2005-01-0750

Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences

Hartmut Pohlheim, Mirko Conrad
DaimlerChrysler AG, Berlin, Germany
{hartmut.pohlheim, mirko.conrad}@daimlerchrysler.com

Arne Griep
IAV GmbH, Berlin, Germany
arne.griep@iav.de

Copyright © 2005 SAE International

ABSTRACT

Whereas the verification of non-safety-related, embedded software typically focuses on demonstrating that the implementation fulfills its functional requirements, this is not sufficient for safety-relevant systems. In this case, the control software must also meet application-specific safety requirements.

Safety requirements typically arise from the application of hazard and/or safety analysis techniques, e.g. FMEA, FTA or SHARD. During the downstream development process it must be shown that these requirements cannot be violated. This can be achieved utilizing different techniques. One way of providing evidence that violations of the safety properties identified cannot occur is to thoroughly test each of the safety requirements.

This paper introduces Evolutionary Safety Testing (EST), a fully automated procedure for the safety testing of embedded control software. EST employs extended evolutionary algorithms in an optimization process which aggressively tries to find test data sequences that cause the test object to violate a given safety requirement.

A compact description formalism for input sequences for safety testing is presented, which is compatible with description techniques used during other test process stages. This compact description allows 1) an efficient application of evolutionary algorithms (and other optimization techniques) and 2) the description of long test sequences necessary for the adequate stimulation of real-world systems. The objective function is designed in such a way that optimal values represent test data sequences which violate a given safety requirement. By means of repeated input sequence generation, software execution and the subsequent evaluation of the objective function each safety requirement is extensively tested.

The use of EST for the safety testing of automotive control software is demonstrated using safety requirements of an adaptive cruise control (ACC) system.

The EST approach can easily be integrated into an overall software test strategy which combines different test design techniques with specific test objectives.

INTRODUCTION

Many of the innovations in the automotive field are results of the use of software-intensive systems in vehicles. The proportion of functions which are activated for regulatory or controlling purposes when the vehicle is in motion has risen steeply in the last few years. Especially in the case of safety-relevant functions, the quality of the control software is of the utmost importance.

Software development is not yet at the stage at which the quality of embedded control software can be ensured using constructive procedures alone. Their use must be supplemented with analytical procedures which detect errors in the software. Since there are currently no universally recognized measurement procedures for software quality, measuring is often replaced with dynamic testing [HV98]. Functional (black-box) test design techniques are mainly used for testing embedded control software, i.e. testing that focuses on demonstrating that the implementation fulfills its functional requirements. The test cases or test scenarios are predominantly created manually. In some cases, the functional test design techniques are also complemented with structural (white-box) test methods in order to demonstrate the structural integrity of the software.

For safety-related systems this approach is not sufficient. In this case, the control software must also meet application-specific safety requirements. Such safety requirements typically arise from the application of hazard and/or safety analysis techniques, e.g. Failure Mode and Effect Analysis (FMEA) [IEC60812], Fault Tree Analysis (FTA)

[IEC 61025] or Software Hazard Analysis and Resolution in Design (SHARD) [FMN+94]. During the downstream development process it must be shown that these safety requirements cannot be violated. This can be achieved utilizing different techniques. One way of providing evidence that violations of the safety properties identified cannot occur is to thoroughly test each of the safety requirements. As is the case for any other test problem, the search for suitable test cases/test scenarios and their appropriate description is of decisive importance.

Being universally applicable strategies for improvement and optimization, Evolutionary Algorithms (EA) have become widespread in a broad range of searching problems. Their application is based on a seemingly simple and easily comprehensible principle, namely 'Darwin's evolution paradigm' or, in other words, the principle of the 'survival of the fittest'. Like in nature, solutions are improved step-by-step, and optima are identified or approximated by applying variation and selection to a population of alternative solutions [VDI/VDE 3550].

Prerequisites for the application of EA are the definition of the search space, in which solutions are searched for and of an objective function, with which the fitness of the solution proposals found can be evaluated.

The interpretation of the search for suitable test scenarios, which violate a given safety requirement, as an optimization problem and the subsequent use of EA to solve this problem leads to the concept of Evolutionary Safety Testing (EST) [Weg01]. Thereby, the search space is defined by a compact description formalism for possible input sequences.

In principle, the EST approach can be applied to different embedded control software development stages. In order to be able to incorporate the results of evolutionary safety testing early in the development process, it is, however, advisable to already apply the procedure to the executable model of the future software.

Here, EST should not be the only test design technique which is employed but rather it should be used together with others as part of an overall test strategy for automotive control software developed in a model-based way.

The remainder of the paper is structured as follows: Section 1 explains why sequence testing is necessary for cyclic software components which are typical for embedded automotive controls. Section 2 shows how evolutionary algorithms can be used to automatically generate real-world input sequences for the safety testing of embedded automotive control systems. An example is used to illustrate the entire process. In Section 3 we show how the evolutionary safety testing is incorporated into a model-based test strategy. Related work is presented in Section 4.

1 CYCLIC SOFTWARE COMPONENTS AND SEQUENCE TESTING

1.1 CYCLIC SOFTWARE COMPONENTS

As embedded controls mostly interact with the outside world continuously via sensors and actuators, they must be able to process time-variable sensor signals and produce time-variable outputs.

When analyzing the software of those systems, a large class of implementations can be identified which use a implementation scheme whereby the control algorithm is triggered at regular time intervals by its environment to compute output values and an internal state from given inputs. This class is referred to as cyclic software components (cf. [GHD98], [MK99]). In the world of automotive controls it is very common for a software function to be based on an initialization and step function. Whereas the initialization function is only called once at the beginning, the step function is executed periodically e.g. every 10 ms. An adaptive cruise control system (ACC) which checks the velocity and distance to a preceding vehicle at regular intervals to find out if a safe distance is maintained, is an example of this system class.

Example ACC

If the driver so desires, the ACC system controls the speed of the vehicle whilst maintaining a safe distance from the preceding vehicle. The activated system monitors the section of road in front of the vehicle. If there is no preceding vehicle ('target') on this section, the system regulates the longitudinal vehicle speed in the same way as a conventional cruise control system. As soon as a preceding vehicle is recognized, the distance control function is activated which makes sure that the vehicle follows the vehicle in front at the same speed and at a safe distance.

