Workshop on Software-Embedded Systems Testing (WSEST), Gaithersburg (MD), Nov. 1999

Model-based Generation and Structured Representation of Test Scenarios

Mirko Conrad, Heiko Dörr, Ines Fey, Andy Yap

DaimlerChrysler AG, Research and Technology
Software Technology Research Lab
Alt-Moabit 96A, D-10559 Berlin, Germany

{Mirko.Conrad | Heiko.Doerr | Ines.Fey | Andy.Yap}@DaimlerChrysler.com

Abstract

The interaction between various steps during the development of embedded systems is currently low. There is a large potential of cost reduction and increase in quality by removing redundant development activities. This potential is exploited by coupling modeling and testing activities.

This paper introduces a specific approach to model-based testing. This strategy drastically reduces the effort of testing and increases the quality of the testing process. The different, partially automated steps of testing based on Matlab/Simulink models and the current tool support are presented. Observing that the design of the test scenarios is the most important prerequisite for a trustworthy test the focus of this paper is on the generation and structured representation of test scenarios.

1. Introduction

The current development of embedded software applies various techniques with low interaction. Hence, there is a large potential to increase reliability and efficiency by integration of various techniques into a common platform. The integration must not only be conceptional but also implemented by using appropriate tools, which are acceptable on an industrial scale. This paper addresses the integration of testing and modeling for embedded systems. Though testing is the most powerful technique for assuring high quality, it is time consuming. Therefore the automation of testing is an important goal.

Model-based testing takes advantage of the fact that a model of any system under development captures a lot of information which is also relevant for testing. For instance the information on data flow or system behavior can be used to generate test sequences. Based on this observation, the Software Technology Research Lab of DaimlerChrysler has developed a pragmatic approach to model-based testing for embedded systems based the classification-tree method.

The goal of this work is first to provide a powerful tool support and second, to integrate this tool with current development and simulation tools such as Matlab/Simulink/Stateflow, MatrixX/SystemBuild or Statemate Magnum. A first prototype using Matlab/Simulink/Stateflow /ML,SL,SF/ models has been realized.

Our approach to a model-based generation of test scenarios utilizes the classification-tree method /GG93/. This method is a specific instance of black-box partition testing partly using and improving ideas from the category partition method /OB88/. The classification-tree method has been used successfully in various application fields at DaimlerChrysler. Commercial tool support is available with the classification-tree editor CTE /GGW93,ATS/. The basic idea of the classification-tree method is to partition the input domain of the test object separately under different aspects. Then the different partitions are recombined to form test cases which could be sequenced into test scenarios. Adapting this basic idea, our approach consists of two major steps:

1. Selecting and Structuring Test Aspects
2. Determining Test Scenarios

Both steps are dealt with on the basis of the functional specification and the interface description of the test object. A Simulink or Stateflow subsystem block serves as executable specification of the test object and provides information about its interface (Figure 1).

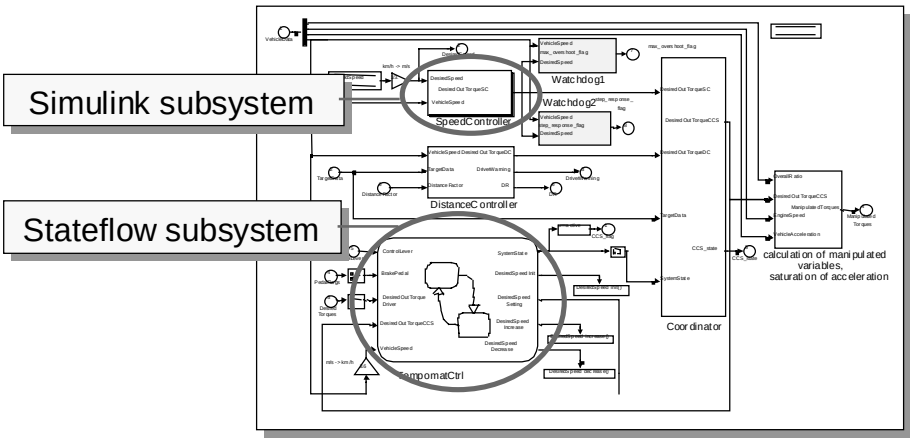

Figure 1: Executable Specification with Simulink and Stateflow Subsystems (Example)

2. Selecting and Structuring Test Aspects

In the first step an interface-based classification-tree is generated semi-automatically on the basis of the Simulink model of the test object.

