Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 264 (2010) 85–99
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Model-Based Testing 2010: Short Abstracts

Jan Tretmans[1]

Embedded Systems Institute
Eindhoven, The Netherlands

Florian Prester[2]

sepp.med gmbh
91341 Roettenbach, Germany

Philipp Helle[3]

EADS Innovation Works
Filton, UK

Wladimir Schamai[4]

EADS Innovation Works
Hamburg, Germany

Abstract

This article contains three short abstracts of presentations given at the Sixth Workshop on Model-Based Testing (MBT 2010) on March 21, 2010 in Paphos, Cyprus: **Theory of Model-Based Testing and How ioco Goes eco** by Jan Tretmans (invited presentation), **Model-Centric Testing** by Florian Prester and **Specification Model Based Testing in the Avionic Domain - Current Status and Future Directions** by Philipp Helle and Wladimir Schamai.

[1] Email: jan.tretmans@esi.nl
[2] Email: florian.prester@seppmed.de
[3] Email: philipp.helle@eads.net
[4] Email: wladimir.schamai@eads.net

doi:10.1016/j.entcs.2010.12.016

1 A Theory of Model-Based Testing *and How ioco Goes eco*

Jan Tretmans [5]
Embedded Systems Institute
Eindhoven, The Netherlands

Model-based testing is one of the promising technologies to meet the challenges imposed on software testing. In model-based testing test cases are generated from a model that describes the required behaviour of the implementation under test. First, we will argue that model-based testing is more than just the generation of some test cases from a model. A well-defined and sound theory of model-based testing is feasible and necessary, and it brings many benefits, also practical ones. As an example we will consider the ioco-testing theory, where models are expressed as labelled transition systems and correctness is defined with the ioco-implementation relation. Second, we consider component-based testing as an application area of model-based testing, triggered by the popularity of component-based development. The strong and weak points of the ioco-testing theory for component-based testing will be discussed, and an ioco-variant called 'eco', environmental conformance, is presented that allows model-based testing of both provided and required interfaces of a component.

1.1 Model-Based Testing

Quality of software is an issue of increasing importance and growing concern. Systematic testing is one of the most important and widely used techniques to check the quality of software. Testing, however, is often a manual and laborious process without effective automation, which makes it error-prone, time consuming, and very costly. One of the new technologies to meet the challenges imposed on software testing is *model-based testing*.

In model-based testing a model of the desired behaviour of the *implementation under test* (IUT) is the starting point for testing. The main virtue of model-based testing is that it allows test automation that goes well beyond the mere automatic execution of manually crafted test cases. It allows for the algorithmic generation of large amounts of test cases, including test oracles, completely automatically from the model of required behaviour.

From an industrial perspective, model-based testing is a promising technique to improve the quality and effectiveness of testing, and to reduce its cost. The current state of practice is that test automation mainly concentrates on the automatic execution of tests for which a multitude of commercial test execution tools is available. These tools, however, do not address the problem of test generation. Model-based testing aims at automatically generating high-quality test suites from models, thus complementing automatic test execution.

[5] This work has been supported by the EU FP7 under grant ICT-214755: Quasimodo.

From an academic perspective, model-based testing is a natural extension of formal methods and verification techniques, where many of the formal techniques can be reused. Formal verification and model-based testing serve complementary goals. Formal verification intends to show that a system has some desired properties by proving that a model of that system satisfies these properties. Thus, any verification is only as good as the validity of the model on which it is based. Model-based testing starts with a (verified) model, and then intends to show that the real, physical implementation of the system behaves in compliance with this model. Due to the inherent limitations of testing, such as the limited number of tests that can be performed, testing can never be complete: testing can only show the presence of errors, not their absence.

1.2 A Theory of Model-Based Testing

An important benefit of model-based testing is the automatic generation of large numbers of test cases from a model. Model-based testing, however, is more than just generating an amount of test cases from a model. To prevent that just any artifact generated from a model could be called a test case, test generation must be based on a sound and well-defined underlying theory of model-based testing. Such a theory must support precise reasoning about the objects of model-based testing, such as models, IUTs, test cases, test generation, and verdicts. Even more important, such a theory must establish relations between these objects: it defines precisely when an IUT is correct with respect to a model, what it means for a test case to be valid, and what a correctness proof for a test generation algorithm encompasses.

