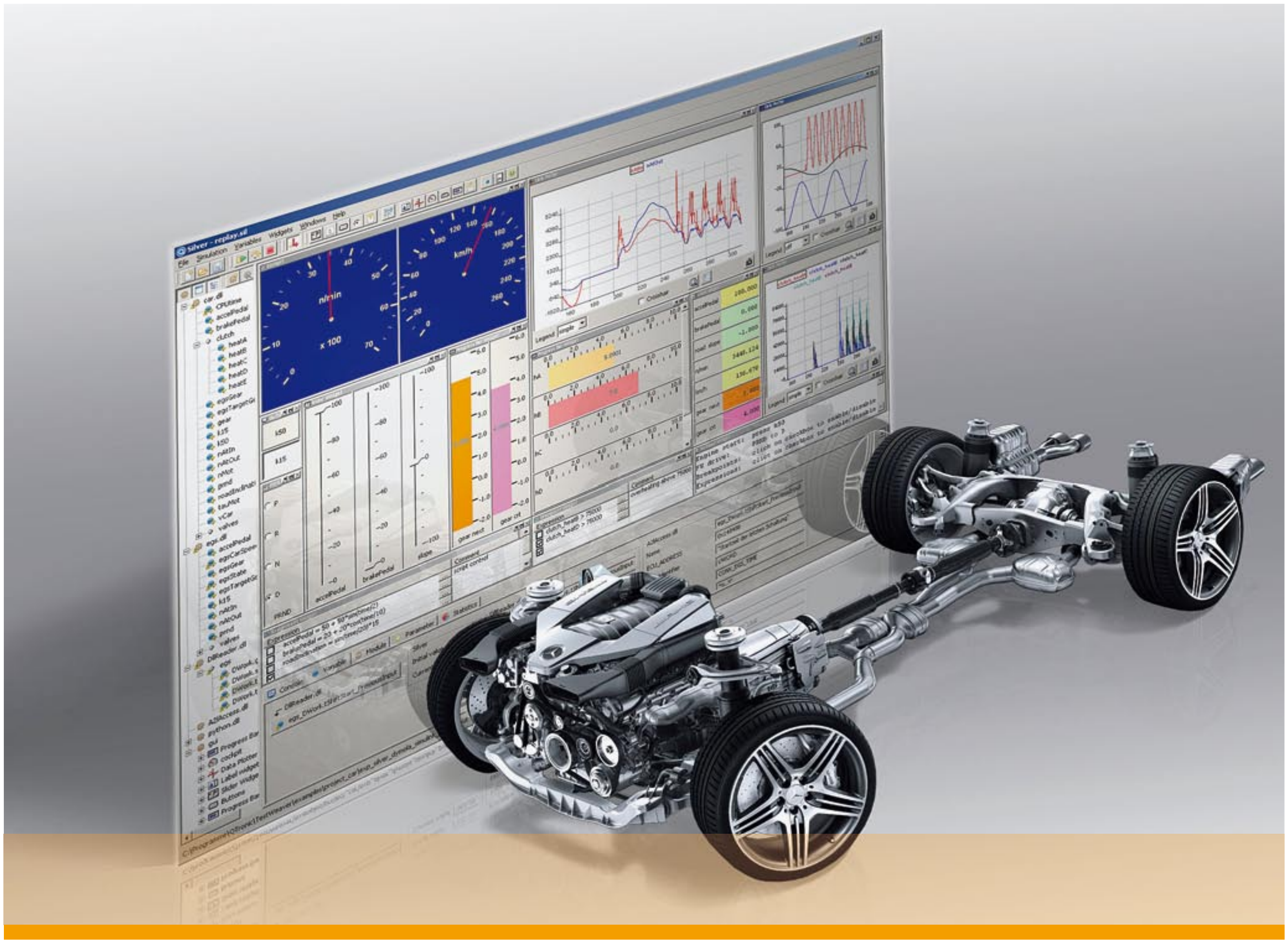

Automating Test of Control Software

Method for Automatic Test Generation

Daimler uses a model-based test process for the development of software modules for automatic transmissions. The validation of the module functions is carried out with a test generator developed by QTronic. This test generator is able to automatically generate, simulate and analyze thousands of test scenarios. The individual components of the development environment, such as, powertrain and vehicle models, software modules and test generator, are coupled by a Software-in-the-Loop (SiL) tool and can be executed on standard laptops. The software modules are operating as in a real vehicle and can be thoroughly tested.