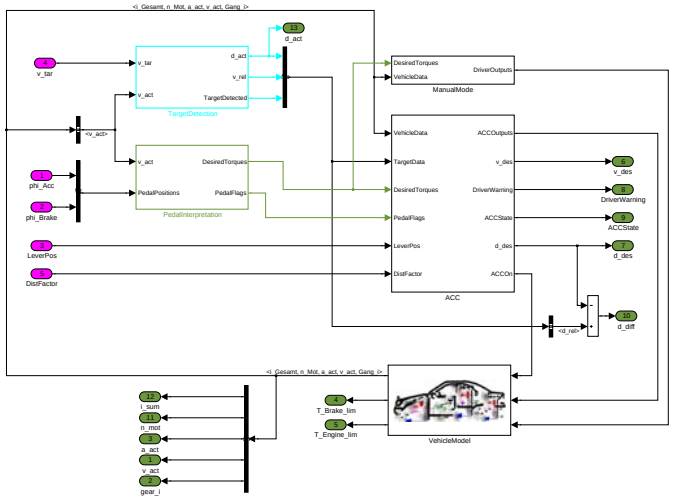

Figure 1. Simulink/Stateflow model of the ACC system

In the model-based development process [Rau03], [KCF+04], the ACC is first described using an executable Simulink/Stateflow [SLSF] model. The structure of the

overall system is presented in Figure 1. Besides the actual ACC system, there are two preprocessing systems for the recognition of preceding vehicles (TargetDetermination) and the evaluation of the current pedal values (PedalInterpretation). A system for parts of the plant model (VehicleModel, ManualMode) also exists. These make a closed loop test of the entire system, including the vehicle and the vehicle in front, possible. A more detailed description is contained in [CH98] and [Con04].

If a cyclic software component such as the ACC is to be simulated with realistic input data during the test and if a system reaction is to be induced, the test inputs must be given as time-variable signal waveforms.

1.2 SEQUENCE TESTING

In principle, the testing of cyclic software components can be performed in different ways.

Single input-state-pairs

The approach traditionally used is to generate pairs of internal states and input situations as shown in Figure 2. This means that a test case represents a certain point in time within a test sequence. For this, internal states have to be set directly. This approach works well for simple software components but raises several problems.

The direct setting of internal states is not possible in all cases and requires changes to be made to the test object. When generating the internal states, the test environment has to make sure that the states produced are valid according to the specification. Forcing the system into a generated state is, in most cases, not useful for the tester. This is because the tester has to use the test data generated to analyze software bugs. He first needs to ascertain how to produce the initial state which caused the problem. In other words, he has to manually reconstruct the initial part of the relevant sequence. This information is not provided by the test environment.

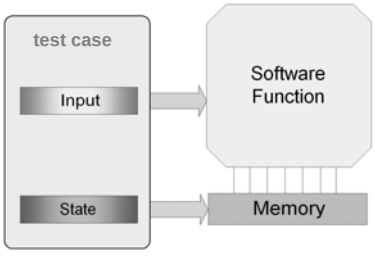

Figure 2. Automatic testing by generating (state, input) pairs

Input sequences

For complex systems it is necessary to provide real input sequences.

(a) Extensional input sequences: Within this group, one approach is to create lists of inputs for sequential calls of the function under test. This means that a test sequence is formed using a list element for each point in time in the test sequence. In other words, the test sequence is represented by enumerating / listing all of its elements (Figure 3).

As a rule, this approach leads to long and low-level test descriptions, but is often sufficient for software components based on pure state models.

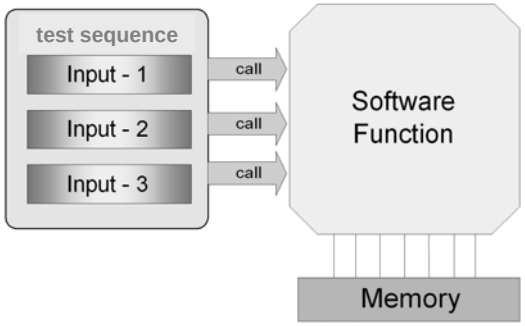

Figure 3. Automatic testing by generating list of inputs

(b) Intensional input sequences: For control systems requiring long 'real world' input sequences, an abstraction from the extensional sequence description is indispensable. Test sequences are described by their main characteristics. One way of doing this is to describe only selected points in time in a test sequence (time tags) and to specify transition functions for those values in between. This leads to an intensional or encoded sequence description (Figure 4).

Intensional descriptions can be very compact and comprehensive and are also often easier to understand for humans. A test sequence can be described using substantially less parameters than is the case for extensional input sequences.

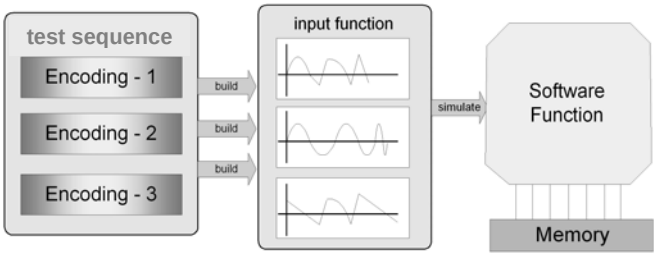

Figure 4. Automatic testing by generating input sequences from encoded input functions

The two variants for describing tests with input sequences are, from the tester's perspective, the better solution, since they guarantee that the software component is tested in the same way as it will later be used.

Test approaches which use test sequences (as opposed to single input-state-pairs) in order to stimulate the actual test object will be referred to as sequence testing (approaches) from here on. Within sequence testing, test inputs and system reactions are not considered to be static values but rather time-variable signal waveforms.

2 EVOLUTIONARY SAFETY TESTING

Evolutionary algorithms (EA) have been used to search for data for a wide range of applications. EA is an iterative search procedure using variation and selection to copy the behavior of biological evolution. They work in parallel on a number of potential solutions, the population of individuals. In every individual, permissible solution values for the variables of the optimization problem are coded. The range of possible variable values of the individuals spans the search space of the optimization problem.

The fundamental concept of evolutionary algorithms is to evolve successive generations of increasingly better combinations of those parameters which significantly affect the overall performance of a design. Starting with a selection of good individuals, the evolutionary algorithm achieves the optimum solution by exchanging information between these increasingly fit samples (recombination) and introducing a probability of independent random change (mutation). The adaptation of the evolutionary algorithm is achieved by the selection and reinsertion procedures used because these are based on the fitness of the individuals. The selection procedures control which individuals are selected for reproduction depending on the individuals' fitness values. The reinsertion strategy determines how many and which individuals are taken from the parent and the offspring population to form the next generation. The fitness value is a numerical value that expresses the performance of an individual with regard to the other individuals in the population so that different designs may be compared. The notion of fitness is fundamental to the application of evolutionary algorithms.

