

Systematic Model-Based Testing of Embedded Control Software: The MB³T Approach

Mirko Conrad, Ines Fey
DaimlerChrysler AG, Research and Technology
{Mirko.Conrad / Ines.Fey}@DaimlerChrysler.com

Sadegh Sadeghipour
IT Power Consultants
Sadegh@itpower.de

Abstract

The software embedded in automotive control systems increasingly determines the functionality and properties of present-day motor vehicles. However, the development and test process of the software-based systems has become an ever more limiting factor. While these challenges, on the development side, are met by employing model-based specification, design, and implementation techniques, satisfactory solutions on the testing side are slow in arriving. With regard to the systematic test design and the description of test scenarios especially, there is a lot of room for improvement.

This paper introduces the model-based black-box testing (MB³T) approach in order to effectively minimize these deficits by creating a systematic procedure for the design of test scenarios for embedded automotive control software and its integration in the model-based development process. The MB³T approach which has recently been successfully employed in a number of projects at DaimlerChrysler makes it possible to define test scenarios for software developed in a model-based way from different perspectives and also to create consistency between them.¹

1. Introduction

A large part of today's innovation in the automotive industry is achieved by extending the functionality of vehicle software [21]. The software's increase in scope and the increase in complexity connected with it, call for new ways of dealing with the development and testing of embedded software.

On the development side, these challenges have been met, since the mid 1990s, by a paradigm shift in automotive software development. This leads to traditional, document-based software development in the engine/powertrain, chassis and body/comfort vehicle

subsystems being increasingly displaced by model-based development [4,15,25,28].

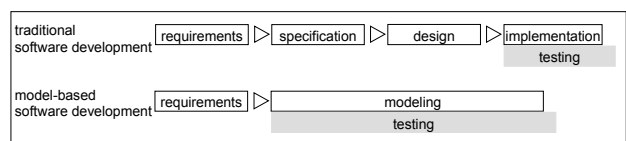


Figure 1. Traditional vs. model-based software development

Analogous to traditional development, the model-based development process starts with a requirements phase, in which the requirements of the functionality to be realized are specified textually by using tools such as DOORS [10]. Following that, this innovative development approach is characterized by the integrated deployment of executable models for specification, design and implementation, using commercial modeling and simulation environments such as Matlab/Simulink/Stateflow [19] or ASCET-SD [2]. These tools use block diagrams and extended state machines as modeling notations.

Very early in this development procedure an executable model of the control software (functional model) is made available, which can be simulated as well as tested. This executable model is used throughout the downstream development process and forms the 'blueprint' for the automatic or manual coding of the embedded software and its integration into the electronic control unit (ECU). As compared to traditional software development, where phases are clearly separate, a closer integration of the specification, design and implementation phases can be seen in model-based development (Figure 1). The seamless utilization of models facilitates highly consistent and efficient development.

Also within the framework of model-based development it is essential to subject the software being developed to an appropriate combination of verification and validation measures in order to detect errors and produce confidence in its correct functioning. In the industrial field, dynamic testing forms the focal point of analytical quality assurance. Since the executable model

¹ The work described was partially performed as part of the IMMOS project funded by the German Federal Ministry of Education and Research (project ref. 01ISC31D).

could be exploited as an additional, comprehensive source of information for testing, new possibilities and synergy potentials for the test process arise in the context of model-based development. From an efficiency point of view, one should make the most of these possibilities.

This kind of test process which is closely interlinked with model-based development, encompasses a combination of different test methods which complement each other and utilizes the executable model as a rich source of information for the test, is called a model-based test. So, the automotive view of model-based testing [16,25] is a rather process-oriented one. In particular, no additional models are being created for test purposes, but the functional models already existent within model-based development are used for the test (cf. [11]).

Satisfactory solutions for model-based testing are slow in arriving. With regard to the systematic design of test scenarios in particular, there is a lot of room for improvement. In order to make a systematic model-based test possible, a test design process from different perspectives, and with a subsequent consistency check is presented in this paper. The following description of the model-based black-box testing (MB³T) approach explicates this basic concept.

