Formal Specification and Systematic Model-Driven Testing of Embedded Automotive Systems

Sebastian Siegl, Kai-Steffen Hielscher, Reinhard German
University Erlangen-Nuremberg
Martensstr. 3
91058 Erlangen, Germany
E-Mail: sebastian.siegl|kai-steffen.hielscher|german@ini.fau.de

Christian Berger
Automotive Safety Technologies GmbH
Sachsstrae 16
85080 Gaimersheim, Germany
E-Mail: christian.berger@autosafety.de

***Abstract*—Increasingly intelligent energy-management and safety systems are developed to realize safe and economic automobiles. The realization of these systems is only possible with complex and distributed software. This development poses a challenge for verification and validation. Upcoming standards like ISO 26262 provide requirements for verification and validation during development phases. Advanced test methods are requested for safety critical functions. Formal specification of requirements and appropriate testing strategies in different stages of the development cycle are part of it. In this paper we present our approach to formalize the requirements specification by test models. These models serve as basis for the following testing activities, including the automated derivation of executable test cases from it. Test cases can be derived statistically, randomly on the basis of operational profiles, and deterministically in order to perform different testing strategies. We have applied our approach with a large German OEM in different development stages of active safety and energy management functionalities. The test cases were executed in model-in-the-loop and in hardware-in-the-loop simulation. Errors were identified with our approach both in the requirement specification and in the implementation that were not discovered before.**

Keywords: ***Road Vehicles, Safety Critical Systems, Software Testing, Requirements Engineering, Automated Testing, Verification, Validation***

ACRONYMS & ABBREVIATIONS

(A)SIL	(Automotive) safety integrity level
AUTOSAR	AUTomotive Open System ARchitecture
TDD	Test Driven Development
FMEA	Failure modes and effects analysis
HIL	Hardware-in-the-loop
MCUM	Markov-Chain Usage Model
MIL	Model-in-the-loop
MISRA	Motor Industry Software Reliability Association
SIL	Software-in-the-loop
SUT	System-under-test
TUM	Timed Usage Model

I. Introduction

Software is increasingly contributing to the functionality of embedded systems in modern vehicles. The verification and validation of these increasingly complex systems requires methods that cope with this development. Forthcoming standards like ISO-26262 [1] state requirements on the methods applied in the design, implementation, verification and validation of safety critical embedded systems. The MISRA guidelines [2] substantiate this by prescribing the use of models for safety critical software and hardware. A process that follows the ideas of test driven development (TDD) [3] and model-based testing could satisfy these requirements to a large extent. Although in industry there is often still a lack of formal requirement specification and model-based testing. The reasons are that specifications are often informal and therefore there is no basis for an automated generation of test cases. On account of this often they are, moreover, incomplete and ambiguous. Other reasons are that often there is no adequate tool support or available modeling languages are too general. The contribution of this paper is how formal specification and model-based testing could be applied in industry. The remainder is organized as follows: In section II an overview of related work is given. The established processes and methods in the implementation and integration phases at the OEM are described in section III. Following in section IV our approach and the enhancement of the existing methods are presented. A selection and results of projects that we have conducted following our approach in the development phases of MIL- and HIL testing is described in section V. The requirements for an active safety pre-sensing function and energy-management were formally described by models which served as basis for the following testing activities. The results were compared with those of the traditional method. We present our conclusions in section VI.

II. Related Work

The idea of test-driven development [3] is to specify the test cases for the system consequently before the implementation of the system-under-test (SUT). Benefits of this approach are that it is less likely to forget about test cases because of possible knowledge of the implementation or of a lack of time. Typically the same language is used in TDD for the specification of the test cases as well as for the implementation of the system. This way it should be avoided that developers have to learn an additional programming language. Tools such as *slunit* [4] or *SystemTest* [5] follow this principle. However,

978-3-9810801-7-9/DATE11/©2011 EDAA

this procedure is not feasible in practice w.r.t. the maintanance and enhancement of existing test cases. Especially in the case of changing input- and output-signals and reuse of common aspects in between test cases this principle of TDD is hardly practical in industry. AUTOSAR [6] does not provide neither suitable tools nor a method to verify models in early stages of the development [7]. It provides only guidelines to check the conformance of the models w.r.t. the AUTOSAR standard. Also when it comes to integration testing on HILs, AUTOSAR does not provide tools or methods to verify the systems.