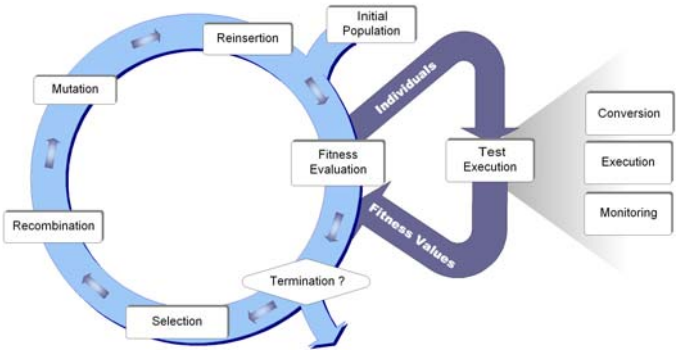

Figure 5. The structure of Evolutionary Testing

Figure 5 gives an overview of a typical procedure for evolutionary algorithms. First, an initial population of guesses as to the solution of a problem is initialized, usually at random. Each individual in the population is evaluated by calculating its objective value. Afterwards, the fitness of each individual is calculated by ranking the objective values of all the individuals of the population. The remainder of the algorithm is iterated until the optimum is achieved, or another stopping condition is fulfilled. Pairs of individuals are selected from the population according to the pre-defined selection strategy, and are combined in some way to produce a new guess in an analogous way to biological reproduction. Combination algorithms are many and varied. Additionally, mutation is applied. The new individuals are evaluated for their objective value (and fitness), and survivors into the next generation are chosen from the parents and offspring, often according to fitness though it is important to maintain diversity in the population to prevent premature convergence to a sub-optimal solution.

For the optimization we employed extended evolutionary algorithms. This includes the use of multiple subpopulations, each using a different search strategy. We also used competition for limited resources between these subpopulations.

For a detailed discussion of evolutionary algorithms and the extensions used see [Poh99]. The algorithms employed are implemented in the widely used MATLAB based 'Genetic and Evolutionary Algorithm Toolbox for use with Matlab – GEATbx' [Poh05]. It consists of a large set of operators e.g. real and integer parameter, migration and competition strategies.

When using EA for a search problem, it is necessary to define the search space and the objective (fitness) function.

2.1 EVOLUTIONARY ALGORITHMS FOR SEQUENCE TESTING

Evolutionary Testing uses EA to test software automatically. The different software test objectives formulate requirements for a test case / test sequence to be generated. Until now, the generation of such a set of test data usually had to be carried out manually. Automatic software testing generates a test data set automatically, aiming to fulfill the requirements in order to increase efficiency and resulting in an enormous cost reduction.

The general idea is to divide the test into individual test objectives and to use EA to search for test data that fulfills each of the test objectives.

In the past, evolutionary testing has proven itself to be of value for different testing tasks:

- Evolutionary temporal behavior testing [Weg01]
- Evolutionary structural testing: Evolutionary Structural Testing has the goal of automating the test case design for white-box testing criteria [BSS02]. Taking a test object, namely the software under test, the goal is to find a test case set (selection of inputs) which achieves full structural coverage

Depending on the task and test objective, both single input-state-pairs and test sequences can be created by the evolutionary test as stimuli for the test object.

Since complex dynamic systems, like the ACC, must be evaluated over a long time period (longer than the highest internal dead time or time constant), such systems must be tested by using input sequences. Furthermore, they must not be stimulated for only a few simulation steps, but rather the input signals must be up to hundreds or thousands of

time steps long. Long input sequences are therefore necessary in order to realistically simulate these systems. As a consequence, the output sequences to be monitored and evaluated have the same duration.

Example ACC

In order to check the correct reaction of the ACC system with regard to speed changes of the vehicle in front, the length of realistic test sequences has to be in the magnitude of some 10s. Assuming a sampling rate of 10ms for the cyclic ACC software component, this results in input (and subsequently) output sequences which are some 1000 time steps long.

Several disadvantages result from the length of the sequences necessary: using extensional sequence descriptions, the number of variables to be optimized and the correlation between the variables is very high. For this reason, one of the most important aims is the development of a very compact, intensional description for the input sequences. Such a description has to contain as few elements as possible but, at the same time, offer a sufficient amount of variety to stimulate the system under test as much as necessary.

Moreover, possibilities for automatically evaluating the system reactions must be developed, which allow differentiation between the quality of the individual input sequences.

The generation of test sequences for dynamic real-world systems poses a number of challenges:

- The input sequences must be long enough to stimulate the system realistically.
- The input sequences must possess the right qualities in order to stimulate the system adequately. This concerns aspects such as data type of the signal, speed and rate of signal changes.
- The output sequences must be evaluated regarding a number of different conditions / properties. These conditions/properties are often contradictory and several of them can be active simultaneously.

In order to develop a test environment for the functional test of dynamic systems which can be applied in practice, the following steps must be completed:

- definition of the test objective, i.e. the safety property to be violated,
- description of the search space, i.e. the description of the input sequences,
- description of the objective (fitness) function in order to evaluate output sequences regarding the test objective,
- assessment of counter examples generated

2.2 DEFINITION OF THE TEST OBJECTIVE

In order to utilize evolutionary sequence testing for safety testing, the test objective is to violate a safety requirement resulting from a hazard and/or safety analysis.

Example ACC

A major requirement for the ACC system is to maintain a ‘safe’ distance to the vehicle in front. More precisely the desired distance d_{des} between one’s own and the target vehicle is defined according to the German ‘Tacho-halbe-Regel’ by:

$$d_{des}\ [m] = v_{act}\ [m/s] * 3.6/2 * DistFactor$$

For normal road conditions the distance factor is 1, for wet or icy roads >1.

A derived safety requirement could be that the actual distance d_{act} is not allowed to go below the d_{des} by more than 10m. In other words:

$$d_{act}\ [m] > d_{des}\ [m] - 10$$

2.3 DESCRIPTION OF THE SEARCH SPACE

In order to define the search space, we have to deal with a number of descriptions. At the beginning, there is the compact description of the search space given by the safety tester (compact search space description). Finally, we need the individual (sampled) signals, which are used as inputs for the simulation (extensional test sequences). In between, there are the boundary descriptions of the variables to be optimized (optimization variable boundaries) and the instantiation of the individual sequences (intensional test sequences). These different sequence description levels are illustrated in Figure 6.