Two important ingredients of such a theory of model-based testing are a testing hypothesis and an implementation relation. A *testing hypothesis*, or test assumption [1], establishes the link between the black-box, real IUT, which consists of software, hardware, physical components, or a combination of these, and the world of models. The assumption is made that any real IUT can be modelled by some object in a domain of models. The testing hypothesis presupposes that such a domain of models is known apriori, and that a valid model of the IUT exists in this domain, but not that this particular model is apriori known. In this way, the testing hypothesis allows reasoning about IUTs as if they were models in this (formal) domain.

Building on the testing hypothesis, an *implementation relation*, also called conformance relation or refinement relation, is a formal relation between models of IUTs and specification models [3]. It defines when an IUT is correct with respect to a specification model. This implies that an implementation relation is at the core of a theory of model-based testing, and that validity of test cases and correctness of a test generation algorithm must always be assessed with respect to an implementation relation.

Only with a well-defined and sound theory of model-based testing we can expect to achieve the full benefits of model-based testing:

- The validity of a test case can be precisely defined and established. Also a test-case generation algorithm can be proved to produce valid test cases, i.e., test

cases that detect errors, and only errors.

- Model-checking and model-based testing serve complementary goals. Model-checking aims at showing that a model is valid and has particular properties. Model-based testing starts with a valid model to show that the IUT behaves in compliance with this model. Only when model-based testing and model-checking are based on compatible theories, we can ensure that model-checked properties are preserved in the IUT.
- The boundaries between validation techniques such as model-based testing, model checking, static analysis, abstract interpretation, theorem proving, run-time verification, etc. diminish. Combined and integrated use of these techniques is necessary in order to be able to choose the best combination of techniques for every validation task, but this requires compatibility and consistency of their respective theories.
- Many current model-based testing approaches and tools are restricted to deterministic, complete, and fully-defined models. Defining an implementation relation is then straightforward. Model-based testing, however, must also be able to deal with the imperfect and incomplete real world, in which requirements are never complete and always evolving, e.g., in an agile process, and specifications are always partial or loose, and in addition non-deterministic and concurrent. In such a case, defining an implementation relation is not straightforward, and proving validity of a test-generation algorithm is not trivial.
- By making the testing hypothesis and the implementation relation explicit, you also make explicit what kind of properties are being tested, and, perhaps even more important, what kind of properties are not tested, and, consequently, what kind of errors may remain after testing.
- Having a well-defined theory allows to formally compare model-based testing approaches and test-generation algorithms, for example, in terms of their error detecting capabilities.

1.3 Model-Based Testing for Labelled Transition Systems

One of the theories for model based testing is the **ioco**-testing theory, where models are expressed as labelled transition systems and compliance is defined with the **ioco** implementation relation [7]. This model-based testing theory, on the one hand, provides a sound and well-defined foundation for transition system testing, having its roots in the theoretical area of testing equivalences and refusal testing [4]. On the other hand, it has proved to be a practical basis for several model-based test generation tools and applications [6].

The implementation relation **ioco** is based on the testing hypothesis that implementations behave as input-enabled labelled transition systems. Over time, a number of variants of **ioco** have been defined which differ in, e.g., specification models, input-enabledness, and the treatment of partial specifications. The relation **uioco** is more specific with respect to partial specifications, **wioco** does not assume input-enabledness of implementations, **tioco** and **rtioco** add real-time properties

to models, and **hioco** extends this to hybrid transition systems. The relation **sioco** treats data in a symbolic way, which facilitates transposing pre/post-conditions to the realm of labelled transition systems, and **stioco**$_{\mathcal{D}}$ extends this to the combination of real-time and data. The relation **mioco** does it with multiple input and output channels, **ioco**$_r$ considers refinement of actions, **qioco** adds quantification of imprecision, **poco** does it with partial observability, and **eco** takes the environment into account.

1.4 Model-Based Testing for Components

In component-based development, the correctness of a system depends on the correctness of the individual components and on their composition. In the first place, this requires a compositional implementation relation. But **ioco** is not compositional [2]. In the second place, it requires a behaviour model that describes how the component can be invoked, as well as how the component itself invokes other components. Such a model is usually not available, in particular not the part that models how the component invokes other components (the required interface). Stubs are used instead. We discuss an approach where required interfaces are tested based on what the invoked component, i.e., the environment of the component under test, expects. This leads to environmental conformance **eco** [5], where the model of the provided interface of the invoked component is used for the generation of test cases.