ISO-26262 itself requires methods according to the automotive safety integrity level (ASIL) in which the functionality is classified [1]. It also does not prescribe concrete tools and methods to fulfill the stated requirements.

In our approach we introduce a Timed Usage Model (TUM). TUMs are based on Markov-Chain Usage Models (MCUM), extended by distributions of time. Our approach, which overcomes the drawbacks of TDD, bases on the idea of [8] and is further described in section IV.

III. Existing Processes

In the following the established methods and tools for the verification of implementation models and integration testing are presented.

A. MIL-testing

The tool chain ASTunit with TSdsl, which is used in the early development phase of MIL-testing, consists of two parts. The first is the application TSdsl which allows the processing of instances of the test specification language TSdsl. The second is the tool ASTunit, which is a further development of slunit. TSdsl is a platform independent test specification language. It is the basis to generate platform dependent test environments. The language and the tool were developed with MontiCore [9]. The following aspects were especially considered in the design of the language:

- platform independent representation of test specifications. Hence it is applicable in different phases of the development cycle.
- changing data types for inputs and outputs during the design phase.
- traceability between different versions of test specifications.
- modeling of variability to support software product lines.

An instance of a test specification comprises four parts. The first part specifies the ports of the SUT. The second part can be used to declare variables that are used in the third part, which is the abstract test specification. The forth and last part are concrete test specifications. In figure 2 an example test specification in TSdsl is given. As regards content a seat belt warning system in a standard- and premium variant shall be tested. The system should warn the car passengers if a velocity of 3kmph is exceeded and the driver or codriver has not fastened his seatbelt. In the premium variant this should be signaled in addition accoustically.

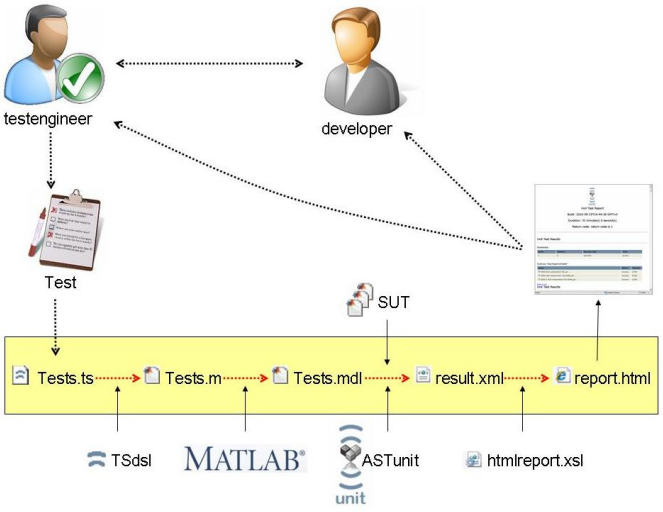

Fig. 1. Test Process ASTUnit

Based on the instances platform dependent code is generated (cp. figure 1). Therefore it is possible in early phases of the development cycle to generate executable scripts in MATLAB from the .ts-files. The generated scripts contain all information to generate the graphical test environment in MATLAB/Simulink for all test cases containted in the test specification. The graphical test environment is executed with the tool ASTunit, which enables the automation of continuous integration systems. With ASTunit the developer can interactively execute the test cases in MATLAB/Simulink and monitor the stimulation of the SUT and evaluate it; alternatively the test specification can be executed completely automatically without interaction. In this case the results are available in an XML-file, which can be viewed in a web-browser. The automation is depicted in the yellow box in figure 1.
The textual test specification (cp. figure 2) can be put under version control. Thus, different versions can be compared and if needed manually changed or extended.