By application of the classification-tree method, the input domain of the test object is analyzed (based on its functional specification) with respect to various aspects considered to be 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 and structured graphically by a tree. The selection and structuring of the test aspects is supported by the so-called ModelExtractor tool.

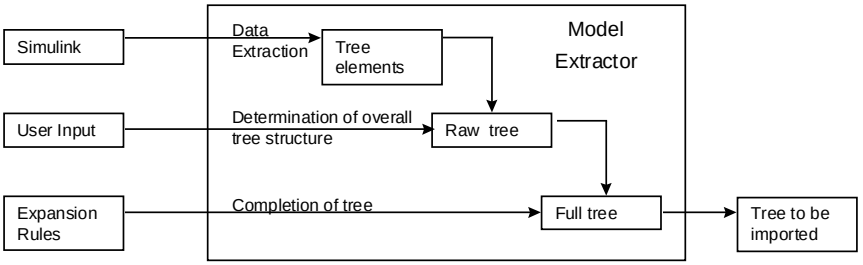

Figure 2: Generating Classification-Trees **-** Data Flow of ModelExtractor

The ModelExtractor generates for the present a raw classification-tree by using two sources of information (Figure 2):

First, the Simulink model of the subsystem under test (particularly its input interface and its internal parameters) is analyzed systematically with respect to relevant test aspects which are extracted as tree-elements. The name of the test object itself forms the root node. Input signals, elements of input busses and internal model parameters are extracted as classes. To preserve the structure of hierarchical inputs (busses), the structure of busses and vectors is also extracted (Figure 3).

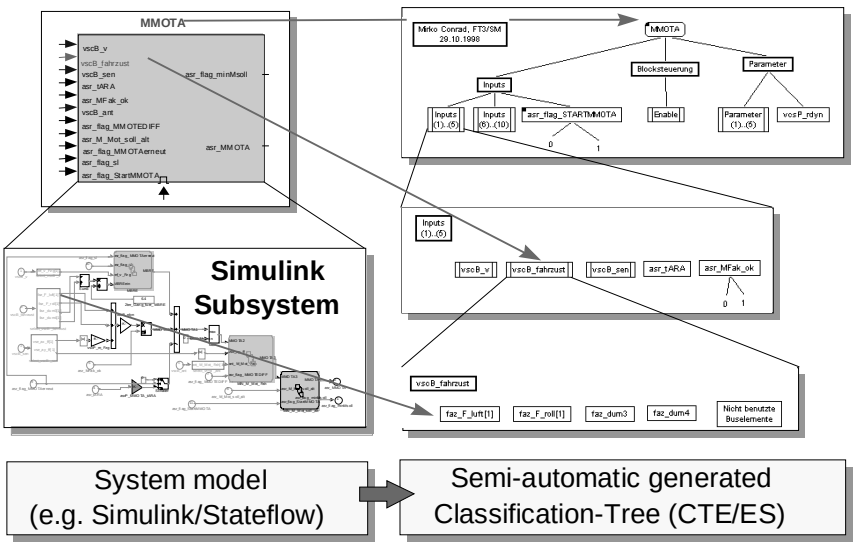

Figure 3: Generating Classification-Trees **-** Data Extraction from Simulink

Second, the overall tree structure can be predefined by the tester in a special GUI. Based on these user preferences additional nodes are inserted for structuring the classification-tree, thus improving the readability and allowing an adaptation to domain-specific constraints (Figure 4).

The generated tree elements, describing the input data space of the model, and the user defined tree elements for structuring the tree are processed by the ModelExtractor which fully automatically outputs a first instance of a classification-tree called raw tree.

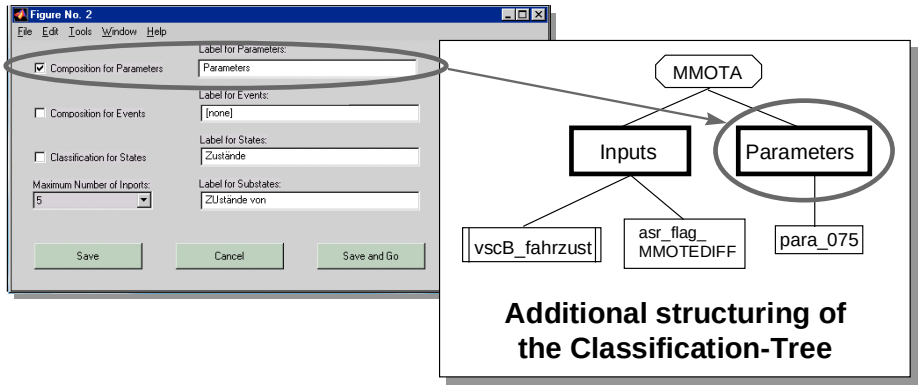

Figure 4: Generating Classification-Trees - Determination of Overall Tree Structure