1 Introduction

The complexity of vehicle engine and transmission systems is steadily increasing. One of the reasons is the increasing market expectation with respect to the engine and transmission efficiency, agility, driving pleasure and emissions. This multitude of market demands can only be achieved by combining robust mechanics with intelligent software. An example for this development is the MCT 7-speed sports transmission, where a compact, wet start-up clutch replaced the conventional torque converter [1]. For controlling the start-up clutch new software modules had to be developed and integrated into the existing software of the 7G-Tronic transmission. In addition to the already existing test procedures, such as Hardware-in-the-Loop (HiL) tests, test bench investigations, and road trials with physical prototypes, a new method based on automatic test generation was used for the test and validation of the software modules. This new method allowed for significantly improved test coverage and testing efficiency. The goal of this automation is a comprehensive system test, maximizing the relevant system states that are reached and tested.

2 The Environment for Model-based Software Test

The method explained here is used for testing complex software modules. Such modules can be tested only in closed-loop interaction with the controlled physical subsystem due to the feedback interaction of the software functions with the dynamics of the physical subsystem. For this purpose we use a testing environment that allows coupling the vehicle simulation with the software modules that are to be tested. As a test generator we use TestWeaver, a tool developed by QTronic. During the design process the test method is repeatedly used for finding faults and weaknesses of the software functions as early as possible.

2.1 The Test Method Workflow

As a first step, the test environment has to be setup. The plant simulation model has to be developed and the tester has to be configured. For this, one needs to identify which variables should be controlled by the test generator during the test (such as, acceleration pedal, braking pedal, road inclination, etc.). One also needs to define the criteria that are used to assess the system behaviour as good or bad. Furthermore, project-specific templates used for reporting the test outcome are created. The software version under test is compiled for the SiL target (DLL) and the co-simulation tool is configured. Now, the test generator can be started. The test tool automatically generates, simulates and evaluates thousands of differing test scenarios in a reactive way and reports the states and the problems reached by the system during the test. Afterwards, the test results are revised by a development engineer. After fixing the found problems the test is repeated in order to ensure the success of the correction. The configuration of the test generator can be reused for subsequent tests and software versions. This allows an iterative approach that continuously improves the quality of the software functions. **Figure 1** shows a summary of these steps.

2.2 The Plant Model

A central piece of the test environment is the simulation model of the transmission, **Figure 2**. The level of detail and the

The Authors

Dr.-Ing. Anton Rink
is Manager of Transmission Logic at Daimler AG in Stuttgart (Germany).

Dipl.-Ing. Emmanuel Chrisofakis
is Simulation Engineer at Daimler AG in Stuttgart (Germany).

Dr. Mugur Tatar
is Managing Director of QTronic GmbH in Berlin (Germany).

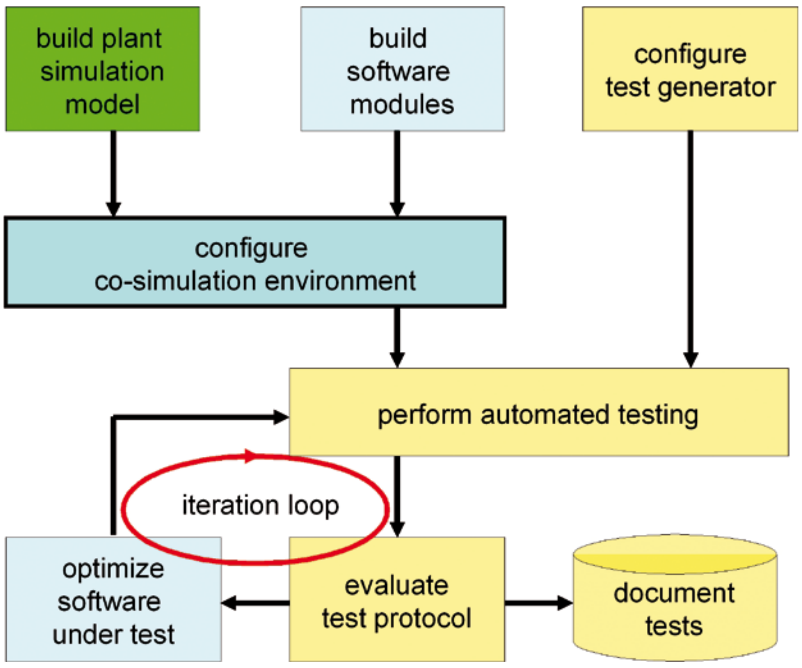
build plant simulation model
build software modules
configure test generator
configure co-simulation environment
perform automated testing
iteration loop
optimize software under test
evaluate test protocol
document tests