B. HIL-testing

In the following the test automation *Extended Automation Method (EXAM)* is introduced. EXAM is applied within the Volkswagen AG in the development phase of integration testing on HIL-simulators.

a) EXAM Testing Process: – Test automation in the scope of EXAM means the automated generation of platform dependent code and the execution of the derived test suite without human interaction. The EXAM testing methodology [10] defines a process, roles, and tools used to

1) model test cases graphically and platform independently in UML. Sequence diagrams are used for this task and build the formal basis for test case specifications.

```
testsuite "SimpleTestSpecification" {
  system-under-test "MySUT" {
    model = "BeltLockWarning.mdl";
    duration = 0 - 20.0;
    step = 0.01;
    sysin(1, "BeltLockDriver");
    sysin(2, "BeltLockPassenger");
    sysin(3, "EgoSpeed");
    sysout(1, "WarningLEDBeltsUnlocked");
    variant "Premium" {
    sysout(2, "WarningGongBeltsUnlocked");
  }
 }
  constants {
   UNLOCKED = 0;
   LOCKED = 1;
 }
  testcasetemplate "NoWarningTemplate" {
   description = "No warning if both belts are locked.";
   out("BeltLockDriver", LOCKED);
   out("BeltLockPassenger", LOCKED);
   out("EgoSpeed",
   [0;10): 0,
   [10;15): linear(0;5),
   [15;*): 5);
   assert("WarningLEDBeltsUnlocked",
   [0;*): signal == 0);
   variant "Premium" {
   assert("WarningGongBeltsUnlocked",
   [0;*): signal == 0);
  }
 }
 testcase "NoWarn" bases on "NoWarningTemplate" {
  description = "Realization of the template.";
 }
 testcase "DriverNotLocked"
  bases on "NoWarningTemplate" {
  description = "Warning when driver is not locked.";
    tags = "shorttest", "releasetest";
 out("BeltLockDriver", UNLOCKED);
   assert("WarningLEDBeltsUnlocked",
    [0;13]: signal == 0,
    (13;*): signal == 1);
   variant "Premium" {
   assert("WarningGongBeltsUnlocked",
    [0;13]: signal == 0,
    (13;*): signal == 1);
  }
 }
}
```

Fig. 2. Test Specification in TSdsl

2) generate platform dependent test scripts automatically from the formal description in UML.
3) to use sharable test automation functionalities from a structured database.

The majority of test cases is automatically executed on Hardware-in-the-loop (HIL) systems.

C. Drawbacks of existing processes

In both existing processes each single test case must be invented, developed, and specified by an engineer. This procedure has many drawbacks, e.g., the estimation of the test coverage proves to be a very hard task. Moreover, it is unavoidable that one or more important test scenarios remain undiscovered and it is hardly possible to systematically derive test suites that meet the requirements in the automotive domain in an optimal manner.

As test cases have been manually created and selected for automated execution by test designers until now, the decision which test cases to create and execute within time constraints remained in the hands of the engineer. Thus, this decision has not been done systematically on the basis of an unambiguous model. Furthermore, as the requirement specification is not formalized, it is hard to reflect the impact of changes in the requirements to the existing test cases.

To overcome these drawbacks we introduced our model-driven approach which is presented in section IV.

IV. Model based approach

We introduce a Timed Usage Model (TUM) which serves as a formal requirement specification and is the basis for all following test activities. The model provides the possibility to describe timing and data dependencies of the system to be tested [11]. The test planning and test case generation is supported by the models. The appliance of models allowed the systematic generation of test cases and the assessment of the significance of the conducted test activities with respect to the coverage of requirements, from which they are derived.

TUMs are based on Markov Chain Usage Models (MCUM) as they are used in the field of software testing [12], [13]. Concepts from the performance analysis of network systems [14] can be applied to TUMs to derive indicators for the test planning. Hence, dealing with the increasing time criticality of functionality is directly supported by the model from which the test cases are derived.

