Partitioning the Requirements of Embedded Systems by Input/Output Dependency Analysis for Compositional Creation of Parallel Test Models

Sebastian Siegl
Audi Electronics Venture GmbH
Ingolstadt, Germany
sebastian.siegl@audi.de

Martin Russer, Kai-Steffen Hielscher
Department of Computer Sciences
University Erlangen-Nuremberg, Germany
martin.russer|kai-steffen.hielscher@fau.de

***Abstract*—In industry, embedded systems continue to become even more complex. The development is done increasingly in a virtual environment to accelerate the prototyping and the implementation. Independent from this fact, testing is still the essential activity for verification and validation and obtaining confidence in the quality and reliability of the product. The main driver behind innovation is software, and due to its relevance in September 2013 the first international standard for software testing, i.e. the ISO/IEC/IEEE 29119, was published. It states the main purpose of requirements based testing as to determine whether the test item meets end-user requirements [1, Part 1, p.31]. In the same section it is mentioned, that testing may suffer if the requirements are incomplete or not consistently specified. This highlights, that with the importance of testing, the crucial issue is the quality of the testing activities. Research and industry have researched, how to derive test cases e.g. to cover requirements within minimal time or to achieve coverage criteria like pairwise testing. For the automation of the test execution, the assessment and reporting a broad tool support is available. Yet, one question is increasingly difficult: What are good, i.e. significant test cases. This leads to the question, from what should test cases be derived, to represent the most significant scenarios of use. Constructive requirements modeling methods guarantee that the requirements are analyzed and the final test model is complete and unambiguous. To achieve this output, however, all observable states of use and their combinations are considered during creation of the model. This is apparently cumbersome and hardly feasible in the industrial routine.**

In this paper we present a novel approach to reduce the effort for creating the model and facilitate its appliance in industry. We stick to the foundations of constructive enumeration to create a complete, traceably correct, and consistent model, but we do first decompose the task into manageable units by input/output dependency analysis. The expected behavior is formalized in temporal logic. The resulting model is a composition of all models, that run in parallel. As time is explicitly considered during the creation of the model, timing information is available for structured testing of non functional, e.g. real time requirements, as well as for the determination of measures and dependability estimators. By this approach, the subsequent activities for quality assurance, such as validation and verification, measurement of coverage criteria, and dependability estimators, e.g. of reliability, safety, and risk, profit from this approach, as they rely on a provably correct basis. We applied the method to an embedded system of a German automotive OEM, that was designed in Matlab Simulink and architectured with AUTOSAR 3.2 methodology. An existing test suite was at hand, that was created with the established method. This existing test suite served as benchmark to assess the quality of the new test suite, derived from the model. We compared the reachability of the test cases inside the implementation with code coverage measures and examined the variance of use imposed by the test suites. We present the promising results in this paper.

I. Introduction

Testing is an essential activity for verification and validation in the development of embedded systems. By observing the execution of the system under test, one judges, whether the system behaves as expected. Misbehavior and malfunctions can be identified. As testing provides realistic feedback of the behavior, it is a key activity in industry before releasing a product on the market. Nowadays, a broad tool support exists for the automation of test case execution and reporting [2], [3]. Before testing, however, many activities are required. Indeed, these activities have impact on the dependability and significance of the testing activities. A direct effect is related to the quality of the requirements, that specify the intended scenarios for use and the expected behavior of the system. A major risk is the mismatch between implemented functionalities and customer's needs.

Requirements analysis is the first main activity after the decision for the development of a system is made. It also constitutes the first activity in which errors can be made. It is even the most critical activity with regard to faults and defects, because defects discovered in late development phases might have their origin in the initial requirements. In this paper we present an approach, to base the testing activities on a validated, consistent and complete requirements model.

The paper is structured as follows: In the next section II related work is described and the issue of this paper is extrapolated. In section III terminology definitions and foundations of the method are given. In the subsequent section IV our new approach is presented, followed by the application on an automotive embedded system in section V. In section VI we offer the conclusions and an outlook for future work.