2. Model-based black-box testing

In order to define tests capable of detecting errors and producing confidence in the software, test scenarios should be designed which are systematically based on the software requirements. In the case of the test scenarios being directed only towards the technical realization of the test object, there would be the danger of neglecting and not adequately testing the original requirements made on the test object.

However, requirement-based, i.e. logical, test scenarios are abstract and not executable. This means that they cannot be used directly for test execution. Therefore, additional executable test scenarios are needed, which can stimulate the interface of the respective test object. Finally, it should be checked, whether or not the logical test scenarios are fully covered by the executable ones. These demands led us to define the model-based black-box testing approach, which has recently been successfully employed in a number of embedded control software development projects at DaimlerChrysler (cf. [9]).

The MB³T approach makes it possible to define test scenarios for software developed in a model-based way from two different perspectives and to create consistency between both (cf. Figure 8):

① **Requirement-based test design.** Early in development, logical test scenarios are defined, based on the textual requirements specification of the embedded

software. The requirement-based test scenarios created in this way by means of the classification-tree method are specified at a high level of abstraction and do not contain any implementation details for the test object. Due to their close link to the requirements it is easy to check which requirements are covered by which test.

② **Model-based test design.** Once the functional model is available, executable test scenarios are derived from it with the help of the classification-tree method for embedded systems. These are tailored to the functional model's interface.

③ **Consistency check.** By means of a set of checking rules, the consistency between logical, requirement-based and executable model-based test scenarios can be checked and thus guaranteed.

The three above-mentioned steps will be described in more detail in the following subsections and illustrated using a chassis system as an example.

2.1. Requirement-based test design

In the automotive field, requirements on embedded software are usually described textually (cf. [29]).

□ *For an antilock braking system (ABS) a high-level requirement (HLR) of this kind is shown in the upper part of Figure 2.*

For the design of requirement-based tests we utilize the classification-tree method (CTM) [12]. According to this black-box testing method, the input domain of a test object is analyzed on the basis of its requirements specification with respect to various aspects regarded as relevant for the test. For each aspect, disjoint and complete classifications are formed. Classes resulting from these classifications may be further classified iteratively. The stepwise partition of the input domain by means of classifications is represented graphically as a (classification-)tree. Subsequently, test scenarios are formed by combining classes of different classifications. This is done by using the tree to define the columns of a combination table in which test scenarios are specified.

□ *HLR-1 suggests considering different speeds at the moment of braking, while HLR-1.1 restricts the speed interval to be considered. Consequently, the vehicle speed is a significant test aspect. Further, HLR-1 requires near-optimum braking performance to be achieved by the ABS system (expected test object behavior). Therefore, one also has to regard aspects impacting the braking behavior of the vehicle. These are the brake pressure initiated by the driver, the ground friction and the kind of road (straight or curved). All the test aspects mentioned are illustrated by the classification-tree in the lower part of Figure 2. Due to the abstractness and understandability of tests on this level we specify the classifications in a qualitative way, e.g. we use 'low',*

'middle' and 'high' to describe the speed classes, rather than specific values or value intervals.

Note that both the classes ' $v < v_{min}$ ' and ' $v > v_{max}$ ' have been specified in order to have a complete classification for speed. They are not used in the test scenarios for HLR-1 (they may be used for testing a different requirement, for instance).

The maximum number of test scenarios which could be formed from a classification-tree is the number of combinations of leaf classes. Since this number is generally too high (in the case of the classification-tree in Figure 2 this number is 120), a strategy for selecting a subset of whole possible tests is necessary. Such a strategy is related to the question of test depth determination which depends on the criticality of the test object, the project guidelines, the testing standards to be met as well as the decision of the test engineer.

□ For our example we need to cover all the classes of the main test aspect, i.e. vehicle speed. Therefore, as shown in Figure 2, three test scenarios are selected, each of them covering a relevant speed class.

The insights obtained during the requirement-based test design serve as the starting point for model-based testing. The classifications of the requirement-based tree determine the test aspects which have to be interpreted as model inputs. The classes indicate how the domains of the inputs considered should be partitioned. Requirement-based test scenarios describe the situations which are to be covered by model-based test scenarios in a time-independent manner.