References

[1] Bernot, G., M. G. Gaudel and B. Marre, *Software Testing based on Formal Specifications: A Theory and a Tool*, Software Engineering Journal **1991** (1991), pp. 387–405.

[2] Bijl, M. v. d., A. Rensink and J. Tretmans, *Compositional Testing with* IOCO, in: A. Petrenko and A. Ulrich, editors, *Formal Approaches to Software Testing*, Lecture Notes in Computer Science **2931** (2004), pp. 86–100.
URL http://dx.doi.org/10.1007/b95400

[3] Brinksma, E., *A Theory for the Derivation of Tests*, in: S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII* (1988), pp. 63–74.

[4] De Nicola, R. and M. Hennessy, *Testing Equivalences for Processes*, Theoretical Computer Science **34** (1984), pp. 83–133.

[5] Frantzen, L. and J. Tretmans, *Model-Based Testing of Environmental Conformance of Components*, in: F. de Boer, M. Bosangue, S. Graf and W.-P. de Roever, editors, *Formal Methods for Components and Objects*, Lecture Notes in Computer Science **4709** (2007), pp. 1–25.
URL http://dx.doi.org/10.1007/978-3-540-74792-5_1

[6] Mostowski, W., E. Poll, J. Schmaltz, J. Tretmans and R. Wichers Schreur, *Model-Based Testing of Electronic Passports*, in: M. Alpuente, B. Cook and C. Joubert, editors, *Formal Methods for Industrial Critical Systems*, Lecture Notes in Computer Science **5825** (2009), pp. 207–209.
URL http://dx.doi.org/10.1007/978-3-642-04570-7_19

[7] Tretmans, J., *Model Based Testing with Labelled Transition Systems*, in: R. Hierons, J. Bowen and M. Harman, editors, *Formal Methods and Testing*, Lecture Notes in Computer Science **4949** (2008), pp. 1–38.
URL http://dx.doi.org/10.1007/978-3-540-78917-8_1

2 Model-Centric Testing

Florian Prester
sepp.med Gmbh
91341 Roettenbach, Germany

Model-centric Testing (.mzT[8]) is an efficient method to design and specify (software) tests. Using .mzT it is possible to achieve and even improve the systematic approach and the completeness of the test coverage. .mzT is based on a model based test design but focuses on user workflows, test management information and the tester's mindset. .mzT can be used for every kind of software and system test. It serves all relevant parts of software development, beginning with component view, which is targeting complete test coverage, and ending at system view, which in respect to the number of possible test cases is focusing primarily on systematic reduction of test cases.

2.1 Introduction

Model based testing has been around for years. It is based on the idea to derive test cases from the model of the system under test (SuT) that was created for development purposes. This concept has a drawback: using development information alone only provides verification but not validation. A development model of a software component can be used to generate test cases which can obtain c0 to c2 coverage of the model but do not cover actual user workflows or exceptional and error provoking scenarios. Validation is about answering the question if the correct system was realized. The test cases necessary for this need to include information from the different stakeholders, designers, tester etc. Within this paper the methodology of .mzT is getting explained and sample implementations shown.

2.2 .mzT - model-centric Testing

Model-centric testing (.mzT) [4] is is based on model-based testing (MBT) and therefore a method for the visualization and systematization of test designs and test plans. Whereas with MBT, the implementation is verified based on models from the system definition or the system design, .mzT goes one step further and adds system usage, test management and test aspects to the pure behaviour models (see figure 1). Thus .mzT extends the focus to validation: to test the system similar to the user workflows (usage) and to add exceptional situations/worst cases (tester's mindset) while including the procedures and objectives of the model-based approach. The integration of test management information into the model (such as priorities) and the automatic generation of test cases from the model with the test case generator .getmore [3] also allows test target oriented reduction of the test cases. The advantages of .mzT as opposed to a "traditional" document or script approach are:

[8] .mzT = modellzentriertes Testen (German), English: model-centric testing

- **Systematic** design and generation of test cases, providing controllable completeness in terms of coverage and in the reduction of test cases by test management criteria.
- **Visualization**: both developers and testers make a model of the system at least in their heads. Using .mzT both can work on the same visual model, typically based on UML. This ensures better coordination between stakeholders, requirements managers and developers.
- **Reuse** of information: Avoiding redundancy. The individual test cases no longer have to be created and maintained individually.
- **Modularity** in the models and the possibility of using abstraction levels by means of hierarchies leads to a reduced complexity of the design.
- **Automation** of generating test cases and of the connection to test management and test execution tools provides economic efficiency and avoidance of errors.