A. Definition

A TUM consists of:

- A set of *states* $S = \{s_1, \ldots, s_n\}$, that represent possible usage states.
- A set of *arcs* A, representing state transitions. An arc from state s_i to state s_j is denoted by a_{ij}, multiple arcs between s_i and s_j are not allowed.
- A set of *stimuli* Y on the SUT. A stimulus y_j is assigned to each arc.
- The *transition probability* from state i to state j, denoted by p_{ij} for an existing arc a_{ij}. Otherwise the transition probability is $p_{ij} = 0$. The transition probabilities obey the conditions $0 \leq p_{ij} \leq 1$ and

$$\sum_{j=1}^{n} p_{ij} = 1 \quad \forall i = 1, \ldots, n \tag{1}$$

  states that the probabilities of all outgoing arcs from a certain state s_i must sum up to one.
- A probability density function (pdf) t_i to reflect the *sojourn time* is assigned to each state s_i.
- A pdf of the stimulus time t_{ij} is assigned to each arc a_{ij}. This pdf describes the duration of the execution of a stimulus on the SUT. The concept provides the possibility to characterize the stimuli by its typical variation in time, that can be fix or variable and vary from very small to large values.

In Figure 3 an example of a TUM as a directed graph is presented.

Two states have special characteristics, that are:

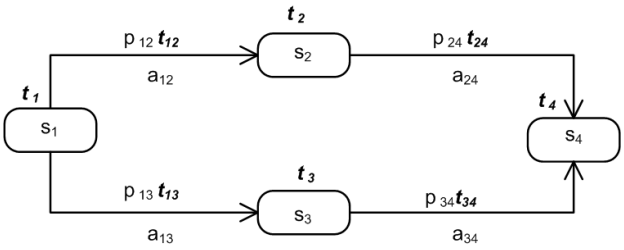

Fig. 3. Timed Usage Model

- State s_1 is the sole initial state (also: start state).
- State s_n is the final state (also: end state).

All paths from the start to the final state are valid test cases. The transition probabilities p_{ij} from state s_i to state s_j as well as the values of the timing attributes t_i and t_{ij} can be stored and exchanged by means of a matrix P. This way different user types can be distinguished w.r.t. the appliance of stimuli and the timing of and between stimuli.

The statistical sampling of test cases can be guided by different usage profiles, that represent different operational conditions of the SUT.

B. Creation of the model

The effect of invocation of functionalities depends much on timing dependencies and there is also a dependency of the functionality to the previously applied data. So to summarize purely functional testing is not sufficient. It is necessary to consider:

- Timing dependencies
- Data dependencies
  - Of the same datum
  - Between different data

These dependencies may exist in any combination and must therefore be considered in testing. As it is not reasonable and possible to test all possible values and dependencies they must be modeled in a manner that provides the basis to derive significant test cases. We achieved this by partitioning each possible input into functional equivalence classes.
A functional equivalence class is specified unambiguously by the stimulus itself, by the range of the input values, and by the timing characteristics of the stimulus. Using Timed Usage Models it is possible to describe timing dependencies such as *not earlier than* and *not later than* [15]. This way it is possible to classify the possible usage into functional equivalence classes, including non-functional aspects like data characteristics and timing. The TUM makes it possible to describe timing dependencies not only by constant values. Additionaly the range and shape of the occurence distribution can be specified.

C. Test Case Generation

Test cases can be sampled from the model via random walk using the probabilities. It is also possible to take the graph abstraction of the model and to apply e.g. deterministic algorithms for the sampling of test cases. In the following the random walk algorithm is presented, that takes the model as stochastic source for the sampling of test cases. We obtain the values p_{ij}, t_i, t_{ij} from the usage profile P that is assigned to the model. An overview is given in figure 4.