In this way, requirement-based tests represent snapshots of system behaviour, which are refined into time-dependent test sequences during the model-based test design phase.²

2.2. Model-based test design

Model-based test design is performed using an extended version of the classification-tree method, namely the classification-tree method for embedded systems (CTM/ES) [6,7,16]. The CTM/ES is a model-oriented black-box test design technique which allows a comprehensive graphical description of time-dependent test scenarios by means of abstract signal waveforms that are defined stepwise for each model input.

The classifications of the model-based tree are the input variables of the functional model. Its classes are obtained by dividing the range of each variable into a

set of complete and non-overlapping intervals (Figure 3).

□ The upper part of Figure 3 shows the interface of the functional model as well as the model-based classification-tree for the antilock braking system ABS. The functional model's input signals constitute the test aspects for the model-based test design. In contrast to the requirement-based classification-tree (Figure 2) the classifications on this level are specified using specific values or intervals of values.

Based on the input variable partitions, the test scenarios describe the course of these inputs over time in a comprehensive, abstract manner.

□ The lower part of figure 3 shows the model-based test scenarios for ABS. The three test scenarios specified in the combination table correspond to the logical test scenarios in Figure 3. Each test scenario consists of a number of test steps, while each step is associated with a time stamp.

Time dependent behaviour, i.e. changing the values of a model input over time in successive test steps can be modeled by marking different classes of the classification corresponding to that input. Continuous changes are defined by transitions (solid connecting lines) between marks in subsequent test steps (see [6.7] for details).

□ Examples of variations of inputs over time are shifting the gear (discrete changes), as shown in Figure 3, or moving from wet to dry pavement (continuous changes).

2.3. Consistency check between requirement-based and model-based tests

MB³T also provides the tester with a set of checking rules which make it possible to examine the consistency between the abstract, requirement-based tests and the implementation-oriented, model-based tests, defined in different phases of the development process [9]. The consistency checks in their entirety ensure the necessary degree of thoroughness and completeness for the model-based testing process. These checking rules comprise checks on the classification-trees as well as on the test scenarios (Figure 4).

2.3.1. Comparison of classification-trees. The first stage of the consistency check is the comparison of the requirement-based classification-tree (R-CT) with the model-based classification-tree (M-CT).

Classification relations. The first step here is to assign the R-CT classifications (i.e. requirement aspects regarded as relevant to the test) to the respective M-CT classifications (i.e. model inputs). These have to be in a 1:1 or 1:n relation to each other. If there is no valid relation for an R-CT classification to the M-CT

² If necessary, changing test aspects could be described using additional R-CT classes, e.g. by adding a new class called 'moving from wet to dry pavement' for the classification 'ground' to the R-CT shown in Figure 2. However, we do not recommend this style of test design, since it leads to a reduction in the abstraction grade as well as in the understandability of the requirement-based tests.

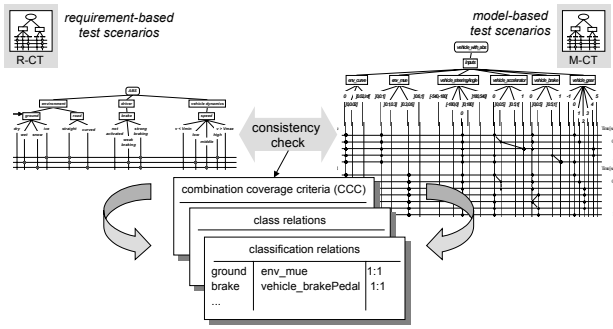


Figure 4. Consistency check between requirement-based and model-based test scenarios

classifications, this signifies that the test aspect corresponding to the model input represented by that classification was not considered when the R-CT was generated or that it is not relevant to the test after all. In the case of irrelevance, such model inputs may be set to default constant values.

Class relations. In the second step of the comparison of the classification-trees, the classes of the related classifications are compared. The possible relations between classes within a 1:1 classification relation are 1:1, 1:n, n:1 and m:n. The relation between classes within a 1:n classification relation is a little more complex: For an R-CT class the following cases for the related M-CT classes are possible:

- one class from each n related classification,
- a set of classes from each n related classification.