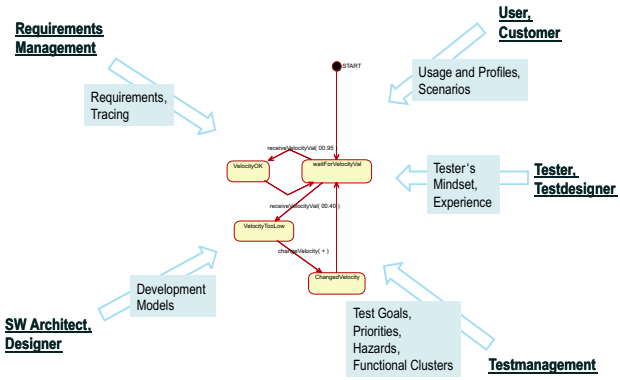

Fig. 1. Model-centric testing (.mzT): In addition to possible input from development models, test modelling is based on user scenarios, test management information and necessary test cases from the viewpoint of the test (exceptional scenarios, error scenarios).

Because .mzT is independent of the tooling, it is possible to systematize the test design without having to modify existing test systems or processes.
The .mzT test design is integrated consistently into the software development process. This results in higher quality and higher process efficiency, and tracing between requirement management, development and testing. In addition, usable and applicable domain standards as well as the experience of experts and users of the system are also included.
The test model is the central repository of the test process. This permits extensive planning and control of the test process from the test model and considerably improved adaptability of the test design to the requirements of the test process. From the requirement to the test report, .mzT and .getmore provide an automated tool chain that places the .mzT model at the center of the test activities.

2.3 Industrial Examples

2.3.1 .mzT in the Automotive World

Automotive OEMs need to integration-test the various control devices, which are supplied by different component manufacturers and the coordination of the functions between different suppliers. Consistent representation of the functionalities of the overall systems and their dependencies poses a great challenge in this field. The practical suitability of the .mzT method and the .getmore test case generator has been demonstrated in several projects in the automotive industry [2].

Testing a turn signal control is presented as an example here. With the turn signal lights, a distinction has to be made between the actual turn signal and the hazard warning signal. The turn signal can be switched on momentarily or continuously activated. The hazard warning signal can be interrupted by a turn signal and has to be re-activated after the end of the turn signal. The activity of the different signals is further detailed in subdiagrams. Each node represents either an instruction to the test system or an instruction to check the current condition of the turn signal light. From this model, the test case generator can generate test cases. If the corresponding library functions are available automated test cases can be generated. Compared with the standard method (manual generation of individual test cases), creation of the diagrams required only very little time, the desired test cases can be generated from the test models created.

2.3.2 Research Project TestNGMed

TestNGMed is a specialized sector solution of an automated test bed for Medical IT, in which .mzT and .getmore are applied [6]. In the domain of medical engineering, the protocol standard HL7 is used as the communication and interaction protocol and IHE as the standardized definition of medical processes. Here the efficient adaptability and expandability of the test designs with the .mzT methodology comes into its own. IHE scenarios form the framework of the test design and, based on the special clinical processes and on the tester's mindset, are expanded to become high-quality validation suites adapted to the specific system. TestNGMed addresses, above all, the topics of conformity with standards and interoperability. The complete test process with TestNGMed is shown in figure 2. The test automation is implemented using a test standard TTCN-3 (Test and Test Control Notation). Currently initial industrial projects are being planned for the deployment of TestNGMed in projective projects with medical engineering manufacturers and in the integration of clinical IT [1].

2.3.3 Prototype TestNGMost

In a cooperation project the extension of the TTCN-3-based test and simulation system TTsuite MOST with the model-centric test design approach was evaluated [5]. MOST (Media Oriented Systems Transport) is the leading network standard for car infotainment. The objective of this project is to systematically test the remote operation possibilities while retaining the greatest possible test coverage. The functions to be tested include play, pause, forward, scan, shuffle, and list, which

