Model Based Generation of Driving Scenarios

Thomas Hempen[1(✉)], Sanjana Biank[1], Werner Huber[1], and Christian Diedrich[2]

[1] Research Center CARISSMA, Technische Hochschule Ingolstadt, 85049 Ingolstadt, Germany
{thomas.hempen,sanjana.biank,werner.huber}@carissma.eu
[2] Institute for Automation Technology, Otto-von-Guericke-University Magdeburg, 39104 Magdeburg, Germany
http://www.carissma.eu

Abstract. For the system test of automotive safety systems, thousands of kilometers need to be driven on real roads. In the future, that number will increase significantly through higher complexity of the functions. To reduce that number and guarantee the controllability, reproducibility and increase the flexibility, a high amount of virtual driving kilometers will be done in X-in-the-Loop (XiL) tests, simulating sensors, weather conditions, vehicle dynamics, car drivers, vulnerable road users, etc. Defining these driving scenarios manually is very complex, time consuming and can not be traced to test coverage conditions. This paper presents an approach to extract simulation based driving scenarios from state based test models. Through building a test model of the requirements and expending that with scenery and maneuver information of the driving tests, it is shown, that complete driving scenarios can be generated automatically to reach every possible state of the system under test.

Keywords: Model based testing · MBT · Simulation
Automotive testing · Maneuver based testing · Test case generation
Test · Verification · Validation

1 Introduction

Highly automated and autonomous cars will have an increasing proportion on the roads in the next few years, leading to changes and improvements in ecological, economical and safety aspects [1]. To survive in the dynamic automotive market with its high pressure to new technologies and innovations, the automotive companies, suppliers as well as original equipment manufacturers, have to make more and more tests within every step of the development process. To test the highly interconnected systems, consisting of sensor systems, environment interpretation, driving function and motion control [5], up to billions of driving kilometers would be necessary to ensure the safety requirements within

T. Kováčiková et al. (Eds.): INTSYS 2017, LNICST 222, pp. 153–163, 2018.
https://doi.org/10.1007/978-3-319-93710-6_17

the system test in the future [2]. Because of the higher reproducibility, traceability, flexibility and controllability, those driving tests will be done more and more in simulation based environment [12]. The so called P.E.A.R.S. initiative is working on harmonized and standardized simulation based methods on how to validate advanced driver assistances systems with fully virtual simulation [6]. Within X-in-the-Loop (XiL) tests, where the X stands predominately for model, software or hardware, the simulation offers the generation of data as near as possible to the real world data within a car and can show the later usability and effectiveness of the system under test (SUT) in its final state [3]. With models of sensors, weather conditions, vehicle dynamics, car drivers, vulnerable road users, etc. this is a very mighty and flexible tool to test new functions through the whole development process but has high expenses in designing the test scenarios and finding necessary test cases. In this paper, a method is presented, generating relevant test scenarios for safety critical automotive systems under reproducible conditions with as little effort as possible. Tracing requirements and reaching every possible state of the SUT was essential. Using a state based test model, derived from the requirements, combined with simulation aspects helped to obtain a solution.

The paper starts with related work in Sect. 2 and continues with the description about the idea behind generating driving scenarios from test models in Sect. 3. In Sect. 4, the whole toolchain will be described. After the practical realization and execution in Sect. 5, the paper will end with a conclusion and forecast in Sect. 6.

2 Related Work

For this paper, it is important to differentiate between model-based development (MBD) and model-based testing (MBT). Within the term MBD the system functionality itself will be described and is usually executable. MBT, on the contrary, describes the test cases of the SUT and can be analyzed on the expected results under every possible condition. The third type of model in this paper is within the driving simulation and consists of mathematical models of vehicles, drivers, environment, etc.

For the full test of a system model, test cases have to be generated through analyzing the test model. The simulation offers the whole test environment for the SUT. In contrast to many other MBT methods, no additional environment model has to be implemented. Siavashi and Truscan analyzed the different methods modeling environments for MBT. They selected and analyzed 61 papers according to methods and tools used. Resulting, Unified Modeling Languages (UML) is the most used language to describe environments. Others are Timed Automation, Attribute Event Grammar or Petri Nets [7]. In contrast to that types of environment modeling, using a driving simulator helps to reduce the work to do tremendously and offers the possibility to concentrate on generating driving scenarios for a high test coverage.

Approaches to generating driving scenarios were made through analyzing real world scenarios and reconstructing them virtually [8], generating them through

stochastic algorithms with Markov Chain Monte Carlo methods [9], or rebuilding generic environment descriptions through UML models [4]. In MBT the classic way to describe test cases is, to add parameter values to state transitions to trigger conditions within the system model [10]. This approach will be extended through driving scenario information, resulting in driving scenario descriptions for every possible state in the system model.

In this paper, the MBT approach for the system test is used in combination with driving scenarios. In contrast, other papers do not describe those two in connection, either using MBT to generate test parameters for software components from the verification side, or using models to generate driving scenarios for the validation of the driving function [4]. The direct connection between the system dependent test and the description of the driving scenario was not yet done.