□ The relation between "ground" in the R-CT and "env_mue" in the M-CT as well as the relation between "road" in the R-CT and "env_curve" in the M-CT of the ABS system are examples of 1:1 relations between classifications (see Figures 2 and 3). The above-mentioned classifications represent the road friction and road curve in two different abstraction levels respectively. Figure 5 illustrates these classification relations and the corresponding 1:1 and 1:n relations between their classes.

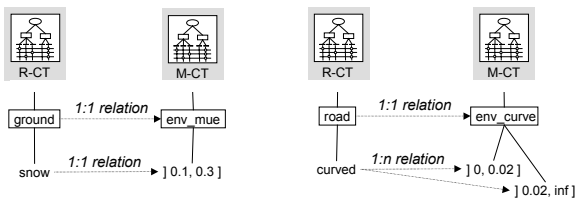


Figure 5. 1:1 classification relation as well as 1:1 and 1:n class relations between R-CT and M-CT

□ An example of a 1:n relation between classifications in both trees of the ABS system is the relation between "speed" in the R-CT and the "vehicle_accPedal", "vehicle_brakePedal" and

"vehicle_gear" classifications in the M-CT, since the vehicle speed is determined by the combination of values of the inputs mentioned. Of course, in a more complex environment model further aspects such as the road slope or the wind speed would also play a role in determining the vehicle speed and should be considered while defining the M-CT classifications related to "speed".

Figure 6 illustrates the above-mentioned classification relation and the relation between the "high" speed class and the possible sets of the corresponding classes of the "vehicle_accPedal", "vehicle_brakePedal" and "vehicle_gear" classifications. The class relation depicted makes it possible to, for example, relate the combination of the values

- $vehicle_accPedal \in]0.5, 1[$
- $vehicle_brakePedal = 0$
- $vehicle_gear = 4$

to the high speed.

Note, that this relation indicates the possibility of reaching a high speed if the corresponding classes have been marked in the M-CT (necessary condition). The question of whether the high speed is actually reached, depends on the dynamic aspect of the test scenario, i.e. the initial test step and the duration of the test step or test steps carried out with the above values (sufficient condition).³

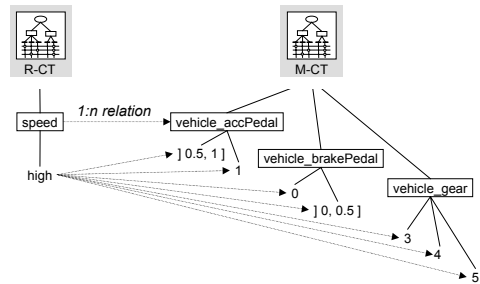


Figure 6. 1:n classification relation between R-CT and M-CT

2.3.2. Comparison of test scenarios. The second stage of the consistency check is an analysis of whether or not the requirement-based test scenarios are covered by the model-based test scenarios.

Combination coverage criteria (CCC). An R-CT test scenario is covered by an M-CT test scenario, if the combination coverage criteria for mapping R-CT test scenarios onto M-CT test scenarios, as shown in Table 1, are met. The combination coverage criteria

³ When defining the relation between the high speed and the corresponding classes of the M-CT in Figure 6, we assumed the behaviour of a 'normal' driver. It could be argued that a driver may select gear 2 when driving at high speed. Consequently, the high speed may be maintained for a few seconds, although the vehicle's gear has been set to 2. In this case, the relation depicted in Figure 6 must surely be extended to vehicle_gear from the set {2,3,4,5}.

have to be checked for all classes marked in the R-CT test scenario.