II. Related Work

Model-based Testing (MBT) techniques make use of formal descriptions (models) of either the system under test (SUT) or the expected usage by users of the SUT. In the latter case usage models are deduced from the requirements. The usage model serves as basis for the subsequent verification

978-1-4799-5927-3/15/$31.00 ©2015 IEEE

and validation activities. Elaborated techniques are understood to determine dependability measures of a system with usage models. Examples are with a Bayesian model [4] and on basis of Markovian Models [5] to estimate the reliability, safety, and risk of a system.

The credibility of the dependability measures is up to the quality of the usage model and the derived testing activities. Sequence-based specification is a constructive requirements engineering method to create a usage model. Sequence-based specification originates from the functional treatment of software by Mills [6]. It is based upon a component oriented view of software, which is also a main principle of the AUTOSAR methodology [7]. In AUTOSAR, embedded functionality is partitioned into application software components. Nowadays, AUTOSAR is the prevailing methodology used in the automotive domain for the development of embedded software systems [8].

III. Constructive Requirements Modeling

In this section we introduce foundations of our approach. We rely on a Time Usage Model (TUM) [9], that is modeled in state diagrams in the state chart notation [10]. TUMs are established test models in the automotive domain [9]. State charts have emerged to one of the most widely used notations in automotive software development [11].

A. Time Usage Model: Formal Representation

The definition of a Mealy machine is restated briefly: A Mealy machine is a 5-tuple $(S, S_0, \Sigma, \Lambda, \Theta)$, consisting of [12]:

- a finite set of states (S)
- a start state (also called initial state) $(S_0 \in S)$
- a finite set called the input alphabet (Σ)
- a finite set called the output alphabet (Λ)
- a function Θ which coalesces δ and λ

The total functions δ and λ are:

- a transition function $\delta : S \times \Sigma \to S$
- an output function $\lambda : S \times \Sigma \to \Lambda$

The initial definition of a mealy machine is extended with annotations for the transitions and states pdfs (probability density function) of the timing and the usage probabilities of transitions to so called TUMs M^{TUM}. The definition for M^{TUM} is given in Equation 2 with an example shown in Figure 1 [13]. Timing an attribute for the *Duration* is used as a representative of the pdf for the duration.

$$M^{TUM} = (S, S_0, \Sigma, \Lambda, \Theta). \quad (1)$$

$$with$$

$$\Theta : S \times \Sigma \times \underbrace{\mathbb{R}_0^+}_{\text{Duration}} \times \underbrace{[0;1]}_{\text{Probability}} \to \Lambda \times S. \quad (2)$$

In time sensitive systems the response to the same input data can be dependent on timing, e.g. durations or time

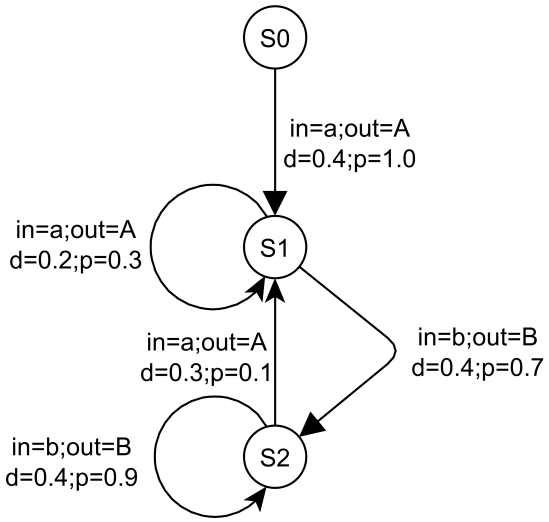

Fig. 1: Exemplary instance of an extended mealy machine M^{TUM}.

intervals between the inputs. The same input may result in another system reaction when applied shorter or longer. Not only the execution duration of a stimulus, but also the time between the execution of stimuli can result in another system reaction. Hence also the time between the execution of stimuli is a decisive factor and should hence be defined in the requirements and reflected in the specification. Waiting time must be considered as it may influence the reaction of the SUT. The duration of a stimulus y is explicitly considered in the extended method used for our work.

Continuous behavior is handled with discrete sampling points of continuous functions. The continuous behavior between the sampling points is handled with continuous functions [14], [15]. The evaluation and logical assessment is treated with expressions in temporal logic [16], [17].