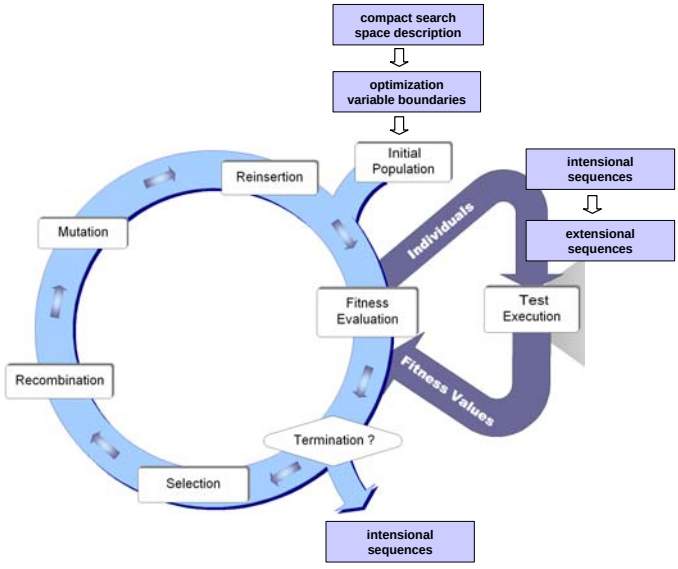

Figure 6. Different levels of input sequence description

For an intensional sequence description, each signal of a long simulation sequence is subdivided into consecutive sections. Each section is parameterized by the variables: section length, signal amplitude (at the beginning of the section) and interpolation function. Typical interpolation functions are, for instance, step, ramp (linear), impulse, sine and spline.

Only the admissible ranges for these parameters are specified for the optimization. In this way, the boundaries are defined, within which the optimization generates solutions which are subsequently evaluated by the objective function with regard to their fitness.

The signal amplitude of each section can be defined absolutely or relatively. The length of the section is always defined relatively (to ensure the monotony of the signal). The interpolation functions are given as an enumeration of possible types. These boundaries must be defined for each input of the system. For nearly every real-world system these boundaries can be derived from the specification of the system under test.

Example ACC

An example of an input sequence description in the MATLAB .m scripting language is provided in Figure 7. Figure 8 provides an example of an extensional sequence generated by the optimization based on the settings given in Figure 7.

```
% Input settings

% input names and order: 'phi_Acc', 'phi_Brake', 'Lever-
Pos', 'v_tar', 'DistFactor'

% number of sections / base points for each input signal
BasePoints = [10 10 10 10 10];

% relative section length
BasisBounds = [ 1  1  1  1  1; ...
               10 10 10 10 10];

% min. / max. amplitude for each input signal
AmplitudeBounds = [0 0 0 20 1; ...
                   0 0 2 45 1];

% possible interpolation functions for each input signal
TransitionPool = {{}; {}; {'impulse'}; {'spline'}; {}};
```

Figure 7. Textual description of input sequences

The ACC system under test has 5 inputs (see Figure 1). We use 10 sections for each sequence. The amplitude for the brake and accelerator pedal can change between 0 and 100, the control lever can only have the values 0, 1, 2 or 3.

In real-world applications the variety of an input signal is often constrained with regard to possible base signal types. An example of this is the control lever in Figure 7. This input contains only impulses as the base signal type.

To generate a further bounded description, it is possible to define identical lower and upper bounds for some of the other parameters. In this case, the optimization has an empty search space for this variable – this input parameter is a constant. An example is the distance factor in Figure 8. This input signal is always set to a constant value of 1. Thus, it is not part of the optimization (but is used as a constant value for the generation of the internal simulation sequence).

The different descriptions ensure that the requirements for an adequate simulation of the system and a very compact and comprehensible description by the tester are fulfilled. The compact description is used for the optimization, ensuring a small number of variables.

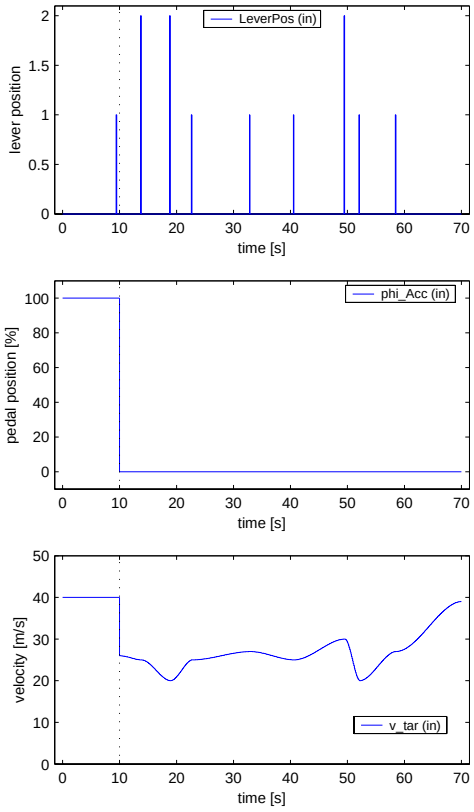

Figure 8. Simulation sequence generated by the optimization based on the textual description in Figure 7; top: throttle pedal, middle: control lever, bottom: velocity of target car

Example ACC

When comparing the size of both descriptions for the example dynamic system used (5 inputs – 3 of the inputs are constant, 10 signal sections, 2 variable parameters for each section (section length and amplitude), 60s simulation time, sampling rate 0.01s) the differences are enormous:

$$NumParameter_{Extensional} = 5 \cdot 60 \cdot (1/0.01) = 30000 \quad (1)$$
$$NumParameter_{Compact} = (5-3) \cdot 10 \cdot 2 = 40$$
$$CompressionRatio = \frac{30000}{40} = 750$$

Only this compact description opens up the opportunity to optimize and test real-world dynamic systems within a realistic time frame.

2.4 DESCRIPTION OF THE OBJECTIVE FUNCTION

The test environment must perform an evaluation of the output sequences generated by the simulation of the dynamic system. These output sequences must be evaluated regarding the optimization objectives. During the test, we

always search for violations of the defined requirements. Possible aims of the test are to check for violations of:

- signal amplitude boundaries,
- maximal overshoot or maximal settlement time.

Each of these checks must be evaluated over the whole or a part of the signal lengths and an objective value has to be generated.