Table 1. CCC for mapping R-CT test scenarios onto M-CT test scenarios

classification relation	class relation*	combination coverage criteria
1:1	1:1 $a \leftrightarrow b$	class b is used in ≥ 1 test step of a test scenario
	1:n $a \leftrightarrow b_1 + \dots + b_n$	one of the classes b_1, \dots, b_n is used in ≥ 1 test step of a test scenario
	n:1 $a_1 + \dots + a_n \leftrightarrow b$	class b is used in ≥ 1 test step of a test scenario
	m:n $a_1 + \dots + a_m \leftrightarrow b_1 + \dots + b_n$	one of the classes b_1, \dots, b_n is used in ≥ 1 test step of a test scenario
1:n	one class from each n related classifications $a \leftrightarrow b_1 \times \dots \times b_n$	the combination of classes b_1, \dots, b_n is used in ≥ 1 test step of a test scenario
	a set of classes from each n related classifications $a \leftrightarrow (b_{11} + \dots + b_{1n}) \times \dots \times (b_{k1} + \dots + b_{kn})$	a combination of classes b_{1i}, \dots, b_{ki} is used in ≥ 1 test step of a test scenario

* a and a_i are classes of R-CT, b and b_i classes of M-CT

□ As examples for CCC consider the following cases of the R-CT and M-CT of the ABS system, shown in Figures 3 and 5:

"ground" in the R-CT and "env_mue" in the M-CT are 1:1 related classifications. The classes of these classifications, "snow" and $]0.1,0.3]$, are also 1:1 related. According to the first row of Table 1 the use of $]0.1,0.3]$ in the test steps of the third M-CT test scenario (Braking on a curve) fulfills the coverage criterion concerning the class "snow".

"road" in the R-CT and "env_curve" in the M-CT are 1:1 related classifications. The classes of these classifications, "curved" and $]0,0.02]$, $]0.02,inf[$, are 1:n related. According to the second row of Table 1 the use of $]0,0.02]$ in two test steps of the third M-CT test scenario (Braking on a curve) fulfills the coverage criterion concerning the class "curve".

"speed" in the R-CT and {"vehicle_accPedal", "vehicle_brakePedal", "vehicle_gear"} in the M-CT are 1:n related classifications. Figure 6 shows a relation between the class "high" and the corresponding sets of classes of the M-CT. According to the last row of Table 1 the use of 1 for "vehicle_accPedal", 0 for "vehicle_brakePedal" and 4 for "vehicle_gear" in the third step of the first M-CT test scenario (Strong

braking on ice) fulfills the coverage criterion concerning the class "high".

Accomplishing the combination coverage check, as shown in the above three examples, for all classes marked in the R-CT test scenarios will provide the guarantee for the consistency between the M-CT and R-CT test scenarios of the ABS system.

Note, that the consistency checks consider the test inputs only. Since the requirement-based test scenarios are non-executable, they do not produce any actual outputs. Their expected results are described textually. In contrast, model-based test scenarios produce actual results in the form of courses of output values. As a result, consistency for the test outputs must be checked manually and individually for the related test scenarios.⁴

3. Model-based test strategy

Later in the testing process, the resulting graphical test descriptions are transformed into sequences of input values over time. These sequences can be applied to the different executable artifacts of model-based development.

The test scenarios gained by using the MB³T approach contain abstracted stimulus information because only equivalent classes have been used, but no specific data. Thus, in a further step, the test data is instantiated using specific numbers (cf. Figure 8):

④ **Instantiation of test stimuli.** The test scenarios defined in the M-CT combination table form signal corridors for the individual input signals, within which the actual courses of input values must be located (cf. [6,16]). The borders of the equivalent classes form the constraints of the value ranges at the respective test steps.

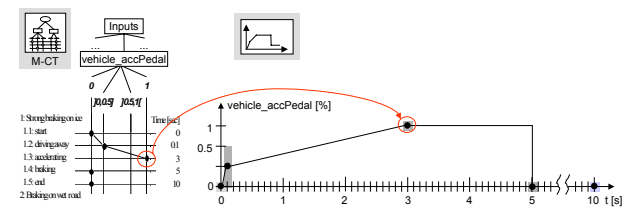


Figure 7. Instantiation of test stimuli

□ Figure 7 illustrates this with an example of the course of the "vehicle_accPedal" signal in the first test scenario of Figure 3. The gray bars show the signal corridor at the individual test step, within which one possible sequence of input values is shown. Marked classes in the combination table connected with solid

⁴ An automated test evaluation could be carried out later in the test process when back-to-back tests are performed (cf. Section 3).