B. Modeling Elements

The user view on the Mealy automaton is a TUM. The TUM-representation is the output of the constructive requirements modeling method, that is presented in the following section III-C, and the basis for all subsequent activities on the basis of the model. 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}.
- 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}. The transition probabilities obey the conditions $0 \leq p_{ij} \leq 1$ and

$$\sum_{j=1}^{n} p_{ij} = 1 \quad \forall i = 1, \ldots, n \quad (3)$$

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.

Test scenarios can be sampled by traversing the model in a semi-Markov Process. Computations such as the probability of occurrence of a state can be computed in advance before test generation. After test cases have been executed, dependability estimators such as the reliability, risk, and safety of the system can be determined [5], [18]. In our approach we focus on the graph abstraction of the model and to apply deterministic and heuristic algorithms for the derivation of test cases. Transitions and states of the model are tagged with information to control the test case generation.

C. Foundations of Constructive Requirements Modeling

The method to create the TUM is the key activity, since the following activities for validation and determination of estimators about the reliability depend on the quality of the model. The method to create the TUM as test model follows the principles of sequence-based specification (SBS) [19].

Classical sequence-based specification consists of two steps:

1) Identify the system boundary
2) Enumerate all sequences of stimuli and their responses

Based on the requirements all stimuli that cross the system boundary are extracted. This comprises stimuli S that represent operations on the SUT and responses R from the SUT. A stimulus y_i is an event that can comprise an information flow, i.e. one or more distinct inputs. In the extended method used for our work also the timing is explicitly considered.

A response R can be composed of one or more outputs. An output is an externally observable information about the SUT. There are two special responses, null, denoted as 0, and illegal, which is denoted as ω. The null response is assigned if no response is observable after the extension with a stimulus, but the sequence is legal. The illegal response is assigned and mapped to the sequence if the sequence is identified as illegal. Next all theoretically possible sequences $S*$ of stimuli to the system are enumerated and mapped to a response, stated as $S* \rightarrow R$. The following step is called *Canonical sequence analysis*: For each new sequence it is checked, whether it is *equivalent* to existing sequences. Two sequences u and v are called *equivalent* if extended by any sequence w the future response is the same to uw and vw. In this case u is said to be reduced to v and not extended further. The sequence v is called a *canonical* sequence. It is a sequence, that spans the discriminative state space of use in real operation.

By this procedure, a complete and consistent usage model is created. An additional feature is the easy traceability of requirements. Beside the annotation of the response to each stimuli sequence u the corresponding requirement is assigned for each step. This ensures the correctness of the enumeration and in doing so incomplete and inconsistent requirements are discovered.

IV. Leveraging the Usage Model

. The feasibility of applying sequence-based specification is limited to functions with a manageable number of inputs. Systems with a larger number of input signals are hard to handle, because a large numbers of inputs can cause an immense state growth during enumeration. 20 signals can produce more than one million (2^{20}) state combinations, provided that all signals are boolean. The state-space issue even increases if the signals are not boolean. Therefore, we developed a method to leverage the appliance of SBS in practice. It concentrates on dependencies between input and output signals and is influenced by our experiences with sequence-based specification in industry.

A. Partitioning the Input Domain

The proposed way to reduce the overall complexity of the enumeration process is to identify independent partitions of input stimuli. The first step is to derive from the requirements in a black-box perspective, which dependencies from input to output signals are specified. The second step is to group the inputs into partitions, in which no cross dependencies between inputs and outputs have been identified. In other words: Input stimuli groups are classified as independent, if according to the requirements no input signal from one partition does have an expected influence on an output of another partition.

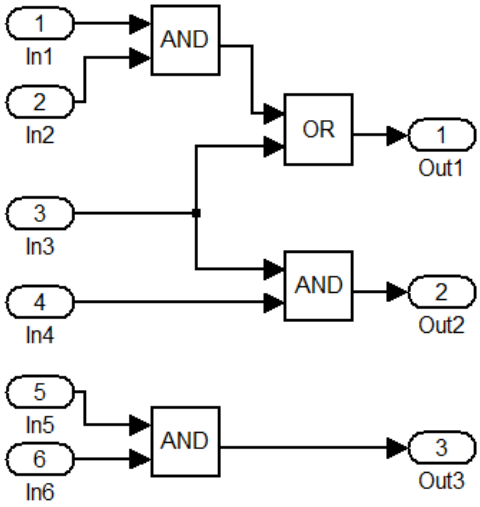

Fig. 2: Example of input/output dependencies

In figure 2 an example is presented, in which the inputs can be grouped into two partitions. On partition has two inputs that affect one output, the other partition consists of four inputs and two outputs. This partition can not be subdivided further, because In3 affects both output signals. This analysis of input/output dependencies is done on the basis of the requirements. It can, obviously, be done on a high level of abstraction, since e.g. algebraic and non functional requirements