After that, the raw tree generated by the ModelExtractor will be expanded according to tree-transformation rules (Figure 2). The application of these rules, which are stored in an experience base, adds e.g. useful standard classes for recurrent classifications or even entire sub-trees. Figure 5 shows an example for a transformation rule and its application to the raw tree. The automated, rule-based expansion enables the user to share the knowledge of experienced testers and to concentrate on non-routine activities

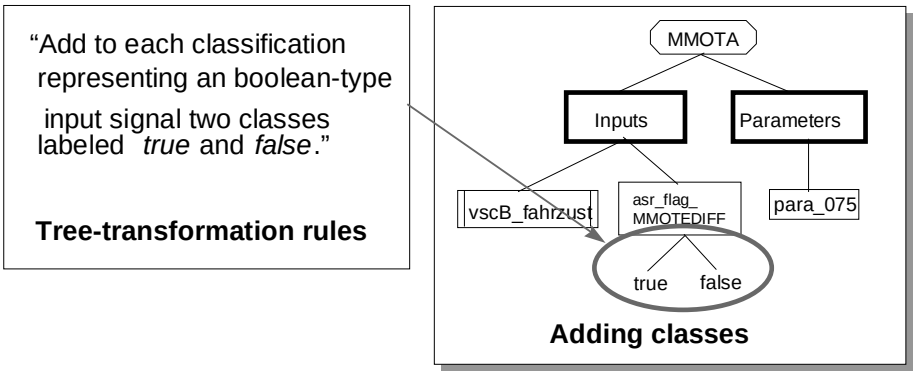

Figure 5: Generating Classification-Trees - Rule Based Expansion (Example)

The expanded classification-tree may be imported into an extended classification-tree editor, called CTE for embedded systems or short CTE/ES. Either a fully automatic import of the entire tree or a flexible semi-automatic import of sub-trees or single nodes, which may be edited further by the user, is possible (Figure 6).

The automated generation of classification-trees from the technical interface of the test object was suggested for a long time, an experimental tool employing artificial intelligence techniques was already constructed in 1993 /GG93/. Later, concepts for the classification-tree construction based on particular specification notations were developed (e.g. for Z specifications /SCS97/ and prototypically realized /Wei97/.

Since the model-based development of embedded software supported by powerful modeling tools as Matlab/Simulink/Stateflow became available, the integration of such technologies in the industrial development engineering process is possible. Furthermore, development engineers accept the approach because information required for generating the classification-trees evolve within the normal development procedure and do not need to be entered twice.

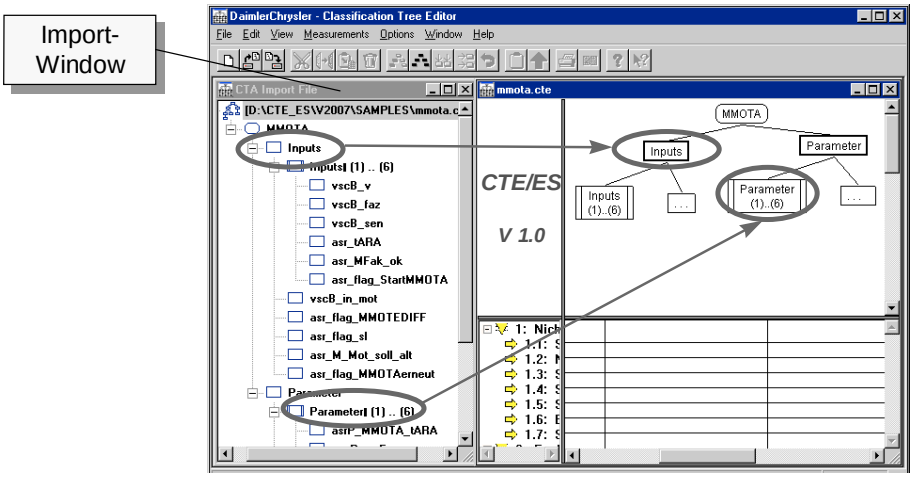

Figure 6: Classification-Tree Editor CTE/ES with Import Interface

3. Determining Test Scenarios

Testing the behavioral properties of the unit under test requires the determination of test scenarios. Their systematic determination forms the second major step of our approach.

