Search-Based Testing with in-the-loop Systems

Dr. Joachim Wegener, Peter M. Kruse
Berner & Mattner Systemtechnik GmbH, Berlin, Germany
{joachim.wegener, peter.kruse}@berner-mattner.com

Abstract
With the continuously growing software and system complexity in electronic control units and shortening release cycles, the need for efficient testing grows. In order to perform testing of electronic control units in practice hardware-in-the-loop test environments are used to run the system under test in a simulation environment under real-time conditions. Tests are usually implemented manually. Even though a lot of academic work has shown the potential of evolutionary testing to fully automate testing for different testing objectives, it is not used in industrial practice. In this work we develop an integration of evolutionary testing with a testing platform supporting model-in-the-loop-, software-in-the-loop- and hardware-in-the-loop-testing of embedded systems. We demonstrate the use of evolutionary testing for functional testing in an industrial setting by applying the developed solution to the testing of an antilock-braking-system electronic control unit.

Keywords: Testing infrastructure, Evolutionary Testing, Hardware-in-the-loop-testing, Antilock-braking-system, Functional Testing

1 Motivation

With the continuously growing software and system complexity in electronic control units and shortening release cycles, the need for efficient testing grows. Test-automation is a must to reduce expensive human efforts. Therefore, a lot of testing tools are on the market specialised in test automation. The main emphasis of the tools is the automation of test execution, monitoring, and test documentation. Seldom, support for test case design is offered.
System testing of electronic control units is usually black-box-testing. Test cases are manually derived from the specification. Various works propose the use of evolutionary testing for the automation of test case design. Most common is the automation of structural test case design, but even for the automation of structural testing evolutionary testing [1] is not widely used in industrial practice due to the missing tool support. In practice, even more important is functional testing validating the system under test against its specification. Only few works have tried to apply evolutionary testing for functional testing, e.g. [2], [15]. Functional testing using search-based techniques is not widely common, because efforts for the development of the fitness functions and for setting up appropriate test environments is high.
Buehler and Wegener [2] showed that driver assistance systems are test objects which could highly benefit from evolutionary testing. However, the tests performed were executed in a software-in-the-loop (SiL) test environment not taking the real electronic control unit into account.
In this work we develop a testing solution which supports search-based functional testing for model-in-the-loop (MiL) testing, software-in-the-loop (SiL) testing and hardware-in-the-loop (HiL) testing in a uniform way. The testing framework developed has been evaluated for testing the functioning of a serial production antilock-braking-system (ABS).

2 Test Automation Framework for Evolutionary Functional Testing

Electronic control units are used in nearly all industrial areas to control complex systems like airplanes, cars, trains, engines etc. Usually, testing of such systems contains unit testing, integration testing and system testing. When model-based development is in place also model testing is performed. Common test platforms for the testing are model-in-the-loop testing (testing the model or parts of the model in a simulation environment), software-in-the-loop testing (testing the resulting software in a simulation environment) and hardware-in-the-loop testing (testing the software integrated on the electronic control unit in a real-time simulation environment). Sometimes also processor-in-the-loop testing is performed (testing the resulting software cross compiled on an evaluation board with the target processor in a simulation environment).
In order to provide a testing framework that supports search-based testing for different testing phases and on different testing platforms we integrate different hardware and software components (Fig. 1).

modularHiL
modularHiL [3] is a modular, universal hardware-in-the-loop test system developed by Berner & Mattner. Hardware-in-the-loop testing systems are necessary to test electronic control units in a simulation environment in real-time. In order to test electronic control units their interfaces are connected to the hardware-in-the-loop test system providing the corresponding input signals and reading the output signals. Usually, a simulation environment is

necessary to run the tests and to simulate the real application environment of the electronic control unit.
As an individual module, with flexible signal conditioning and the use of open industry standards, modularHiL could be used for testing a single embedded control unit. Using an optical networking technology it is possible to link several modularHiL systems together to form a powerful test environment for integration testing. During integration testing the interplay of several electronic control units is tested. Because it uses the same modules for component and integration test systems, modularHiL offers customers cost benefits and high flexibility.