are subordinate to the functional dependencies. We propose to split up the enumeration process for the two mentioned groups of inputs. The stimulus sequences derived from the two enumerations can be run in parallel as the inputs are expected to not affect each other. Moreover, the creation of the model is easier, if smaller quantities of inputs must be considered for one model. Safety critical functions may require to test every possible input combination. This leads to two different test suites. Testsuite A runs all the valid sequences developed by enumerating In1 to In4, while the 'independent' stimuli can be stimulated in parallel. The same principle can be applied to Testsuite B, where the enumerated sequences of In5 and In6 can be evaluated together with the combinations of In1 to In4.

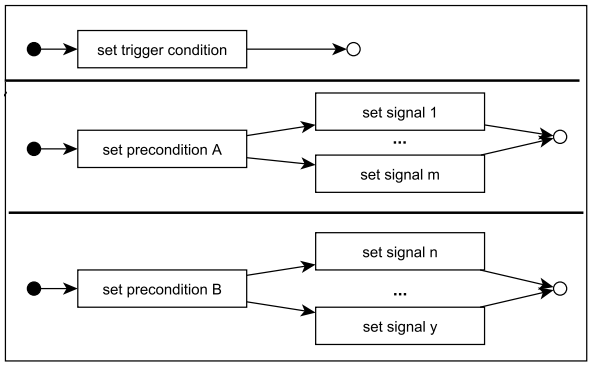

Fig. 3: Resulting test model structure with parallel regions

After decomposing the function, applying the sequence-based enumeration to the partitions and composing them in parallel regions in the test model, the smaller test suites have to be assembled into a single test suite. Figure 3 depicts an example test suite. The test graph is divided into three parallel parts which is shown by two thin lines between the sections. The first one shows the trigger condition, the second (A) and third section (B) result from analyzing and identifying independent input values. Accordingly, input stimuli $1..m$ do not affect any output which depends on the third group of inputs, $n..y$.

V. Application on Embedded Body Functionality

In this section we present the application of the approach on an embedded system and illustrate the methods and processes. The system, which was architectured following the AUTOSAR standard [7] and further designed and implemented with Matlab Simulink [20], controls timing of various modules in the interior of the car. Depending on the interaction of a human with the car like opening or closing doors as well as the cars environment, the system decides whether to trigger modules and checks the activation, the duration of an activation or the deactivation of a specific timer. The selected timing-module of the system has a total input of 14 signals, processes all information in a single step and provides the results by six output signals. Most input signals are described by boolean values and define a certain state of the car, e.g. a signal which gives information whether a door is open(ed) or not. The six output signals give information about the three available timers, two signals for each timer. One describes the running state, a second is true whenever a timer elapses. None of the output signals are returned and used as an input of the module.

The timed behavior is specified by 17 requirements. The initial requirements seemed to be clear and straight forward. However, subsection V-C will show that multiple adjustments had to be made to gain a complete and unambiguous specification, and hence a thorough basis for verification and validation activities.

A. Tool Framework

Based on the steps and principles described in the previous sections, the modeling framework is comprised of two tools to perform the two major steps: sequence enumeration and test execution. The sequence-based specification is associated with requirements engineering. As our primary goal was to improve the activities for validation and verification, we implemented it for test case derivation and automated test case execution and assessment.