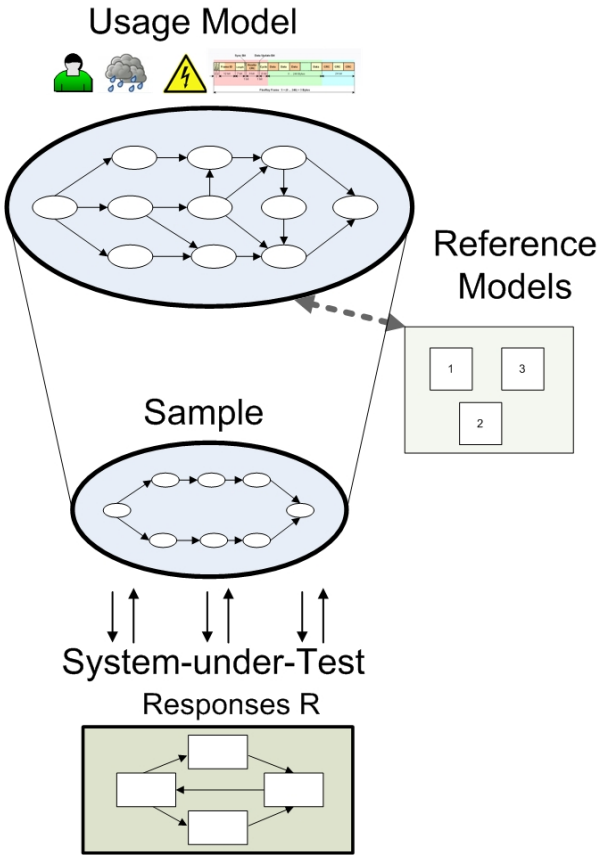

Fig. 4. Test Case Generation from Test Model in Combination with Reference Models

V. Projects

The model-driven approach was introduced in the stages of MIL and HIL testing to extend the existing processes. In figure 5 the extension is presented using the example of EXAM. The first step is to derive the test model from the requirements. The requirements are stored in a requirement management system in natural language. Guidelines with design patterns were elaborated how to formalize the requirements into the TUM [16]. The resulting test model serves as basis to derive test suites with different strategies according to different test aims.

A. MIL - Active Safety System

The test specification language TSdsl is the interface between the test models and the SUT in MATLAB/Simulink. M-scripts were called from the test model during test case generation that wrote the output signal vectors and the reference vectors.

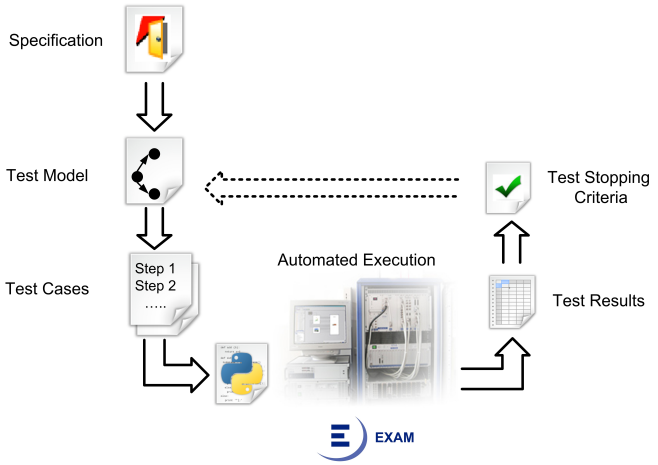

Fig. 5. Extended Process with Creation of Test Model from Requirements and Test Stopping Criteria

These vectors were automatically transformed into a test suite specification in TSdsl.

We applied the presented method in the predevelopment of an active safety system. The system conditions the passengers in critical situations w.r.t. the longidutinal dynamics and vertical dynamics in the seat by applying shortening of the seat belt in dependence of the occupancy. In doing so the effect of passive safety systems like window-bags should be optimized.

During the creation of the test model the requirements, which are stored in DOORS, were analysed. The analysis is done automatically during the creation of the model, in which the requirements are put into a unique and formal description. Two independent behaviour dimensions were identified for the system and modeled in the state space of the TUM. Between the states of the model the stimuli are defined, e.g. increasing or decreasing acceleration and changing of the braking-pressure. In addition to the stimuli sequences the set of expected responses was described with the model. For each possible path through the model and hence each possible stimulus sequence taking into consideration the timing dependencies the expected behaviour of the SUT is defined.

In order to be able to obtain an indicator about the quality of the SUT an executable test environment in MATLAB/Simulink was automatically generated from the model. We used the strategies provided by All4tec Testor for this task. The generated TSdsl testsuite specification can be used by the developer, as described in section III, for interactive verification of the SUT w.r.t. its transitions between states and dataprocessing.

