within 0.5 seconds activate the braking system and enable the alarm in the driver cabin.
007-2	If the alarm in the driver cabin has been activated due to a "caution" signal, it shall be deactivated by the controller within 0.5 seconds if a "proceed" signal is calculated and the automatic braking has not been activated yet.
008	If at any time the automatic braking has been activated, the controller shall ignore all further sensor input until the system has been reset.
009	If the controller receives a reset command from the driver, then it shall within 1 second, deactivate the train brakes and disable the alarm within the driver cabin.