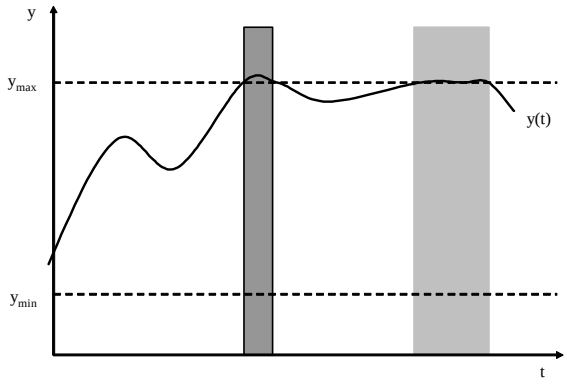

Figure 9. Violation of maximum amplitude

An example of the violation of signal amplitude boundaries is given in Figure 9. A minimal and maximal amplitude value is defined for the output signal y. In this example the output signal violates the upper boundary. The first violation is a serious violation as the signal transgresses the bound by more than a critical value y_c (parameter for this requirement). In this case, a special value indicating a severe violation is returned as the objective value (value -1), see equation (2). The second violation is less severe, as the extension over the bound is not critical. At this point an objective value indicating the closeness of the maximal value to the defined boundary is calculated. This two-level concept allows a differentiation in quality between multiple output signals violating the bounds defined. The direct calculation of the objective value is provided in equation (2).

$$signal_{\max} = \max(y(t)) \qquad ObjVal_{\max} = \begin{cases} -1 & signal_{\max} \geq y_{\max} + y_c \\ \left(\frac{signal_{\max}}{y_{\max}}\right)^6 & signal_{\max} < y_{\max} + y_c \end{cases} \tag{2}$$

Example ACC

Equation (3) shows the objective function for the test objective defined in Section 2.2.

A similar assessment is used for calculating the objective value of the overshoot of an output signal after a step in the respective reference signal. First, the maximal overshoot value is calculated. Next, the relative height of the overshoot is assessed. A severe overshoot outside the specification returns a special value (again –1). This special value is used to terminate the current optimization. The test was successful, as we were able to find a violation of the specification and thus reach the aim of the optimization. In all other cases, an objective value equivalent to the value of the overshoot is calculated (similar to equation (2)).

$$d\_act_{\min} = \min(d\_act(t)) \qquad ObjVal = \begin{cases} -1 & d\_act_{\min} \leq d\_des - 10 \\ \left(\frac{-d\_act_{\min}}{-10}\right)^6 & d\_act_{\min} > d\_des - 10 \end{cases} \tag{3}$$

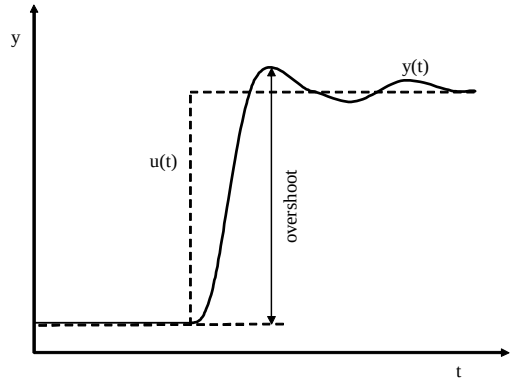

Figure 10. Assessment of signal overshoot

Each of the requirements tested produces one objective value. For nearly every realistic system test we receive multiple objective values. In order to assess the quality of all objectives tested we employ multi-objective ranking as supported by the GEATbx [Poh05]. This includes Pareto-ranking, goal attainment, fitness sharing and an archive of previously found solutions.

Example ACC

With an active ACC, the car can be accelerated only by pushing the control lever upwards (the respective input value is 1). The car is decelerated by pushing the control lever downwards (input value: 2). Besides the amplitude of the input control lever, the relative length of the signal sections could be changed between 1 and10. The results of a successful optimization are shown in Figures 11 and 12.

The optimization process is visualized in Figure 11. The left graph presents the progress of the best objective value over all the generations. The optimization continually finds better values. In the 83rd generation a serious violation is detected and an objective value of –1 returned. The middle graph presents the variables of the best individual in a color quilt. Each row represents the value of one variable over the optimization. The graph on the right visualizes the objective values of all the individuals during the optimization (generations 1 to 82).

The graphs in Figure 12 provide a much better insight into the quality of the results, visualizing the problem-specific results of the best individual of each respective generation. The input of the control lever is shown in the two top most graphs. The resulting velocity of the car is presented below. The graphs are taken from the 3rd (left) and 83rd generation (right). At the beginning the actual distance d_{act} (bottom graph left) is far above the critical boundary. During the optimization, the velocity is increased (the control lever is pushed up more often and at an earlier stage as

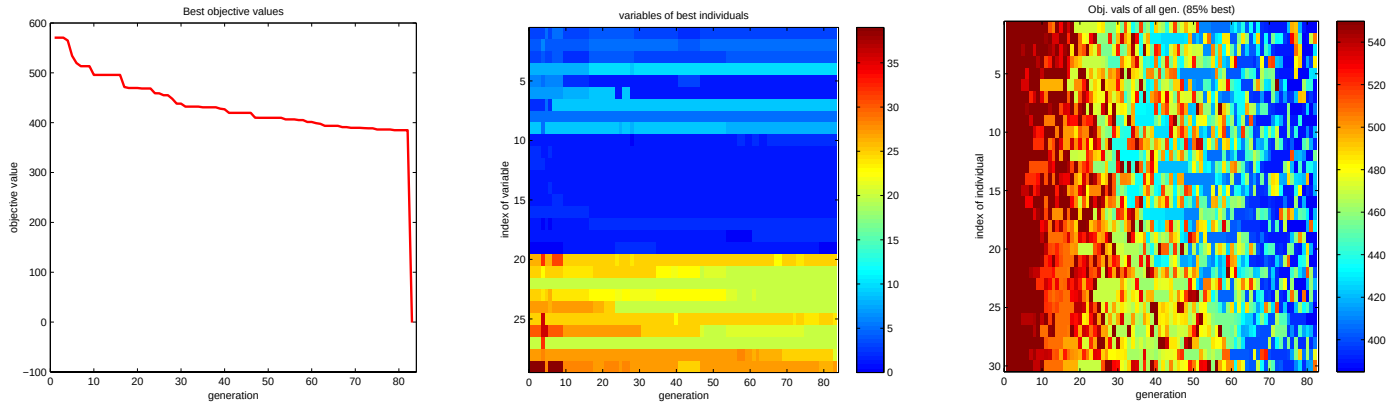

Figure 11. Visualization of the optimization process, left: best objective value; middle: variables of the best individual; right: objective value of all individuals

well as being pushed down less frequently). Additionally, the velocity of the target car is low (nearly the whole time at the defined lower boundary) during the whole scenario (3rd graph from the top). At the end an input situation is found, in which the actual distance d_{act} is smaller than the bound specified (bottom graph right). By looking at the respective input signals the developer can check and change the implementation of the system.