The presented method allowed the systematic analysis of the requirement specification of the system. Inconsistencies of parametrizable features of the system could be identified and clarified during the creation of the model. Furthermore, with the appliance of the presented approach, four usage scenarios could be identified that were not tested in the existing test suite which comprised 127 test cases. The new test cases revealed a faulty transition in the implementation model of the SUT. This faulty transition could then be easily identified and fixed. Because of the extended method the requirements could be clarified, the test coverage could be increased and a previously undiscovered error in the implementation could be identified.

B. HIL - Energy Management Function

The energy management monitors the energy consumption of components and health status of energy supplies. It has to assure that enough power is available to maintain the operation of all safety critical functionalities under all conditions. When the available energy gets critical it requests the start of generators such as the engine. For this decision the remaining energy is predicted.

The requirements specification of the energy management was described in natural language text, drawings, and tables that were structured in a requirement management system.

Furthermore test cases that had been already specified manually before the creation of the usage model were available. So there was a basis to evaluate the manually created test cases w.r.t. the automatically generated test cases.

1) Creation of the model: During the creation of the model the requirements were analyzed and brought into an unambiguous, unique representation. The creation of the model helped to clarify the requirements. Deficiencies in the requirements could be identified such as e.g. usage scenarios in which it was not clearly specified how the system should behave. Function responsibles were involved to clarify these issues. In the end, the system was described by the usage model in a complete and traceably correct manner.

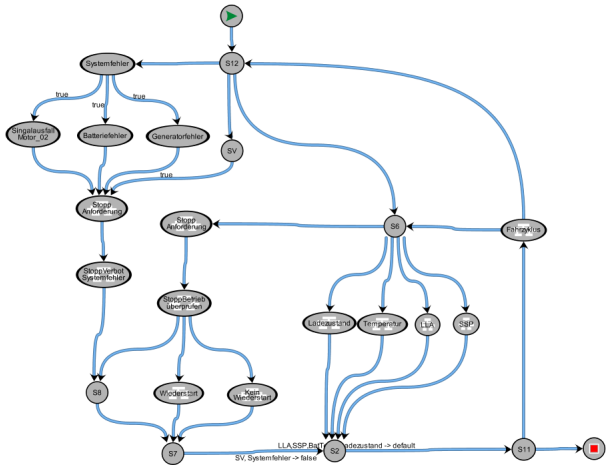

Fig. 6. Detail of Test Model for Energy Management

The final model consisted of 403 states and 520 transitions. The expected average length of a test case was 75 transitions. As the timing information can be stored in the model the testing time can be estimated for each test case.

In figure 6 a top level detail of the model is presented. Scenarios that affect different aspects of the system are grouped by macrostates. By means of the hierarchisation the model is kept well-arranged. Usage scenarios that are of interest at different usage states can be referenced several times.

The set of existing test cases for the functionality was imported and taken into the requirements library of the model editor tool. This way it was possible to associate the existing test cases with the corresponding paths in the model. Later a test suite was generated to cover all existing test cases.

2) Generation of test cases: Test cases were automatically generated from the model.

In addition, previously manually defined test cases were imported into the model. The existing test cases were linked to the corresponding paths in the model. This way the coverage of the existing test cases could be assessed w.r.t. the formal requirement specification, provided by the model.

The existing test suite, which comprises 23 test cases, was analyzed w.r.t. the coverage of possible usage scenarios. The figures are presented in table I. Moreover, the existing set of test cases covered only 46.64% of all possible state transitions. The existing set of test cases is not sufficient to cover all requirements, because possible paths derived from the requirements are not taken.

We automatically derived a testsuite of 23 test cases in order to compare them with the 23 manually created test cases. The results are presented in Table I. The automated generated test cases achieve a higher coverage w.r.t all model elements. The coverage of transitions, to which the stimuli are assigned, is with 57.53% significantly higher than the coverage of 46.64% that is achieved by the manually created test cases. This result shows that on the basis of model driven test case generation the test case generation is optimized in comparison to the manual creation of test cases.

Coverage of 23 Test Cases	Manual	Automated
States	52.70%	63.72%
Transitions	46.64%	57.53%
Inputs	23.96%	33.33%

TABLE I
COVERAGE OF MANUALLY AND AUTOMATICALLY GENERATED TEST CASES

This is an important aspect, as testing time is scarce in industry and should be used as efficiently as possible. Flaws could be already identified during the creation of the model. These flaws were discovered not until a single test case had been executed.