3 Generating Driving Scenarios with MBT

Through the data separation by an already existing driving simulator, the test model and system model can be built by two different development teams, which will be the best approach developing test cases for SUTs. Both teams only have to consider the simulation environment as an interface and relate to the requirements of the system.

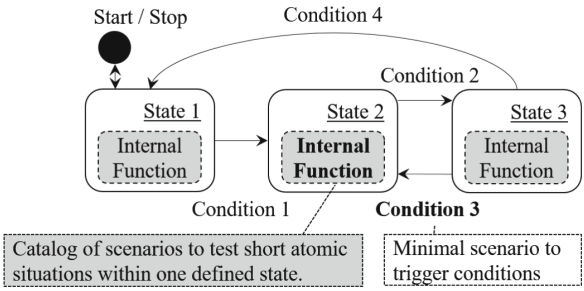

Fig. 1. Example of a state transition model.

The system- and the test model are both designed as state transition models. Figure 1 shows a State Transition Model exemplary. Beginning from the Start, the system will reach different states. Within those states, there are functions/controllers for special driving situations. Parameter changes, timing conditions or external events will lead to state changes [10]. For the test of the system model, two different test items have to be considered separately until reaching the Stop state:

– **Internal function test**
The internal function test will test the reaction of a controller within one single state. The main goal within those tests is to test the system under a high variation of short atomic scenarios, which will represent the variation in the real world as near as possible for a special situation affecting the controller algorithm. Generating a state transition within that scenarios is not permitted. This encapsulation of functionality will lead to a high amount of short driving scenarios which will be controllable in the context of test coverage. Special preconditions for the internal function test have to be considered when generating state transition tests, nonetheless a method for generating these internal function tests will not be main topic in this paper.
– **State transition test**
With the state transitions being in the main focus of the test, hard parameter values are defined for the transition conditions. These conditions will lead to different driving scenarios in the simulation and can be handled under different test coverage conditions.
A much more difficult problem is to ensure the preconditions of the atomic tests scenarios of the internal function tests. Because of the expected high amount of different driving scenarios in that test, the preconditions can not be implemented manually in an acceptable amount of time. Having the post-condition of the previous state, the pre-condition of the internal function test and parameter boundaries through the simulation environment, compare to Fig. 2, the scenario problem can be reduced to a search problem to be solved through evolutionary algorithms [11].

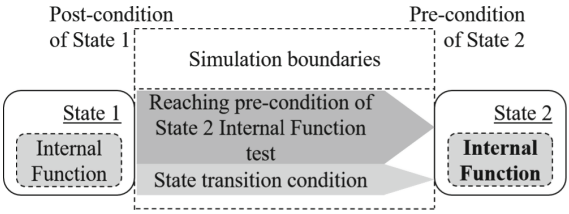

Fig. 2. Boundaries generating state transitions for internal function tests.

4 Framework

To provide test execution with a high grade of automation, a framework for the system model, the test model, the simulation and suitable interfaces for the communication between those tools will be needed. Figure 3 shows an overview of the different tools and utilities used in this project exemplarily. Deriving from the requirements, the system model was developed within Matlab Stateflow as

a state transition model. The test model was developed as a state chart UML model within Enterprise Architect. The XMI data format exported from Enterprise Architect can be imported to the MBTsuite. That tool offers the possibility to extract and filter test cases according to several criteria. An additionally programmed extension to the MBTsuite offers the export of test cases to CarMaker. That simulation tool offers a huge set of models and simulation possibilities to be executed in real-time for driving dynamics, sensors and environment. The system model is directly connected to the driving simulator and can be fed with the data from the CarMaker simulation. The evaluation of executed test cases was done by an additionally developed evaluation tool, which considers different information from the states of the system model and the expected states of the test model and additionally compares the corresponding timing information.

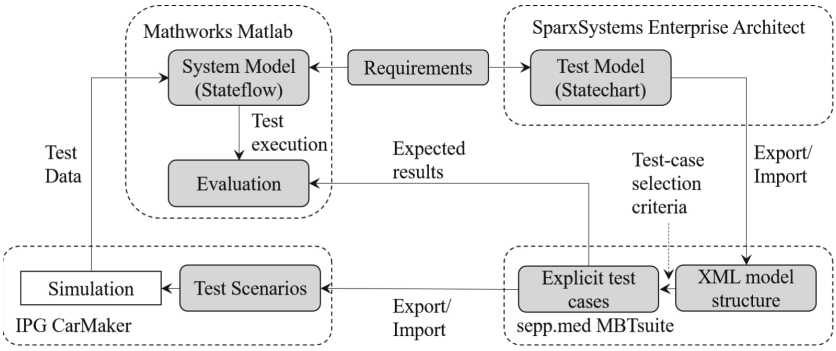

Fig. 3. Overview of the used framework.

4.1 System Under Test

The system model, describes the functionality of the SUT and can be used to simulate the system reactions in an early development state. It is possible to trace the parameter values and state transitions of the model at every time of the execution and analyze the test coverage of the executed test cases after execution. When feeding the model with data from the simulation, the state transitions will be triggered through the driving simulation.