Each scenario captures a particular behavior of the environment which should be presented to the unit under test and, hence, describes – largely independent from concrete data – what is to be tested. In the execution of the specific test the environment must exhibit the respective behavior and drive the unit under test.

To represent test scenarios in an abstract way, they are decomposed into individual test steps. Each test step defines the input situation at a certain time. A sequence of such test steps is called test sequence.

The terminology for the description of test steps and test sequences is provided by the classification-tree. The tree is used as head of the so-called combination table (Figure 7). Whereas the classification-tree is a systematic description of aspects relevant to the test, the combination table is used for the representation of the hierarchical test sequences.

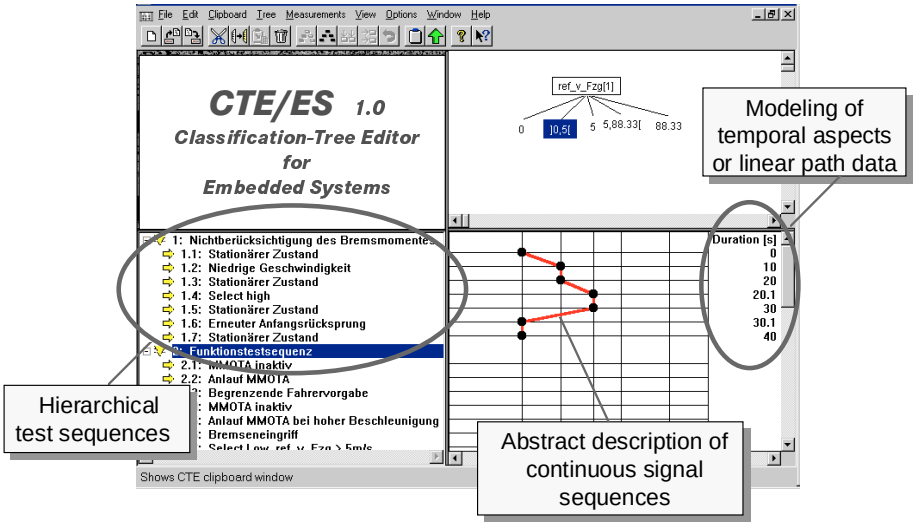

Figure 7: Determination of test scenarios and description of test sequences in the CTE/ES

Since each test sequence corresponds to a run of the test object, the tester has on one hand to define input situations for selected points in time and on the other hand to describe the type of signal course in between in order to describe this run in an abstract way.

The input situation for each test step is defined by combining classes of different classifications of the classification-tree. This is done by marking the appropriate tree elements in the combination table.

This leads to a series of stimuli, which may still be activated at any time during the test. For the definition of a test scenario the stimuli must be ordered and their duration must be specified.

Therefore, the duration of each input situation can be captured by the annotation of time indications at the rightmost column of the combination table (Figure 8). So, settling times of a controller spent in a test step can be taken into account.

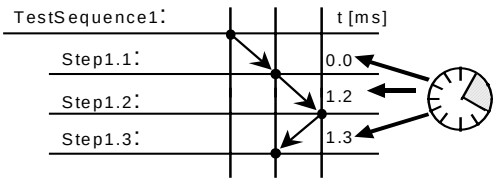

Figure 8: Description of Test Sequences - Annotated Time Indications (Example)

Signal courses are represented by arrows which connect the marks of successive steps in the combination table. A fixed number of course types like step, ramp, and spline is predefined within the CTE/ES. By default the transition between two test steps is performed as a step function. All other types are represented by specific arrow types (Figure 9). This enables a clear graphical differentiation of the transition types. Hereby, discrete as well as continuous signals may be modeled.