During optimization we employed the following evolutionary parameters: 20 individuals in 1 population, discrete recombination and real valued mutation with medium sized mutation steps, linear ranking with a selection pressure of 1.7 as well as a generation gap of 0.9. Other tests with a higher number of relevant input signals and thus more optimization variables employ 4-10 subpopulations with 20-50 individuals each. In this case, we use migration and competition between subpopulations. Each subpopulation uses a different strategy by employing different parameters (most of the time differently sized mutation steps).

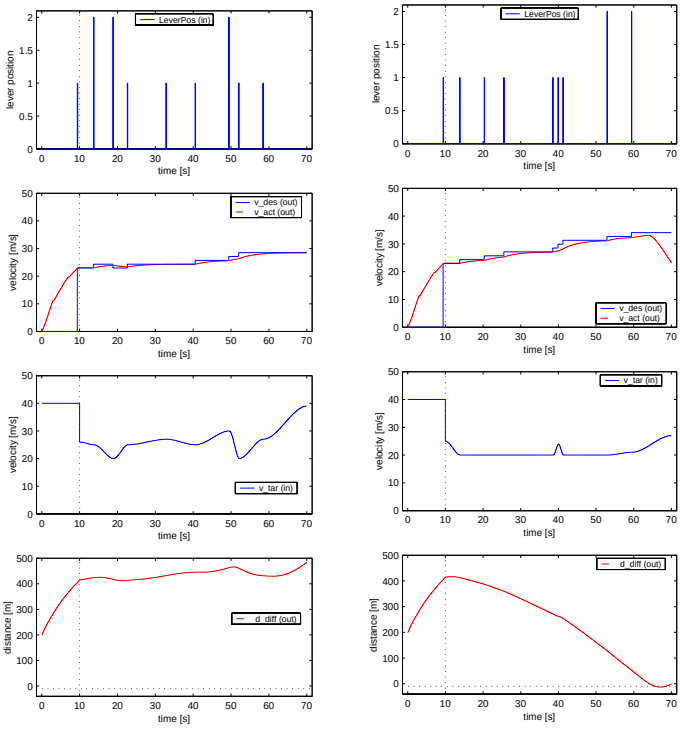

Figure 12. Problem-specific visualization of the best individual during the optimization, left: 3rd generation, right: 83rd generation; from top to bottom: input of the control lever, vehicle velocity (desired and actual), velocity of target, distance

2.5 ASSESSMENT OF COUNTER EXAMPLES

In the context of an automatic safety test it is of central importance that the test sequences which were generated automatically (counter examples) undergo an intensive analysis (assessment) carried out by human experts. Therefore, it is useful to provide a graphical notation for the generated input sequences which caters for the human tester [Con01]. Moreover, for the seamless integration of the test sequences resulting from EST with test sequences generated by other test design techniques, it is desirable to have input sequences depicted in the same way as other functional test scenarios.

For this purpose, one possibility would be to apply the extended classification-tree notation, which is used in the context of the Classification-tree Method for Embedded Systems (CTM_{EMB}) [Con04], [Con04a].

The CTM_{EMB} notation allows a comprehensive graphical description of time-dependent test sequences by means of abstract signal waveforms that are defined stepwise for each input.

The classifications of the tree represent the input variables of the functional model. Its classes are obtained by dividing the range of each variable into a set of non-overlapping values or intervals. Based on the input variable partitions, the test scenarios describe the course of these inputs over time in a comprehensive, abstract manner.

Time dependent behavior, 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.

Example ACC

Figure 13 shows an example of the visualization of the counter example given in Figure 12 by means of the CTM_{EMB} notation. The manual assessment shows that the input sequence which leads to the violation of the given safety requirement could occur in reality too. This means the algorithms or parameters of the ACC function must be modified and the evolutionary safety test has to be repeated.

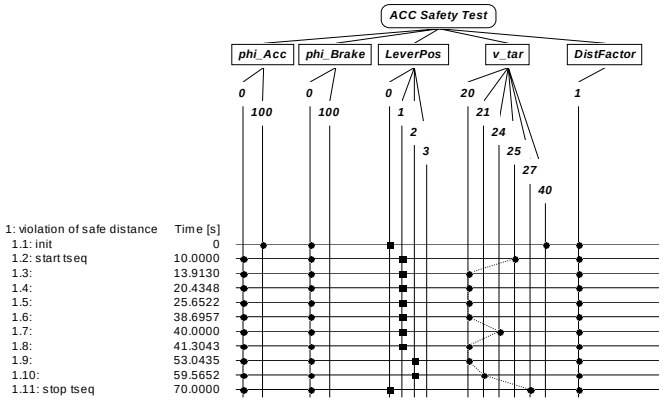

Figure 13. Visualization of a counter example generated by the EST

3 MODEL-BASED TEST STRATEGY

A singular testing technique does not generally lead to sufficient test coverage. Therefore, in practice, the aim is for complimentary test design techniques to be combined in the most adequate way to form a test strategy. The aim of an effective test strategy is to guarantee a high probability of error detection by combining appropriate testing techniques. An effective model-based test strategy must adequately take into account the specifics of model-based development and especially the existence of an executable model.

A model-based test strategy which integrates evolutionary safety testing is outlined in the following. The systematic functional model test based on the functional specification, the interfaces, and the executable model forms the focal point of such a model-based test strategy. In addition, an adequate structural test criterion is defined on model level, with the help of which the coverage of the tests thus determined can be evaluated and the definition of complementary tests can be controlled [Con04a]. If sufficient test coverage has thus been achieved on model level, the model should be mature enough to start safety testing.

Based on the safety requirements, the EST approach should be applied to every given safety requirement in order to aggressively try to generate counter examples.