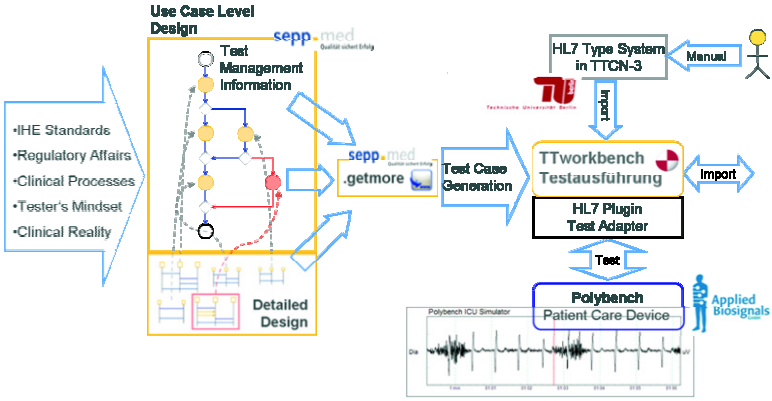

Fig. 2. TestNGMed overall system in the ProInno project

can be executed in any order. Based on the description of all available functions, a test model (.mzT) was created using the model-centric testing methodology. The basis of the test setup is TTsuite MOST with existing TTCN-3 libraries for the MOST connection of the test system, the hierarchical .mzT test model and the infotainment system of a real vehicle. After the test cases have been generated from the test model, they are compiled with TTsuite MOST and are immediately available for execution. Unlike with intuitive, manual test case preparation, with this approach it is possible to systematically identify all test cases and to cut down the number of test cases.

2.4 Future Work

In the near future a focus will lie on increasing tool integration to establish MBT/.mzT test frame works in the market. Also the development of new and innovative strategies that integrate more test management information is important.

Another feature to be developed is "on-the-fly" testing capability. On-the-fly testing means to execute the actions along the paths through the test system model directly. The response of the SuT influences further traversing ot the model and therefore the test execution itself.

Also new projects are at start. With the project "Cost Efficient Test System for Embedded Systems" (CETES) [7] the goal is to optimize and standardize the test process within the embedded community besides established and expensive solutions. CETES will offer a test system using only off the shelf components.

2.5 Conclusion

Model-centric testing shows that MBT can be used in practice successfully. Based on this .mzT grew from an idea to a real methodology including guidelines, training and tooling. .mzT helps the test managers to organize the test not according to a new tool chain, but to use existing tool chains and extend those by MBT/.mzT.

References

[1] K.-H. Kuehnlein, G. Goetz, A. Metzger, M. Seel, *Experiences with Model Centric Testing in Standard Based Medical IT Environments*, Conquest 2009 (Nuremberg)

[2] G. Kiffe, F. Prester, S. Siegl, M. BeisserM, *Model-Driven Test Case Generation in HIL Testing: Usage Models and Model-Centric Testing .mzT*, ATZelektronik 05/2009

[3] http://www.seppmed.de/getmore.65.0.html

[4] http://www.seppmed.de/modellzentrierterTest.187.0.html

[5] http://www.seppmed.de/TestNGMOST.230.0.html

[6] www.testngmed.de

[7] www.cetes.eu

3 Specification Model Based Testing in the Avionic Domain - Current Status and Future Directions

Philipp Helle
EADS Innovation Works
Filton, UK

Wladimir Schamai
EADS Innovation Works
Hamburg, Germany

This paper reviews how the EADS model based systems engineering (MBSE) approach has been adapted to extend to the field of testing to ensure the quality of system specifications and resulting products. The paper provides feedback from the industrial application of model based testing methods in the aeronautic domain. Furthermore, a number of ideas for the future direction of these approaches are provided.

3.1 Introduction

Testing effort grows exponentially with the system size and traditional testing methods seem not to be able to keep pace with the trend in engineering towards more complex and bigger systems. On the construction side a current trend in European industry is to apply model-based systems engineering techniques to deal with the increased complexity of systems and with the intention to produce high quality systems at reduced costs. Unfortunately, the quality assurance side has not kept step with this development.

Intensive research on model-based testing (MBT) and analysis has been conducted in recent years, and the feasibility of the approach has been successfully demonstrated, e.g. in [2,7]. Yet, Boberg [1] shows that most studies apply model-based testing on the component level, or to a limited part of the system while only few studies focus on the application of the technique on the system or even aircraft level. The main difference being that the goal is not to produce code but to provide a high quality specification.

This paper shows how the EADS Innovation Works [9] model based systems engineering [10] (MBSE) approach addressed in [5] has been adapted to extend to the field of testing and provides ideas for future areas of research.