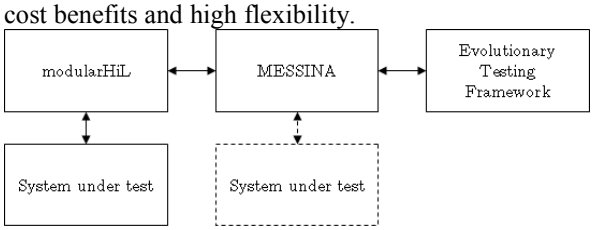

Figure 1: Components of the Test Automation Framework for Search-Based Testing (for testing in a MiL/SiL scenario no hardware-in-the-loop test system is needed, tests are directly executed by MESSINA)

MESSINA
MESSINA [4] is a testing infrastructure allowing the implementation of hardware and software independent test sequences in different notations such as UML, Java, or TPT [5]. Using two abstraction layers it offers execution on different platforms. The first layer is the signal pool containing all system signals provided by the connected devices or software devices. The signal pool allows easy read- and write-access to all the signals used by the system under test or the simulation environment. The second layer is formed by abstract in-the-loop systems, which can be MiL, SiL and HiL systems.
Three stages of testing are necessary and common. Standards like ISO 26262 [12] require even more test stages. The model has to be tested to ensure a good specification quality. If code generation is used the model represents the implementation as well and will be tested thoroughly. Software testing is performed to verify that the software implementation fulfils the specification. If code generation is performed, software testing ensures the correct transformation performed by the code generator from model to software. Integration testing of hardware and software is always performed to verify if the integrated system works correctly.
For MiL and SiL testing MESSINA supports software devices like MATLAB/Simulink models, ASCET models and AUTOSAR software components (Fig. 1 system under test with dotted lines). Multiple models and software components can be run in parallel to perform virtual system integration.
For HiL testing MESSINA is directly connected with modularHiL. It is possible to download tests implemented with MESSINA to the modularHiL where the tests are executed in real-time (Fig. 1 system under test).
As a result tests implemented in MESSINA can be used seamlessly in the model test (MiL), software test (SiL) and hardware test (HiL). Therefore, MESSINA could be used for thorough model-based electronic control unit (ECU) testing from specification to HiL testing. Since tests are defined hardware independent MESSINA ensures a high portability of the tests between different test environments. The only difference between HiL and SiL from MESSINA point of view is the usage of a different environment model (e.g. for MiL and SiL using a software ABS component, for HiL using a real hardware ABS component). Therefore test cases can be used without any further adaption for MiL, SiL, and HiL testing.
Recurrent test steps can be defined as templates in a test library and be reused for defining test sequences. Test variants can easily be created by parameterized generic test cases.

Evolutionary Testing Framework
The EvoTest [6] project implements an extensible and open Automated Evolutionary Testing Architecture and Framework that provides general components and interfaces to facilitate the automatic generation, execution, monitoring and evaluation of test cases using evolutionary computation. The evolutionary testing framework creates an evolutionary algorithm suitable for the system under test using GUIDE [8, 9]. For this, a problem description has to be provided to the framework, which is analysed by GUIDE to generate an evolutionary algorithm best suited to optimize the testing problem automatically. During the optimisation process the evolutionary algorithm generated provides the individuals representing test data for the system under test and expects the fitness values for each individual back. On basis of the fitness values the next generation of individuals is generated. Details of the evolutionary testing framework are described in [7].

Integration of Test Automation Framework
The integration of the EvoTest evolutionary testing framework with MESSINA has been implemented for the automatic generation of test cases. For this, MESSINA has been extended by a PlugIn for configuring and running the evolutionary test.
The PlugIn implements the communication with the evolutionary testing framework, controls the test execution for the generated tests and returns the fitness values calculated for the individuals on basis of the test execution results back to the evolutionary testing framework.
The signals of the system under test held in the signal pool of MESSINA containing type information, value range information etc. are passed to the evolutionary testing framework as individual descriptions. The evolutionary testing framework then creates a specific evolutionary algorithm exactly fitting to this

description using GUIDE [8, 9]. The use of mixed data types is supported by MESSINA and the evolutionary testing framework. This allows in many cases a one-to-one transformation of individuals into test data. The individuals created by the evolutionary testing are executed with the system under test by MESSINA. Generic test cases are parameterized with the individual data.
For MiL and SiL tests MESSINA calls the system under test directly with the parameterized test cases, for HiL testing MESSINA downloads the tests to modularHiL allowing real-time execution of the tests on the electronic control unit.
For the fitness function calculation the behaviour of the system under test has to be analyzed for each test case executed. MESSINA allows recording the behaviour of the system under test for the generated test data sets through monitoring interfaces. The fitness function calculation is implemented in the generic test case, and the fitness function values passed back to the evolutionary testing framework.
The MESSINA run-time system allows a comfortable execution and remote debugging of test cases directly on the target system.
With the implemented solution search based testing could be used for thorough model-based ECU testing from specification to HiL testing. Since tests are defined hardware independent MESSINA ensures a high portability of the tests between the different test platforms. The only difference between HiL, MiL and SiL testing is the usage of different environment models.