If the EST process leads to one or more counter examples, those examples have to be assessed carefully. If the counter examples are impossible in practice, the search space description has to be adapted in order to avoid those scenarios in future. If scenarios described by the counter examples could happen in practice, the model or its parameter has to be corrected. After that, the model test has to be repeated: all existing model tests have to be repeated. By doing this, the tester has to ensure that the structural coverage is still sufficient and that all the tests lead to valid system reactions. Finally, the EST process has to be repeated. If no new counter examples are generated, the code generation can be started. All model test sequences can be reused for testing the control software generated from the model and the control unit within the framework of back-to-back tests. In this way, the functional equivalence between the executable model and derived forms of representation can be verified ([BCS03], [CSW04]).

The proposed model-based test strategy can be tool supported by integrating the different tools for creating test sequences into the model-based testing environment MTest ([LBE+04], [MTest]).

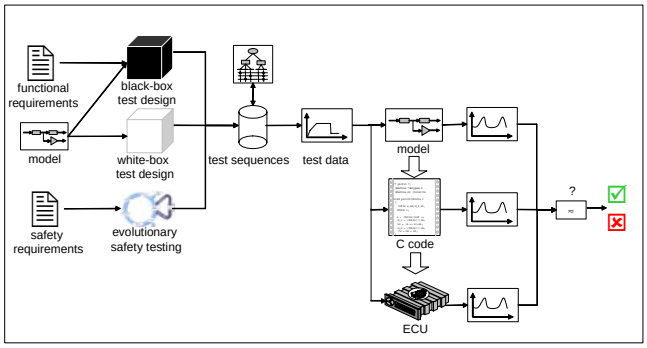

Figure 14. Model-based test strategy

4 RELATED WORK

Within the framework of model-based development, the derivation of safety requirements on model level can be supported by special model-based hazard/safety analysis techniques ([GC04], [PMS01]).

The instrumentation of the executable models with watchdogs/assertions (see e.g.[Rau99], [Reactis], [SL-VV]) or model checking of safety properties (cf. [EmbVal]) are alternative or supplementary techniques for checking the fulfillment of safety requirements.

Test design techniques which can be applied to black-box model testing are described in [HPP+03], [CFS04], [Con04], [Con04a], [HSM+04], and [LBE+04]. [HSM+04]White-box test design techniques for models are discussed in [Ran03], [Lin04], and [HSP05].

An approach for safeguarding the transformation from the tested model into code, i.e. the code generation, is described in [SC03].

CONCLUSION

In this paper we have presented the use of evolutionary sequence testing for the safety testing of embedded automotive control systems.

We have presented a small selection of the results. The results show the new test method to be promising. It was possible to find test sequences, without the need for user interaction, for problems which could previously not be solved automatically.

During the experiments a number of issues were identified which could further improve the efficiency of the test method presented. It is necessary to include as much of the existing problem-specific knowledge in the optimization process as possible.

A systematic approach for the selection of test scenarios and a notation for their appropriate description must therefore form the core elements of a safety testing approach for automotive control software. In order to allow testing to begin at an early stage, test design should build upon development artifacts available early on, such as the specification or an executable model.

ACKNOWLEDGMENTS

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), http://www.immos-project.de

REFERENCES

[BCS03] Baresel, A., Conrad, M., Sadeghipour, S.: The Interplay between Model Coverage and Code Coverage. Proc. 11. Europ. Int. Conf. on Software Testing, Analysis and Review (EuroSTAR 03), 2003.

[BPS03] Baresel, A., Pohlheim, H., Sadeghipour, S.: Structural and Functional Sequence Testing of Dynamic and State-Based Software with Evolutionary Algorithms. Proc. Genetic and Evolutionary Computation Conf. (GECCO 03), pp. 2428 – 2441, 2003.

[BSS02] Baresel, A., Sthamer, H., Schmidt, M.: Fitness Function Design to improve Evolutionary Structural Testing. Proc. Genetic and Evolutionary Computation Conf. (GECCO 02), pp. 1329-1336, 2002.

[Bei83] Beizer, B.: Software Testing Techniques. New York: Van Nostrand Reinhold, 1983.

[CFS04] Conrad, M., Fey, I., Sadeghipour, S.: Systematic Model-Based Testing of Embedded Control Software: The MB3T Approach. Proc. ICSE 2004 Workshop W14S on Software Engineering for Automotive Systems (SEAS 04), pp.17-25, 2004.

[CH98] Conrad, M., Hötzer, D.: Selective Integration of Formal Methods in the Development of Electronic Control Units. Proc. 2. IEEE Int. Conf. on Formal Engineering Methods (ICFEM 98), IEEE Computer Society, pp. 144-155, 1998.

[Con01] Conrad, M.: Beschreibung von Testszenarien für Steuergerätesoftware - Vergleichskriterien und deren Anwendung. VDI Berichte, Vol. 1646, VDI Verlag, pp. 381-398, 2001.

[Con04] Conrad, M.: A Systematic Approach to Testing Automotive Control Software. Proc. Convergence 2004, SAE paper 2004-21-0039, 2004.

[Con04a] Conrad M.: Modell-basierter Test eingebetteter Software im Automobil – Auswahl und Beschreibung von Testszenarien. PhD thesis, Deutscher Universitäts-Verlag, 2004.

[CSW04] Conrad, M., Sadeghipour, S., Wiesbrock, H.-W.: Automatic Evaluation of ECU Software Tests. Proc. SAE World Congress 2005, SAE paper 2005-01-1659, 2005.

[EmbVal] EmbeddedValidator (product information). OSC – Embedded Systems AG, www.osc-es.de/products/en/embeddedvalidator.php

[FMN+94] Fenelon, P., McDermid, J.A., Nicholson, M., Pumfrey, D.J.: Towards Integrated Safety Analysis and Design. ACM Computing Reviews, Aug. 1994, pp. 21-32, 1994.

[GHD98] Grieskamp, W., Heisel, W., and Doerr, H.: Specifying embedded systems with statecharts and Z - An agenda for cyclic software components. Proc. Formal Aspects of Software Engineering (FASE 98), LNCS 1382, Springer-Verlag, 1998.

[GC04] Grönberg, R., Conrad. M.: Werkzeugunterstützung für Sicherheitsanalysen von Hardware- und Software-Systemen. Technical Report FT3/A-2000-007, DaimlerChrysler AG, Forschung und Technologie 3, Berlin, Germany, 2000.

[HH+01] Harman, M., Hu, L., Munro, M., Zhang, X.: Side-Effect Removal Transformation. Proc. IEEE Int. Workshop on Program Comprehension (IWPC), Toronto, Canada, 2001.