Figure 1: The test method workflow

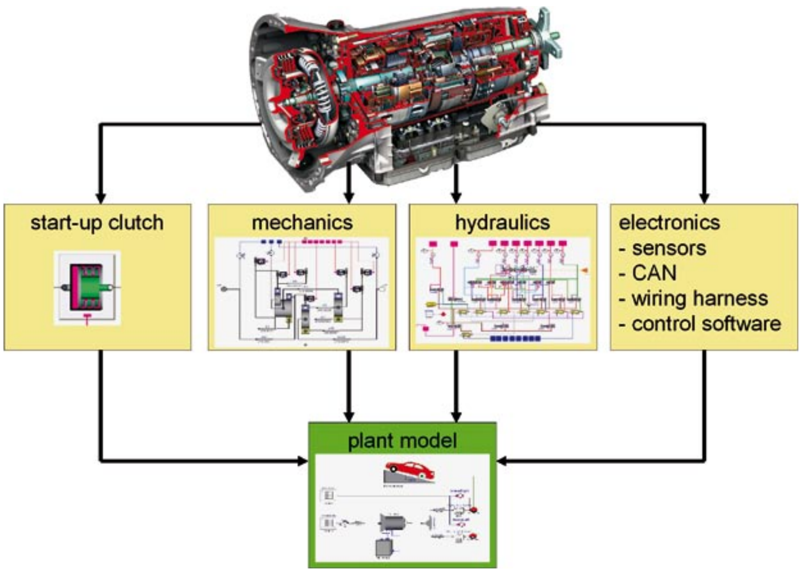

Figure 2: The plant model used within the test environment

fidelity of this model determines which software functions of the transmission control can be tested with which quality expectations. The complex dynamic behaviour of the transmission is described with the object-oriented modelling language Modelica [4]. The model includes planetary gear trains, shafts, clutches and brakes, the hydraulic module, oil supply, and the electronic module. A special attention has been given to the models of the components that play a major role in the control of the transmission: the start-up clutch, internal clutches and brakes and the hydraulic control. Possible component faults, selected based on a risk analysis, are modelled as well, and can be activated and deactivated during the simulation. Further components of the model are: engine, cardan shaft, differential, braking system, street-wheel contact and car body. The model is calibrated to within 10 % deviation from test bench and vehicle measurements for both static and dynamic effects. The result is a model for the longitudinal dynamics of a vehicle drive that allows the simulation of all driving and fault scenarios that are relevant for the transmission.

2.3 The Software under Test

Those software modules that require testing are integrated into the test environment as a DLL. Depending on the development phase, different description forms for the software structure, behaviour, variables and parameters can be used. In the application example presented here a SiL test is described, i.e. the original C-code of the transmission is integrated without modification in the test environment. In order to facilitate the communication between the software controller and the simulation model some low-level functions of the transmission control unit have been emulated. The resulting software DLL is executed with the same cycle rate as in the real transmission control unit.

2.4 The Simulation Environment

The plant model and the software modules of the transmission are executed cyclically by co-simulation, e.g. every 10 ms. The modules exchange computed signal values among themselves at each cycle. This way, the interaction of the software and the vehicle hardware can 'virtually' be recreated and tested on a PC. **Figure 3** shows the structure of the entire test environment, consisting of: plant model, software, connection to calibration tool via XCP, graphical user interface to control the simulation or, alternatively, connection to test automation. The tools that are used in vehicle for measurement and calibration can be used in the SiL environment as well because the ASAM standards XCP and A2L are supported by our simulation environment. For analyzing in detail the C-code execution, a source-code debugger can be attached to the simulation process. Also tools measuring the code-coverage can easily be integrated in the environment.

2.5 Test Generation and Automated Evaluation

The development tool TestWeaver [2, 3] facilitates automatic test generation and evaluation. The test generator autonomously generates thousands of differing control sequences during a test, executes them via SiL and evaluates the system response, **Figure 4**. The test scenarios are not randomly created, but based on an intelligent strategy that heuristically follows two goals. The first goal is to maximize the coverage of the relevant system states. The test generator attempts to find at least one test scenario for each relevant system state and then systematically investigates these states with all (or many) differing input combinations. Which system states TestWeaver should reach can be specified by the test engineer easily in the test configuration. The second goal is to find many "bad" or even