VI. CONCLUSION

The integration of increasing complex and distributed embedded systems in road vehicles poses new challenges to verification and validation. Forthcoming standards such as ISO-26262 explicitly state requirements to the methods applied during the development of safety critical functions. We applied Timed Usage Models to enhance the development methods in the development and integration phase at a German OEM. An active safety system was verified by our approach. Flaws in the requirements could be identified. Furthermore, a faulty transition in the implementation could be identified and fixed that was not discovered by the test suite that was derived by the established method.

We applied our approach also in the stage of integration testing. The energy management functionality for the decision making of the Start-Stop system was described formally by the model. The model served as basis for the validation of the embedded electronics of the energy management and to control the engine. During the creation of the model the requirements were clarified. The test suites to be carried out for the validation of the embedded electronics were improved by automatically generated test suites. The significance of the test cases could be assessed. This is hardly possible without a unique description in form of a model.

Our next steps comprise the evaluation of the appliance of the presented method through the development stages of the whole V-model. Our aim is to be able to reuse the models that we have created in the development phase of MIL-testing for the integration tests on the HIL simulator. Furthermore we examine how the presented approach complies with the requirements of ISO-26262.

REFERENCES

[1] International Organization for Standardization., *ISO/DIS 26262 (2009). Road vehicles Functional safety. Draft international standard.* http://www.iso.org, 2009.

[2] MISRA Ltd, *Guidelines for the safety analysis of vehicle-based programmable systems.*, http://www.misra.org.uk, 2007.

[3] K. Beck, *Test-Driven Development*. Amsterdam: Addison-Wesley Longman, 2003.

[4] T. Dohmke, *slunit, http://slunit.domke.de*, 2010.

[5] The Mathworks, *Systemtest, http://www.mathworks.com/products/systemtest*, 2010.

[6] AUTOSAR Munich, *AUTOSAR 4.0, http://www.autosar.org/*, 2010.

[7] T. C. Müller, M. Lochau, S. Detering, F. Saust, H. Garbers, L. Märtin, T. Form, and U. Goltz, "Umsetzung eines modellbasierten durchgängigen Entwicklungsprozesses für AUTOSAR-Systeme mit integrierter Qualitätssicherung," Technische Universität Braunschweig, Tech. Rep. 2009-06, 2009. [Online]. Available: http://www.digibib.tu-bs.de/?docid=00031645

[8] J. M. Carter and J. H. Poore, "Sequence-based specification of feedback control systems in Simulink®," in *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*. New York, NY, USA: ACM, 2007, pp. 332–345.

[9] H. Grönniger, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel, "Monticore: a framework for the development of textual domain specific languages," in *ICSE Companion '08: Companion of the 30th international conference on Software engineering*. New York, NY, USA: ACM, 2008, pp. 925–926.

[10] G. Kiffe, *EXtended Automation Method (EXAM) Konzeptpapier Version 3.0*, Audi AG, Ingolstadt, Mai 2009.

[11] S. Siegl, K.-S. Hielscher, and R. German, "Introduction of Time Dependencies in Usage Model Based Testing of Complex Systems," in *Systems Conference, 2010 4th Annual IEEE*, apr. 2010, pp. 622 –627.

[12] G. H. Walton, J. H. Poore, and C. J. Trammell, "Statistical testing of software based on a usage model," *Software - Practice and Experience*, vol. 25, pp. 97–108, 1999.

[13] S. J. Prowell, "Using markov chain usage models to test complex systems," in *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, 2005, p. 318.3.

[14] R. German, *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[15] S. Siegl, K.-S. Hielscher, and R. German, "Introduction of Time Dependencies in Usage Model Based Testing of Complex Systems," in *IEEE Systems Conference 2010*, Los Alamitos, CA, USA, 2010.

[16] S. Siegl, *Handbuch Automatische Testfallgenerierung aus Benutzungsmodellen*, INI.FAU, 85057 Ingolstadt, Germany, 2010.