[HPP+03] Hahn, G., Philipps, J., Pretschner, A., Stauner, T.: Prototype-Based Tests for Hybrid Reactive Systems. Proc. 14. IEEE Int. Workshop on Rapid System Prototyping, San Diego, US, 2003.

[HSM+04] Horstmann, M., Schnieder, E., Mäder, P., Nienaber, S., Schulz, H.-M.: A framework for interlacing Test and/with Design. Proc. ICSE 2004 Workshop W14S on Software Engineering for Automotive Systems (SEAS 04), 2004.

[HSP05] Hermes, T., Schultze, A., Predelli, O.: Software Quality is not a Coincidence - A Model-Based Test Case Generator. Proc. SAE World Congress 2005, SAE paper 2005-01-1664, 2005.

[HV98] Hohler, B., Villinger, U.: Normen und Richtlinien zur Qualitätssicherung von Steuerungssoftware. Informatik-Spektrum, 21, pp. 63-72, 1998.

[IEC60812] IEC 60812:2003: Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA)

[IEC 61025] IEC 61025:2004: Fault tree analysis (FTA), 2004.

[JSE96] Jones, B.-F., Sthamer, H., Eyres, D.: Automatic structural testing using genetic algorithms.

Software Engineering Journal, vol. 11, no. 5, pp. 299–306, 1996.

[KCF+04] Klein, T., Conrad, M., Fey, I., Grochtmann, M.: Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler. Lecture Notes in Informatics (LNI), Vol. P-45, Köllen Verlag, pp. 31-41, 2004.

[Kor90] Korel, B.: Automated Test Data Generation. IEEE Transactions on Software Engineering, Vol. 16 no. 8, pp.870-879, 1990.

[LBE+04] Lamberg, K., Beine, M., Eschmann, M., Otterbach, R., Conrad, M., Fey, I.: Model-based Testing of Embedded Automotive Software using MTest. Proc. SAE World Congress 2004, SAE paper 2004-01-1593, 2004.

[Lin04] Linder, P.: Modellbasiertes Testen von eingebetteter Software - Ein Ansatz auf der Grundlage von Signalflussplänen. Proc. Automotive Safety & Security 2004, Stuttgart, Germany, 2004.

[MK99] Mackenthun, R., Kelling, C.: An Agenda for the Safety Analysis of Cyclic Software Components. In: ESPRESS: Final Reports, Berlin, Germany, June 1999.

[MTest] MTest (product information), dSPACE GmbH, http://www.dspaceinc.com

[PMS01] Papadopoulos, Y., McDermid J., Mavrides, A., Scheidler S., Maruhn, M.: Model-Based Semiautomatic Safety Analysis Of Programmable Systems In Automotive Applications. Proc. Int. Conf. on Advanced Driver Assistance Systems (ADAS 01), 2001.

[Poh99] Pohlheim, H.: Evolutionäre Algorithmen - Verfahren, Operatoren, Hinweise aus der Praxis. Berlin, Heidelberg: Springer-Verlag, 1999. http://www.pohlheim.com/eavoh/index.html

[Poh05] Pohlheim, H.: GEATbx - Genetic and Evolutionary Algorithm Toolbox for Matlab. http://www.geatbx.com/, 1994-2005.

[Ran03] Ranville, S.: MCDC Unit Test Vectors From Matlab Models – Automatically. Proc. Embedded Systems Conference, 2003.

[Rau99] Rau, A.: Verwendung von Zusicherungen in einem modellbasierten Entwicklungsprozess. It+ti 3/2002, pp. 137-144, 2002.

[Rau03] Rau, A.: Model-based Development of Embedded Automotive Control Systems. PhD thesis, http://dissertation.de, 2003.

[Reactis] Reactis Tester / Validator (product information). Reactive Systems Inc., http://www.reactive-systems.com

[SC03] Stürmer, I. and Conrad, M.: Test Suite Design for Code Generation Tools. 18. IEEE Int. Conf. on Automated Software Engineering (ASE 03), 2003.

[SLSF] Simulink/Stateflow (product information). The MathWorks Inc., http://www.mathworks.com/products.

[SL-VV] Simulink Verification and Validation Toolbox (product information). The MathWorks Inc., www.mathworks.com/products/simverification

[Sth96] Sthamer, H.: The Automatic Generation of Software Test Data Using Genetic Algorithms. PhD Thesis, University of Glamorgan, Pontyprid, UK, 1996.

[TCC+98] Tracey, N., Clark, J., Mander, K., McDermid, J.: An Automated Framework for Structural Test-Data Generation. Proc. 13. IEEE Conf. on Automated Software Engineering, 1998.

[VDI/VDE 3550] Computational Intelligence, Evolutionary algorithms - Terms and definitions. VDI/VDE guideline VDI/VDE 3550, Part 3, Verein Deutscher Ingenieure, 2003.

[Weg01] Wegener, J.: Evolutionärer Test des Zeitverhaltens von Realzeit-Systemen. PhD thesis, Shaker-Verlag, 2001.

[WSB01] Wegener, J., Sthamer, H., Baresel, A.: Evolutionary Test Environment for Automatic Structural Testing. Special Issue of Information and Software Technology, vol. 43, pp. 851–854, 2001.

[WSJ+97] Wegener, J., Sthamer, H., Jones, B., Eyres, D.: Testing Real-time Systems using Genetic Algorithms. Software Quality Journal, vol. 6, no. 2, pp. 127–135, 1997.

CONTACT

Hartmut Pohlheim received a Diploma in Systems Engineering in 1993 from the University of Technology Ilmenau (Germany). In 1998 he earned his PhD at the University of Technology Ilmenau with a dissertation titled “Development and Engineering Application of Evolutionary Algorithms”. Since 1995 he has been a research scientist at the Software Technology Lab of DaimlerChrysler Research & Technology in Berlin.

In 1995 Mirko Conrad earned a Diploma degree in Computer Science from the Technical University Berlin (Germany). In 2004 he received his PhD from the TU Berlin for his work on model-based testing of embedded automotive software. Since 1995 he has been a project manager and research scientist at the Software Technology Lab of DaimlerChrysler Research & Technology. He is member of the Special Interest Group for Testing, Analysis and Verification of Software in the German Computer Society (GI TAV) and the MathWorks Automotive Advisory Board (MAAB).

Arne Griep received a Diploma in Systems Engineering in 2002 from the University of Technology Ilmenau (Germany). Since 2003 he has been a software developer at IAV GmbH (Germany).