3.2 Need for an integrated MBSE approach

A number of studies have demonstrated that the cost of fixing problems increases as the lifecycle of the system under development progresses, e.g. [3]. 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

[9] EADS Innovation Works is the corporate research organisation of EADS (European Aeronautic Defence and Space Company).

[10] The International Council on Systems Engineering (INCOSE) defines MBSE as follows: "Model-based systems engineering (MBSE) is the formalized application of modeling 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" [8]

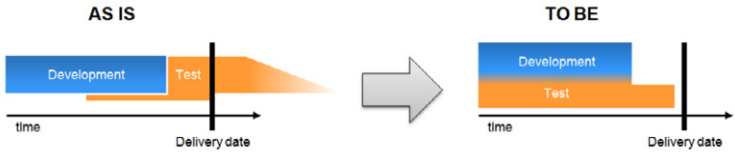

Fig. 3. Envisaged process change

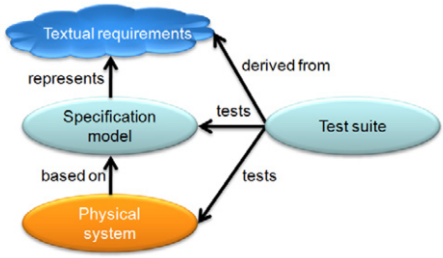

Fig. 4. Engineering artefacts relationship

should be integrated into the lifecycle model so that each phase generates its own tests as figure 3.2 shows.

3.3 Specification model based testing

The term MBT is widely used today with slightly different meanings. Surveys on different MBT approaches are given for example by [6]. For the purpose of this paper, the following definitions of specification model and MBT are used:

Definition 3.1 A specification model is a semi-formal representation of the requirements for a given system.

Definition 3.2 Specification model based testing is testing in which the system specification model as well as the resulting system is automatically tested using test cases that are derived from the system specification model and the system requirements.

The usage of the OMG UML [9] Testing Profile for the test definition in conjunction with SysML [11] for the system specification modelling allows using the same notation from the engineering stage to the testing stage of the development cycle. Figure 4 depicts the relationship of the various engineering artefacts in the frame of specification model based testing.

Figure 5 places the specification model and its different uses in the context of the overall V shaped systems development process [10] and shows how specification model based testing contributes from the system design stage to the other stages of the development cycle.

3.4 Methodology description

Specification model based testing consists of a number of tasks. These tasks build upon each other in a stepwise manner; viz. a lower task can only be performed

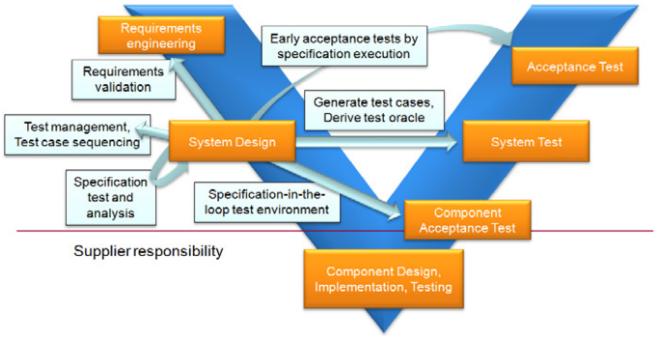

Fig. 5. Specification model based testing in the development process context

when the upper tasks have been completed. The following tasks have been defined:

Identification of system test cases: The analysis of the system requirements in the context of the system specification model allows a derivative identification of test cases. Links provide traceability from the test cases to the originating requirements and allow an assessment of the coverage of requirements by test cases.

Descriptive modelling of system test cases: Descriptive modelling allows a description of the test cases using a combination of natural plain text language and the formal SysML Activity diagram graphical notation. The combination provides a semi-formal notation that is easily used and quickly adapted by system designers and test engineers. Furthermore, the notation is understood easily and the created diagrams can be used for communication purposes with non-engineers as well.

Implementation of executable test cases: Transforming the system test cases from a descriptive form to an executable form is one prerequisite for automatic test execution. Code and test automation tool specific macros are used to translate the natural language description of the individual test steps into concrete code blocks.

Testing of the system model: Automatic test execution tools are capable of executing all the test cases of the test suite, viz. stimulating the simulation of the specification model with the defined inputs and checking if the resulting outputs of the system comply with the expected ones. It can then compile results and coverage reports for individual test cases as well as for the overall test suite. Metrics are automatically computed to show the coverage of the System under Test's states, transitions, operations and event receptions by the test cases.