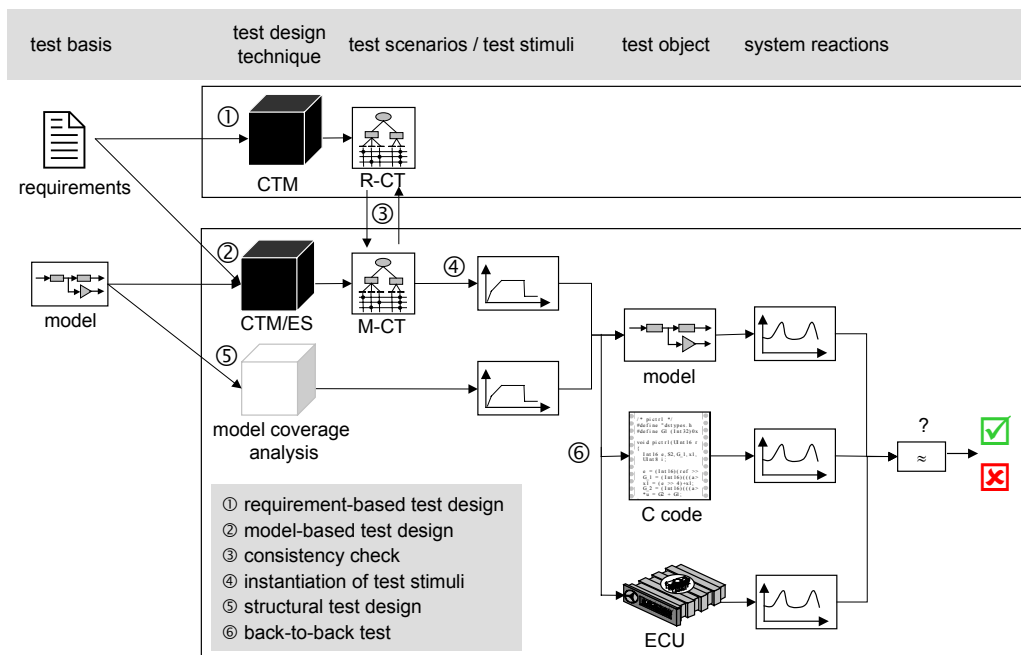


Figure 8. Integration of the MB³T approach into the model-based testing process

lines, e.g. between the rows with $t=0.1$ and $t=3$ s, correspond to ramps in the signal course. Non-connected markings, e.g. between $t=3$ and $t=5$ s correspond to a step function, i.e. a leap in the signal value at the end of the time interval.

The test scenarios systematically determined with MB³T from a black-box perspective and instantiated in the manner described above, are now applied to the executable model (Figure 8, part ④).

⑤ **Structural test design.** If the structural coverage of the model thus achieved is insufficient, the parts of the model which have not been covered are identified using a model coverage analysis [3]. Supplementary structural tests (white-box tests) are then carried out until the structure test criterion required has been reached on model level (Figure 8, part ⑤). The use of automatic test sequence generators on model level could automate this step. Benchmarks resulted in coverage degrees between 80% and 100% for C1 / D1 model coverage [3] and 70 and 100% for Simulink branch coverage [24].

⑥ **Back-to-back test.** Once sufficient test coverage of the model has been achieved, the functional and structural test scenarios can be reused as part of back-to-back tests to test C code and later the embedded system (ECU) so as to prove the functional equivalence between the model and the representation forms derived from it (Figure 8, ⑥). Of course, the test stimuli must be customized to the interfaces of the different test executable artifacts. This is, however, a straightforward task.

The systematic determination of test scenarios according to functional and structural aspects as well as

the comparative test of different representation forms make it possible to effectively uncover software-related error types. Furthermore, they also check the error-free transfer of the model into C code and its embedding into the electronic control unit

4. Related work

Alternative or supplementing test design approaches for embedded control software, especially for reactive systems, are described in [13, 17, 23].