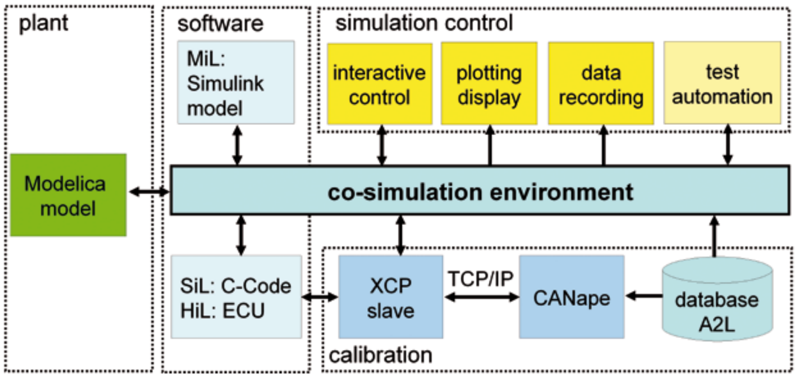

Figure 3: Co-simulation environment and interactive interface with display

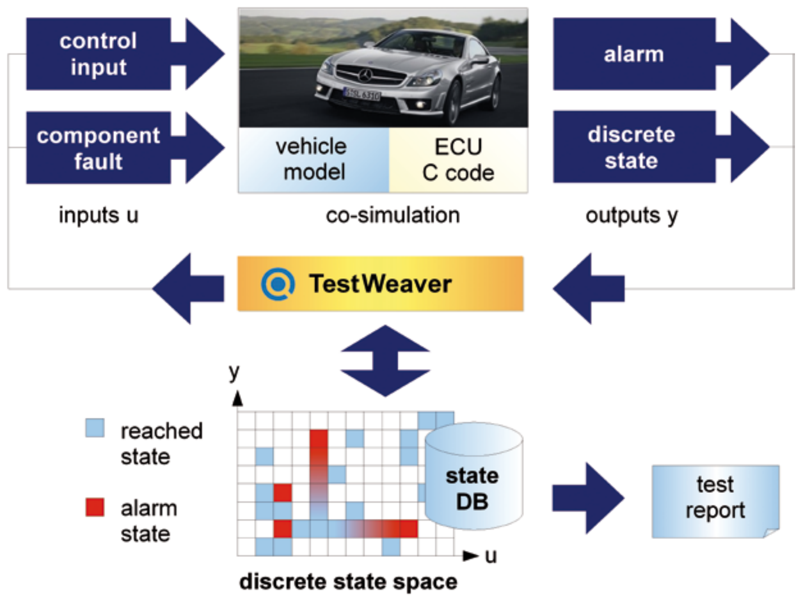

Figure 4: The test generator operation method

"critical" system states. TestWeaver actively attempts to "construct" test cases that worsen the quality of the system behaviour. The evaluation criteria indicating good or bad quality are defined when configuring the test generator.

3 Application Example: the MCT 7-Speed Sports Transmission

The transmission control software modules of the MCT 7-speed sports transmission [1], where the hydrodynamic torque converter was replaced by a freely controllable hydraulic clutch, were developed with this test method. The original C-code and the original calibration parameters of the control software were subject to the tests. The internal adaptation algorithms of the transmission control software were run by a script in the simulation environment to allow the software to self-tune a large set of adaptable parameters prior to the tests. This guarantees an ideal fitting of the software with the transmission simulation. Only then is it possible to use sophisticated quality criteria such as shift quality or stress on components during the test evaluation. For checking the on-board monitoring and diagnosis functions, the plant model contained freely controllable hydraulic and sensor faults. The test generator was configured to control the following signals: starter, acceleration pedal, braking pedal, gear lever (P-R-N-D), drive program selector, road inclination, oil temperature and component faults. A typical test contained, for instance, 1100 drive manoeuvres of 50 seconds duration. During the resulting 15 hours of simulation, 235.000 different system states where reached and classified. Each manoeuvre was automatically analyzed for problems in the C-code (e.g. division by 0) and in the control logic (e.g. wrong fault reaction). After corrective measures, a renewed test was started to ensure their positive effect. Due to the systematic approach of the test generator, new driving manoeuvres were simulated that were not contained in previous test specifications. The high degree of automation of the software test and the high level of reuse of the test configurations lead to a significant test coverage increase combined with a significant reduction in the effort spent by the development engineers when compared to manual testing methods.

4 Summary

The increasing complexity of software modules requires an ever-growing effort for the software test. The presented method uses a test automaton which generates thousands of driving maneuvers autonomously, executes and evaluates them using simulation. A fundamental role for the test automation is played by the plant model for the transmission and vehicle – which simulates the physical systems with the necessary fidelity. Due to the high degree of automation, the test effort spent by the development engineers is significantly reduced, while, at the same time, the test coverage is significantly increased. The test process can further be accelerated by running the tests on several PCs in parallel.

References

[1] G. Korherr, C. Dörr, A. Rink, R. Wörner: Neues Automatikgetriebe im PKW Hochleistungssegment, in: VDI Bericht 2029 zum VDI-Kongress Getriebe in Fahrzeugen, 2008
[2] A. Junghanns, J. Mauss, M. Tatar: Testautomatisierung nach dem Schachspielerprinzip. In: C. Gühmann (Hrsg.): Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik, Expert Verlag Renningen, pp. 320 - 331, 2008
[3] A. Junghanns, J. Mauss, M. Tatar: TestWeaver – A Tool for Simulation-based Test of Mechatronic Designs – In: Proceedings of the 6th International Modelica Conference, Bielefeld, 3.-4.3.2008
[4] Homepage der Modelica Association, siehe http://www.modelica.org