4 Evaluations

To evaluate the implemented test automation framework an evolutionary functional test for an anti-lock braking system has been realized. The anti-lock braking system used for the tests is already in serial production.

Anti-lock braking system (ABS)

An anti-lock braking system is a system which prevents the wheels of a vehicle from locking while braking, in order to allow the driver to maintain steering control under heavy braking and, in most situations, to shorten braking distances. ABS is very effective at braking in adverse weather conditions like ice, snow or rain. When ABS equipped brakes are depressed hard - like in an emergency braking situation - the ABS system pumps the brakes several times per second. Sensors measure the speed at which the wheels are turning. If the speed decreases rapidly, the electronic control system reports blocking danger. The pressure of the brake hydraulics is reduced immediately and then raised to just under the blocking threshold. This process can be repeated several times per second. The goal of the anti-locking control system is to maintain the slip of the wheels at a level which guarantees highest braking power and highest steer ability of the vehicle.

Evolutionary Testing of the ABS

For testing the anti-lock braking system a hardware-in-the-loop test environment has been set up using the test automation framework described including modularHiL. In addition, a proper simulation environment for the ABS system has to be created to simulate the system environment of the ABS system in such a manner that the anti-lock braking system detects no difference to its use in a vehicle. In case the anti-lock braking system detects improper behaviour of the system environment it automatically changes to a failure mode making realistic testing impossible. Therefore, a complex simulation environment was implemented integrating commercial brake models, vehicle dynamics models and wheel speed sensor models from Tesis [10]. Fig. 2 shows the components of the simulation environment implemented.

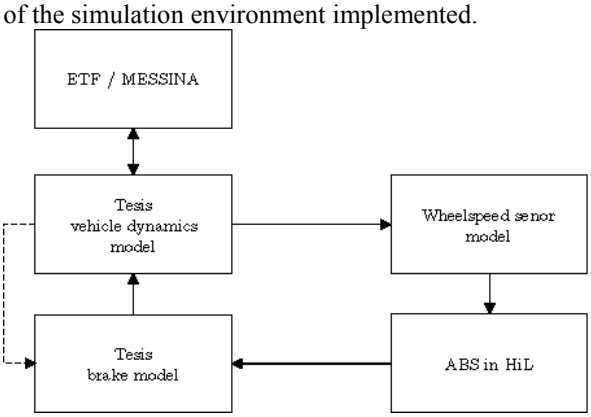

Figure 2: For testing the ABS system complex environment simulations are needed to simulate the wheelspeed sensor inputs for the ABS, the behaviour of the brakes and the vehicle dynamics

Test Results

In our first experiments the main focus lies on the evaluation of the developed test environment. The goal is to show its functioning for the fully automatic functional testing of complex electronic control units. Future experiments will deal with a thorough testing of the ABS functionality itself. Therefore, we use a simplified fitness function for the testing of the ABS system. The fitness of an individual – representing a braking manoeuvre – is calculated on the basis of the resulting braking distance of the vehicle. The fitness function maps the braking distance directly to the fitness value. Longer breaking distances result in higher fitness values indicating weaker system performance. In the test we maximize the fitness value. The search is configured using the GUIDE default parameters presented in [12]. The population size was set to 20 individuals due to the long execution times for braking manoeuvres simulated in real-time and to obtain first results on the principal functioning of the testing framework quickly.
In our experiments testing the ABS system with the test automation framework maximising the braking distance using evolutionary testing worked well. In 20 repetitions of the experiment the maximum possible

braking distance of 43 meters was always found within 10-12 generations. In accordance the vehicle velocity was maximised to 25ms^{-1}, the fastest possible speed in our model. Fig. 3 shows the results achieved by one sample test run.

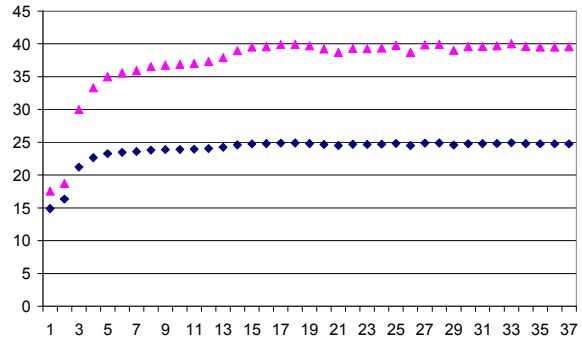

Figure 3: Fastest Vehicle Speed (♦, [$^{m}/_{s}$], y-axis) and Resulting Longest Braking Distance (▲, [m], y-axis) for a Complete Stop of the Vehicle Found in each Generation (x-axis) of the Evolutionary Testing