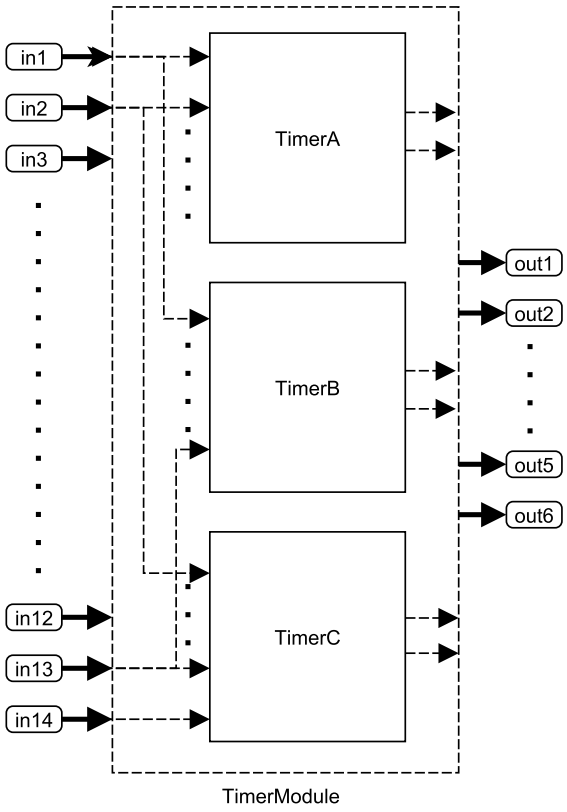

Fig. 4: Architecture of embedded system

1) Enumeration Tool: It is highly recommend to perform the enumeration with tool support, especially, when changes become necessary in previous sequences during a enumeration. Tools can be found in the Internet such as Protoseq [21], written in Ruby, or REAL [22]. REAL is an eclipse-based implementation of the sequence-based enumeration with similar features of Protoseq which works faster than the ruby implementation. Thus, it is more suitable for larger enumerations. Using other software like Excel seems to be justified for very simple examples, but proved to be very error prone when used for more complex functions.

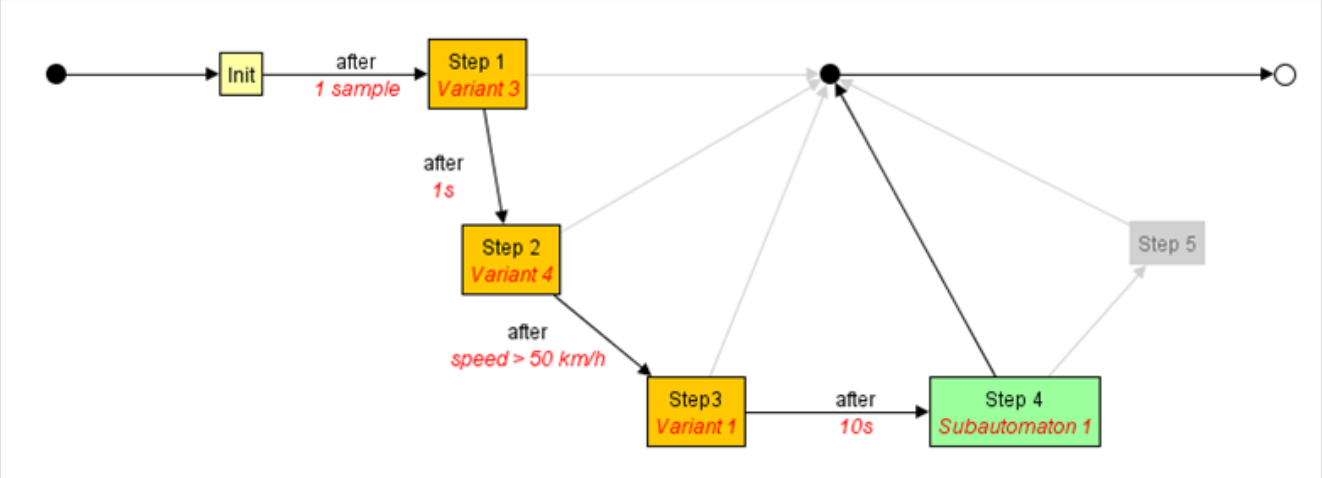

Fig. 5: Detail of a test model in TPT tool

2) Test Environment: For modeling, test derivation, test execution, and test assessment we used the application TPT (Time Partition Testing) [23]. TPT is a model-based testing tool that provides the required features for the method described in this paper, as graphical modeling of hybrid, hierarchical, and parallel state charts.

B. Model of the Embedded System

After defining the test environment, the proposed methods were put into practice. We offer some information about the enumeration below as well as some details and figures about the decomposition in table II.