Specification-in-the-loop testing of real system parts: Specification-in-the-loop testing is testing on a hybrid specification model/real parts test bench setup where a communication between simulated model elements and physical components is possible during the testing process. This allows a verification of system components by testing them against the modelled system specification reusing the test cases that were used in the previous step.

4 Achievements

The presented MBSE methodology was successfully applied and deployed at Airbus for the development of a number of different aircraft systems over the course of three years and included but was not limited to the following applications:

Early design validation: Early design trade-offs by integrating different modelled alternatives in a test bench to conduct usability studies with target users.

Reduced testing effort: Identified and described test cases derived from the specification model will be used as the basis for the physical product verification on a test rig.

Early performance estimation: Integration of stochastic models from Simulink in the specification model to perform multi-system function testing and performance analysis.

Our experience is that specification model based testing integrates well into a project lifecycle that is already employing SysML artefacts and processes and requires little adaptation of the established overall model based systems engineering approach.

5 Way forward

The following points are deemed crucial for future advancements in the field of system verification and validation:

Seamless transfer from research into industrial practice: Boberg [1] states that the "industrial adoption of model-based testing remains low". He continues that this can only partially be attributed to technical concerns and limitations. Much more important are process concerns and most importantly, the model-based testing practice must be integrated into or attachable to current systems engineering software.

Combination of model based analysis and testing: Dijkstra's famous aphorism holds that tests can only show the presence of errors not their absence [4]. Analysis techniques, e.g. model checking, can be used to proof required characteristics of a system. Model-based analysis and testing are complementary quality assurance techniques since static and dynamic analysis provide altogether different types of information. Substantial quality and cost improvement can be obtained when they are systematically applied in combination.

Innovative cross-domain tool framework: Another way forward for model based testing is the integration of the different technologies and tools for analysis and testing into an innovative generic, viz. domain independent, model-based analysis and testing tool solution, a so-called testing framework. Embracing complete separation between tools and its data, and by offering a standard interface for storing and retrieving model elements to and from the platform, tool owners can easily work with their data and integrate with data from other tools in a common, standard way.

Internet based virtual test bench: We envision a development, testing and

analysis framework that allows a verification set up between various physical and logical parts of the design into a virtual test bench via the Internet so that testing can proceed with physical components replacing the virtual model as soon as they become available and the various physical components may be located in widely separate geographical locations.

5.1 Conclusions

We have seen how and why systems development benefits from a model based systems engineering approach that includes model based testing in a number of important ways. In particular, the usage of model based testing techniques throughout the development cycle increases the quality of the system specification and the quality of the system based on the specification at the same time. And all this, without a significant increase in effort compared to more traditional development approaches.

A number of major challenges remain and some ideas of how to overcome in the future have been addressed. There is a strong industrial demand for solutions that perform with a high degree of automation to cope with the ever increasing complexity of the systems that are developed.

References

[1] Boberg, J., *Early fault detection with model-based testing*, In ERLANG '08: Proceedings of the 7th ACM SIGPLAN workshop on ERLANG, 2008, 9-20.

[2] Briand, L., and Y. Labiche, *A UML-Based Approach to System Testing*, In Journal of Software and Systems Modeling (SoSyM) Vol. **1** No. **1** (2002), 10-42.

[3] Davis, A.M., *Software Requirements: Objects, functions and states*, PTR, Prentice Hall, 1993.

[4] Dijkstra, E. W., *Notes of structured programming*, In Structured Programming, O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds. Academic Press, 1972.

[5] Helle, P., and A. Mitschke, C. Strobel, W. Schamai, A. Rivire, L. Vincent, *Improving Systems Specifications A Method Proposal*, In Proceedings of CSER 2008 Conference, April 4-5 2008, Los Angeles, CA.

[6] Utting, M., and A. Pretschner, B. Legeard, *A Taxonomy of Model-Based Testing*, Technical report 04/2006, Dept. of Computer Science, The University of Waikato, 2006.

[7] Zander-Nowicka, J., *Model-based testing of realtime embedded systems in the automotive domain*, Ph.D. thesis, Technical University Berlin, Germany, 2009.

[8] International Council On Systems Engineering (INCOSE), http://www.incose.org

[9] UML, The Unified Modeling Language, information available from http://www.uml.org

[10] Das V-Modell, information available from http://v-modell.iabg.de

[11] SysML, The Systems Modeling Language, information available from http://www.omgsysml.org