Real-time test execution for the individuals with the longest braking distance took up to 40 seconds on modularHiL; including speeding up the car to the maximum possible speed and afterwards performing the braking. The total execution time for all generations of one test run took up to 90 minutes. The scalability of the test execution is limited due to the real-time simulation necessary for the correct functioning of the ABS system. However, test cases resulting in the longest braking distance of the ABS system under the constraints applied for the experiment were found with high reliability fully automatically. The complexity of the search space used in the experiment was limited, so that search techniques such as hill climbing or even random testing might yield good results, too. A more realistic test of the ABS system will evaluate the slip withdrawn by the system with respect to different situations detected by the wheel speed sensors and result in search spaces of higher complexities no longer tack able with simple search techniques.

5 Conclusions and Future Work

In this work a test automation framework has been developed on basis of commercial testing products (modularHiL, MESSINA) and research prototypes (EvoTest's evolutionary testing framework) that allows full automation of functional testing on different testing platforms (MiL, SiL, HiL) by applying search based testing techniques. The provided solution supplements systematic testing by reducing the risk of non-testing situations unforeseen by the testers. The test automation framework supports the application of evolutionary testing in very common industrial settings: testing models and software in simulation environments as well as the examination of electronic control units in a hardware-in-the-loop test environment driven by a software-frontend for the definition and implementation of tests. To evaluate the test automation framework testing an ABS system formed the first case study. Test cases were generated using the evolutionary testing framework and executed in SiL and HiL test environments. The search for interesting testing scenarios was successful: driving manoeuvres with long braking distances were found fully automatically proving that functional evolutionary testing is possible and feasible.

Our next step will be a realistic test of the ABS system searching for errors in the system. For this, a fitness function representing the overall slip produced by the system under various difficult circumstances will be defined. Individuals will be transformed into realistic curve traces for the input signals of the ABS system using the signal generator developed by Windisch [11]. To cope with the higher complexity, search space reduction techniques [14] will be applied and the population size will be increased, leading to longer execution times of the test runs.

Acknowledgements

This work is supported by EU grant IST-33472 (EvoTest).

References

[1] Wegener, J.; Baresel, A.; Sthamer, H.: Evolutionary test environment for automatic structural testing. Information and Software Technology, 43(1):841-854, 2001.

[2] Buehler, O.; Wegener, J.: Evolutionary functional testing. Comput. Operations Research. 35 (10), 3144-3160, 2008.

[3] modularHiL: http://berner-mattner.com/automotive-modulhil.php

[4] MESSINA: http://berner-mattner.com/automotive-messina.php

[5] TPT: http://piketec.com/products/tpt.php?lang=en

[6] EvoTest: http://www.evotest.eu/

[7] Dimitar, M.; Dimitrov, I. M.; Spasov, I.: Evotest - Framework for customizable implementation of Evolutionary Testing. International Workshop on Software and Services, October 2008, Sofia, Bulgaria.

[8] GUIDE. http://guide.gforge.inria.fr/

[9] Da Costa, L.; Schoenauer, M.: GUIDE, a Graphical User Interface for Evolutionary Algorithms. GECCO Workshop on Open-Source Software for Applied Genetic and Evolutionary Computation (SoftGEC), 2007.

[10] Tesis veDYNA. http://www.tesis.de/en/index.php?page=544

[11] Windisch, A.: Search-Based Testing of Complex Simulink Models containing Stateflow Diagrams. Proceedings of the 1st International Workshop on Search-Based Software Testing, Lillehammer, Norway, 2008.

[12] Luis Da Costa and Marc Schoenauer: Description of Evolution Engine Parameters, http://guide.gforge.inria.fr/eeparams/EEngineParameters.html

[13] Automotive Standards Committee of the German Institute for Standardization. ISO/WD 26262: Road Vehicles – Functional Safety. Preparatory Working Draft, Technical Report, October 2005.

[14] Harman, M.; Hassoun, Y.; Lakhotia, K.; McMinn, P.; Wegener, J.: The Impact of Input Domain Reduction on Search-based Test Data Generation. Proceedings of the 6th European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 155-164, 2007.

[15] Lefticaru, R.; Ipate, F.: Automatic State-Based Test Generation Using Genetic Algorithms. Proceedings of the Ninth Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing., pp. 188-195, 2007.