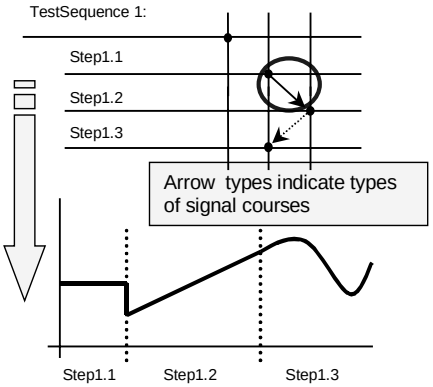

Figure 9: Description of Test Sequences - Types of Signal Courses in the Combination Table

4. Further Test Steps

The test sequences are the input for the subsequent steps of the model based testing approach: With the help of an export function, requirements for the test data description are extracted from the CTE/ES and provided to the simulation tool. Within a graphical test data editor input signal sequences are defined concretely for the simulation of the test object and test drivers with stimuli and evaluation blocks are automatically generated. During the test of the model they stimulate the model directly in the simulation environment with the defined input signals (Figure 10).

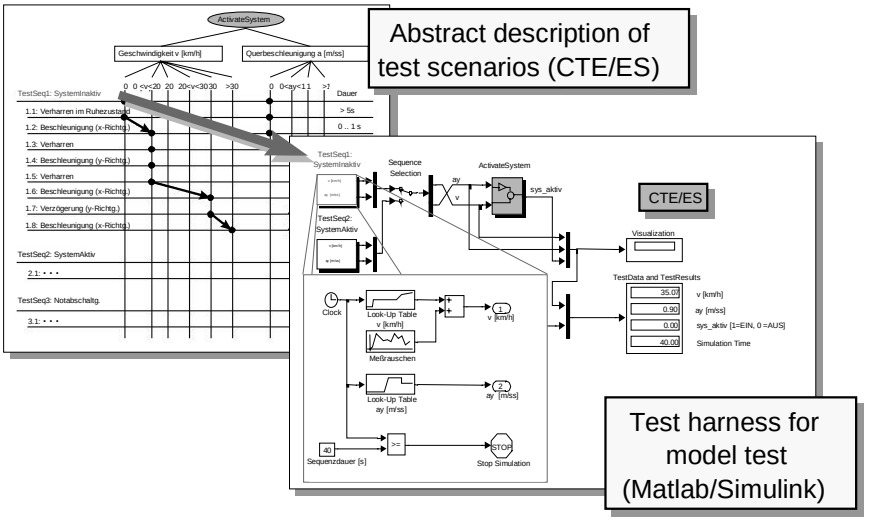

Figure 10: Further Steps in Model-based Testing - From Abstract Description of Test Scenarios to Generation of Test Harnesses

5. Summary

The approach to model-based testing described here shows how information introduced in a development process can be reused by subsequent activities for the special case of modeling and testing.

This approach has two main advantages. First, reuse of information reduces development time and costs. Information already available must not be reentered into further tools but the information is exchanged between tools. Second, the quality of the overall process increases. No typos or more severe inconsistencies between modeling and testing will weaken the quality assurance. Testing

will check whether the unit under test respects precisely the requirements indicated by the model.

References

/ML/ 'Using Matlab Version 5'. The MathWorks Inc., Natick, MA, (1999)

/SL/ 'Using Simulink Version 3'. The MathWorks Inc., Natick, MA, (1999)

/SF/ 'Stateflow User's Guide Version 2'. The MathWorks Inc., Natick, MA, (1999)

/GG93/ Grochtmann, M. and Grimm, K. 'Classification Trees for Partition Testing'. Software Testing, Verification and Reliability, 3, 63-82 (1993)

/OB88/ Ostrand, T. and Balcer, M.: 'The category-partition method for specifying and generating functional tests'. Communications of the ACM, 31(6), 676-686 (1988)

/GGW93/ Grochtmann, M., Grimm, K., and Wegener, J.: 'Tool-Supported Test Case Design for Black-Box Testing by Means of the Classification-Tree Editor'. Proc. of 1st European Int. Conf. on Software Testing, Analysis and Review (EuroSTAR '93), London, UK, pp. 169-176 (1993)

/ATS/ ATS Software Research & Consulting GmbH, Schwartzkopffstr. 6, 10115 Berlin, Germany,
http://www.ats-berlin.de

/SCS97/ Singh, H., Conrad, M., and Sadeghipour, S.: 'Test Case Design Based on Z and the Classification-Tree Method'. Proc. of 1st IEEE Int. Conference on Formal Engineering Methods (ICFEM'97), IEEE Press (1997)

/Wei97/ Weiß, T. 'Testfallgenerierung auf der Basis von Z-Spezifikationen mit Hilfe der Klassifikationsbaum-Methode' (in german). Diploma thesis, Technical University Berlin (1997)