I/O oriented test design within the STEP-X test framework [14,21] is based on a simplified version of the model-based test design approach described in Section 2.2.

Further research in the area of model-based tests deals with the execution of model coverage analysis and the analysis of different structural coverage criteria on model level [1,3,24].

The reuse of test scenarios for the different executable artifacts in model-based development is discussed in [27], the automation of signal comparison in the context of back-to-back tests is the subject of [8].

Integrated tool support for the model-based testing process is made available by MTest [16,22]. Another tool for model-based testing is MATT [18].

An automation of structural test design on model level is possible with the Reactis Tester [26]. Test results of back-to-back tests can be analyzed with the help of MEval [20].

5. Summary

Model-based black-box testing (MB³T), which integrates requirements-based and model-based testing, enhances the model-based development of embedded automotive software with a systematic approach for deriving tests from different perspectives. Test scenarios that are derived in a model-based way are interpreted as executable concretizations of abstract test scenarios derived from the requirements. By using the executable model as an additional source of information for the testing process, MB³T provides a solution, proven in practice, to the challenges of model-based testing in the automotive industry.

Requirement-based test design with the aid of the classification-tree method allows the systematic creation of logical test scenarios early in the development process and assures sufficient requirements coverage.

Model-based test design with the classification-tree method for embedded systems guarantees the systematic derivation of time-variant test scenarios for the executable model. As this method is based on the data-oriented partitioning of the input domain into equivalence classes, the data range coverage of the test scenarios can easily be achieved.

A methodology for analyzing the relations between requirement-based and model-based test scenarios, based on the graphical notations used by the classification-tree method, establishes a close link between tests derived from requirements and tests derived from the model and thus reduces the gap between these different types of test scenarios.

Following black-box test design with the MB³T approach, the test scenarios are instantiated to create specific test stimuli and used for testing the model, the C code derived from it and finally the electronic control unit. If necessary, additional white-box tests should be determined, until a suitable structure test criterion on model level has been fulfilled.

The MB³T approach leads to flexible m:n relations between the requirements and the test scenarios thus created. This represents an improvement in comparison to the rigid 1:1 relation which is currently much in use. Application to different modules within a development project at DaimlerChrysler led to between 1 and 2.16 requirement-based test scenarios and between 0.52 and 1 model-based test scenarios per requirement.

Furthermore, requirements tracing will be facilitated, since it is much easier to understand which requirements are covered by which executable tests by using the intermediate stage of requirement-based tests.

The MB³T approach and its integration into the model-based testing process is largely tool-supported by the MTest tool [16,22]. Within MTest the classification-tree editor CTE/ES [5] is used for all test design activities.

The comparatively greater amount of effort required, resulting from the tests being designed from different perspectives, leads to a better and more systematic way of generating tests which, in turn, ensure test coverage with regard to complementary coverage criteria. Thus, the MB³T approach should, in particular, be employed for software developed in a model-based way which has high safety and reliability requirements.