Timer	Total Number of Inputs	Total Number of Outputs	Decomposed Number of Inputs	Affected Number of Outputs
TimerA	14	6	5	2
TimerB			7	2
TimerC			8	2

TABLE I: Decomposition details

On the top layer, the test model consists of 52 transitions and 26 states and covers all 615 legal sequences gained from the three enumerations. This provides a nice view on the system compared to the overall complexity of the system and model. Furthermore, the visual way of building up the model with preconditions plus stimulus provided a good basis for understanding and discussion between the stakeholders of the project.

C. Results and Evaluation

Prior to applying the decomposed enumeration method, the timer module was already tested on basis of the requirements. The existing test suite achieved a code coverage of 100%. In this section we show the results of the new test suite and compare it with the existing one.

During the first steps of the sequence-based enumeration, the method quickly discovered the incompleteness of the modules specification. It was notable, that many requirements define a expected result under specific conditions, but make no statement of the expected behavior if the condition is not met. Consequently, there were sequences for which no requirement specified the output of the function. It was therefore necessary to extend six requirements. This conspicuousness occurred at all different timers.

All in all, 12 of the 17 existing requirements had to be altered and three additional requirements had to be added. The derived requirements in combination with the enumerated sequences were the basis of the test model. Executing the test cases showed no errors as requirements and implementation had already been corrected during the enumeration process.

Timer	Model Figures		Original Test Coverage	
	Operational States	Operational State Transitions	State Coverage	Transition Coverage
TimerA	11	83	9 ($\approx$ 82%)	9 ($\approx$ 11%)
TimerB	30	231	6 (20%)	10 ($\approx$ 4%)
TimerC	45	301	24 ($\approx$ 71%)	34 ($\approx$ 17%)
Total	86	615	39 ($\approx$ 45%)	53 ($\approx$ 9%)

TABLE II: Original test suite measurements

The most interesting fact was found when a comparison between the existing and the new test suite was made. The existing test suite turned out to be incomplete w.r.t. the possible legal and expected uses, as only 45% of the the possible operating states and only 9% of the operational transitions were reached. Table II lists some detailed numbered for each of the three timers. Even though, the existing test suite reached a code coverage of 100%. This finding highlights the importance and significance of a complete and consistent requirements specification, followed by a good test strategy with a suitable stopping criterion. Evidently modern code generators, due to their optimizations for embedded targets,

reduce control flow. Code coverage measures catch control flow paths and decisions. In the optimized code decisions are carried out on bit level and with bit shifts, that are not covered by control flow measurements.

VI. Conclusion

The development of increasingly complex embedded systems in industry is fostered by standards for the architectural specification and exchange of software components. International norms like the ISO 26262 state requirements for dealing with safety critical systems in their complete life cycle.

The methods for verification and validation of software were substantiated in 2013, when the first, second, and third part of the international standard for software testing ISO/IEC/IEEE 29119 was released. It states the main purpose of requirements based testing as 'to determine whether the test item meets end-user requirements' [1, Part 1, p.31]. In the same section it remarks that testing may suffer if the requirements are incomplete or not consistently specified.

In this paper we presented our approach to facilitate the creation of a complete, traceably correct, and consistent requirements model in industry. By this approach, the subsequent activities for quality assurance, such as validation and verification, measurement of coverage criteria, and estimators e.g. of the reliability profit from this approach, as they rely on a provably correct basis.

In the past, the application of constructive requirements modeling was often too cumbersome in industry, because it can result in a rapid growth of the state space. To overcome this, we introduced techniques, that allow for an early reduction of the modeled state space. These techniques include an a-priori analysis of legal input/output dependencies and a separate constructive modeling of the identified independent input partitions. The resulting model is a composition of all models. In doing so, the creation of the model is de-composed into manageable pieces. Hence, the creation of the model is feasible for real applications in industry.

We presented the application of this approach on a real automotive embedded system in AUTOSAR with timing requirements. During the creation of the model, inconsistencies and flaws in the requirements were identified. These were clarified in collaboration with the function owner and new requirements had to be derived.

The system was already tested before, with the test stopping criterion of 100% branch Code Coverage. We mapped the reached operating states for 100% Code Coverage to the model. The analysis revealed, that only 45% of the operating states from a usage view were reached. This implies, that 55% of all possible use scenarios of the system were untested.

We derived a test suite to cover all identified operating states. The test suite revealed a software failure w.r.t. the completed requirement specification. This result affirms the statement of the ISO29119, that testing may suffer if the requirements are not consistent or incomplete [1, Part 1, p.31]. It highlights the need for a high quality of the requirement engineering activities, as they impact all subsequent activities in the development.

The determination of estimators of the reliability with Markovian computations is well understood for models created in the classical way. In the next step, we examine how computations and metrics hold for the new composed model and how they can be determined.

References

[1] "Software and Systems Engineering Software Testing Part 1-3," *ISO/IEC/IEEE 29119-1-3:2013(E)*, pp. 1–138, Sept 2013.

[2] J. Tretmans and E. Brinksma, "Côte de resyste : Automated model based testing," in *3rd PROGRESS Workshop on Embedded Systems*, M. Schweizer, Ed. Utrecht: STW Technology Foundation, 2002, pp. 246–255.

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

[4] S. Prowell and J. Poore, "Computing System Reliability using Markov Chain Usage Models," *J. Syst. Softw.*, vol. 73(2), pp. 219–225, 2004.

[5] W. J. Gutjahr, "Software dependability evaluation based on markov usage models," *Performance Evaluation*, vol. 40, no. 4, pp. 199–222, 2000.

[6] H. D. Mills, "Stepwise refinement and verification in box-structured systems," *Computer*, vol. 21, no. 6, pp. 23–36, Jun. 1988.

[7] AUTOSAR Munich, *AUTOSAR 4.0, AUTomotive Open System ARchitecture*, 2010.

[8] A. Michailidis, U. Spieth, T. Ringler, B. Hedenetz, and S. Kowalewski, "Test Front Loading in Early Stages of Automotive Software Development based on AUTOSAR," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 435–440.

[9] S. Siegl, K.-S. Hielscher, R. German, and C. Berger, "Formal Specification and Systematic Model-Driven Testing of Embedded Automotive Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '11, 2011.

[10] D. Harel, "Statecharts: A visual formulation for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.

[11] K. Madhukar, R. Metta, P. Singh, and R. Venkatesh, "Reachability verification of rhapsody statecharts," in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, March 2013, pp. 96–101.

[12] S. Sandberg, "Homing and Synchronizing Sequences," in *Model-Based Testing of Reactive Systems*, ser. Lecture Notes in Computer Science, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds. Springer Berlin / Heidelberg, 2005, vol. 3472, pp. 39–49.

[13] S. Siegl, K.-S. Hielscher, and R. German, "Model Based Requirements Analysis and Testing of Automotive Systems with Timed Usage Models," in *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE '10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 345–350.

[14] T. Henzinger, "The theory of hybrid automata," in *Logic in Computer Science, 1996. LICS '96. Proceedings., Eleventh Annual IEEE Symposium on*, Jul 1996, pp. 278–292.

[15] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*. New York, NY, USA: Springer-Verlag New York, Inc., 1992.

[16] E. Lehmann, "Time Partition Testing: Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen," Ph.D. dissertation, Technische Universität Berlin, 2003.

[17] P. Bellini, R. Mattolini, and P. Nesi, "Temporal logics for real-time system specification," *ACM Comput. Surv.*, vol. 32, no. 1, pp. 12–42, Mar. 2000.

[18] K. Sayre and J. Poore, "A reliability estimator for model based software testing," *Software Reliability Engineering, 2002. ISSRE 2002. Proceedings. 13th International Symposium on*, pp. 53–63, 2002.

[19] S. J. Prowell and J. H. Poore, "Foundations of sequence-based software specification," *IEEE Trans. Softw. Eng.*, vol. 29, no. 5, pp. 417–429, 2003.

[20] The Mathworks, *MATLAB/Simulink*, 2010.

[21] University of Tennessee, *Sequence-based Specification Tool, Protoseq*. Software Quality Research Laboratory, 2005.

[22] *REAL Sequence-based Specification Tool*. Software Quality Research Laboratory, 2005.

[23] E. Bringmann and A. Krämer, "Systematic Testing of the Continuous Behavior of Automotive Systems," in *Proceedings of the 2006 international workshop on Software Engineering for Automotive Systems*, ser. SEAS '06. New York, NY, USA: ACM, 2006, pp. 13–20.