6. References

- [1] W. J. Aldrich, "Using Model Coverage Analysis to Improve the Controls Development Process", *AIAA Modeling and Simulation Technologies Conference and Exhibition*, Monterey, US, 2002.
- [2] ASCET-SD, ETAS GmbH, de.etasgroup.com/products/ascet/sd/.
- [3] A. Baresel, M. Conrad, S. Sadeghipour, and J. Wegener, "The Interplay between Model Coverage and Code Coverage", *11. Europ. Int. Conf. on Software Testing, Analysis and Review (EuroSTAR 03)*, Amsterdam, NL, 2003.
- [4] M. Broy, "Automotive Software Engineering", *25th Intl. Conference on Software Engineering (ICSE 03)*, Portland, US, 2003, pp. 719-720.
- [5] CTE for Embedded Systems, Razorcat Development GmbH, www.razorcat.com.
- [6] M. Conrad, "Beschreibung von Testszenarien für Steuergerätesoftware - Vergleichskriterien und deren Anwendung", *VDI-Berichte, vol. 1646*, VDI Verlag, 2001, pp. 381-398.
- [7] M. Conrad, H. Dörr, I. Fey, and A. Yap, "Model-based Generation and Structured Representation of Test Scenarios", *Workshop on Software-Embedded Systems Testing (WSEST)*, Gaithersburg, US, 1999.
- [8] M. Conrad, I. Fey, and H. Pohlheim, "Automatisierung der Testauswertung für Steuergerätesoftware", *VDI-Berichte, vol. 1789*, VDI Verlag, 2003, pp. 299-315.
- [9] M. Conrad, I. Fey, and S. Sadeghipour, "Systematic Model-Based Testing of Embedded Automotive Software", *Int. Workshop on Model Based Testing*, Barcelona, ES, 2004.
- [10] DOORS, Telelogic AB, www.telelogic.com/products/doorsers.
- [11] I.K. El-Far, and J.A. Whittaker, "Model-based Software Testing", in: J.J. Marciniak (Ed), *Encyclopedia of Software Engineering*, 2nd edition, Wiley, 2001.
- [12] M. Grochtmann, and K. Grimm, "Classification Trees for Partition Testing", *Software Testing, Verification and Reliability*, vol. 3, 1993, pp. 63-82.
- [13] G. Hahn, J. Philipps, A. Pretschner, T. Stauner, "Prototype-Based Tests for Hybrid Reactive Systems", *14. IEEE Intl. Workshop on Rapid System Prototyping*, San Diego, US, 2003.
- [14] M. Horstmann, E. Schnieder, P. Mäder, S. Nienaber, and H.-M. Schulz, "A framework for interlacing Test and/with

Design“, *ICSE 2004 workshop on Software Engineering for Automotive Systems*, Edinburgh, UK, 2004.

[15] T. Klein, M. Conrad, I. Fey, and M. Grochtmann, “Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler“, *Modellierung 2004*, Marburg, D, 2004.

[16] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, and I. Fey, “Model-based testing of embedded automotive software using MTest“, *SAE World Congress*, Detroit, US, 2004.

[17] E. Lehmann, “Time Partition Testing: A Method for Testing Dynamic Functional Behaviour“, *TEST2000*, London, UK, 2000.

[18] *Matlab Automated Testing Tool*, The University of Montana, www.cs.umt.edu/RTSL/matt.

[19] *Matlab/Simulink/Stateflow*, The MathWorks Inc., www.mathworks.com/products.

[20] *MEval*, IT Power Consultants, www.itpower.de/meval.html

[21] M. Mutz, M. Harms, M. Horstmann, M. Huhn, G. Bikker, C. Krömke, K. Lange, U. Goltz, E. Schnieder, and J.-U. Varchmin, “Ein durchgehender elektronischer Entwicklungsprozess für elektronische Systeme im Automobil“, *VDI Berichte, vol. 1789*, VDI-Verlag, 2003, pp.43-75.

[22] *MTest*, dSPACE GmbH, www.dspace.de/ww/en/pub/products/prodover/expsoft/mtest.htm

[23] W. A. Pretschner, “Zum modellbasierten funktionalen Test reaktiver Systeme“, PhD Thesis, TU München, Dept. of Computer Science, 2003.

[24] S. Ranville, “MCDC Unit Test Vectors From Matlab Models – Automatically“, *Embedded Systems Conference*, San Francisco, US, 2003.

[25] A. Rau,, “Model-Based Development of Embedded Automotive Control Systems“, *PhD Thesis*, Univ. of Tübingen Dept. of Computer Science, 2002.

[26] *Reactis Tester*, Reactive Systems Inc., www.reactive-systems.com.

[27] E. Sax, J. Willibald, and K. D. Müller-Glaser, “Seamless testing of embedded control systems“, 3. *IEEE Latin-American Test Workshop*, Montevideo, Uruguay, 2002

[28] J. Schäuuffele, and T. Zurawka, *Automotive Software Engineering*, Vieweg Verlag, 2003.

[29] M. Weber, and J. Weisbrod, “Requirements Engineering in Automotive Development - Experiences and Challenges“, *IEEE Software*, Jan/Feb. 2003, pp. 16-24.