Through automatic code generation, the developed system model can be used until the system is running on the target hardware. Considering the different platforms, the system is running on, there will be differences between analyzing the SUT. Tracing the internal states of the system when running on a special hardware can be difficult. Therefore it is recommended to test stepwise from the model tests increasing up to the hardware tests. The test cases can be reused in every step. While having Gray/White-Box tests on the system model, testing the hardware system could only be possible as pure Black-Box test. It is to be

expected, that timing of the state transition will be slightly different in contrast to the simulated model on the desktop pc without real-time conditions and real bus interfaces.

4.2 Test Model

The test model describes the overall test cases necessary to test the system model. It is also modeled in a state based form in Enterprise Architect. In addition to the system model, it also describes test cases of simultaneously possible system states. For example: Different bus systems can simultaneously receive different messages which could lead to different state transitions within the system model. Every possible combination of data, combinable in equivalence classes, will be modeled as different single state transitions. Considering those possibilities, the test model can get much bigger and more complex than the system model. It is possible to fully generate test cases automatically according to the desired test coverage with the MBTsuite (priority based, branch coverage or guided paths, etc.)

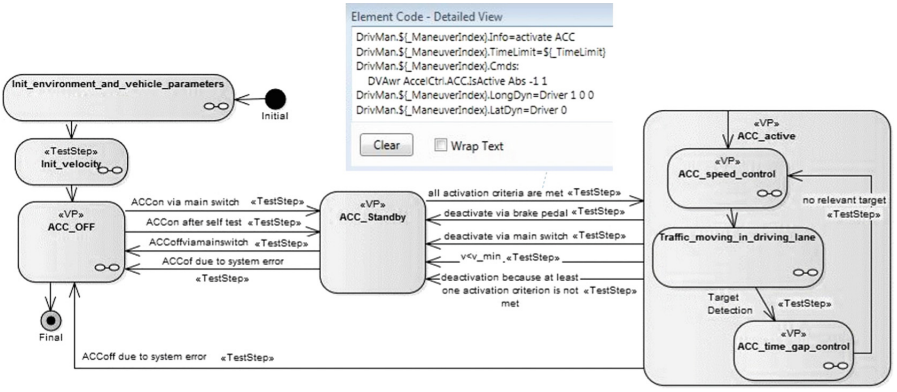

Fig. 4. The test model of the ACC shows the different validation points and the trigger transitions for the system model. Additionally simulation information is added to every test step in form of element code.

In a classical way, test cases would be generated consisting of different test steps, with a pre-condition, a post-condition, the input data and the expected results. The external interfaces of the SUT will be fed with the defined input data, without considering the real environment. The test model was extended to generate test scenarios for the driving simulator through adding environment information (Fig. 4, Element Code). The executed system model will no longer be triggered by the test steps in a data generator, but by the scenario simulator itself, which can simulate the whole system environment. This leads to a much more realistic testing environment in contrary to the methods up to now.

4.3 Simulation

The simulation is one of the most important parts of the framework, providing the information of the environment of the SUT. Poor data can lead to unwanted error states within the SUT, or deliver insufficient or unrealistic test results. Therefore it is important to know the environment of the SUT and integrate the SUT in a way, it would be used in the final target platform. The driving simulator includes a generic model of the environment all around the SUT and can be configured to run in a closed loop [13]. Depending on the test goals, single models can be exchanged through more realistic or better fitting models. E.g., sensor models for environmental sensing. Building a whole simulation environment for virtual driving scenario is not recommended because of its complexity and high costs for the development time. Different companies are specialized in building simulation environments for integrating vehicle components in a closed loop environment, e.g. Vires (VTD), TASS International (PreScan) or IPG (CarMaker). For the research in this paper, an already existing tool was used.

Depending on the selection of the simulation tool, the syntax and grammar of the description of driving scenarios within the test model can change, cf. Sect. 4.2.

5 Example Scenario

5.1 Modeling the Active Cruise Control System

To show the usage of this paper approach, an example implementation of a system model and a test model was made. As an example of the functionality, an Active Cruise Control (ACC) driving function was implemented. The model was implemented according to the ISO15622 and Winner et al. [3]. The ACC can get from state *ACC_OFF* to the state *ACC_standby* until *ACC_active* and backwards. For simplification, the condition from off to standby is the same as from standby to active. When *ACC_active*, the ACC can be in *ACC_speed_control* state, controlling the speed of the vehicle. When detecting a vehicle in front of the ego car, the state *ACC_time_gap_control* will get active and control the distance of the ego vehicle to the vehicle in front. The test model of the ACC, see Fig. 4 consists of different Verification Points (VP), verifying the current internal state of the system model. The transitions within the test model are describing possible events to test in the system model from external inputs or parameter changes. The state *Init_environment_and_vehicle_parameters* is used to initialize the global simulation parameters, that do not change within one scenario.

5.2 Test Case Description

The test case consists of two different aspects separated in bringing the SUT in the expected state (state transition test) and executing a test within the defined state (internal state test). Within the *ACC_time_gap_control* state, the desired time gap between the ego vehicle and the vehicle in front will be changed and the

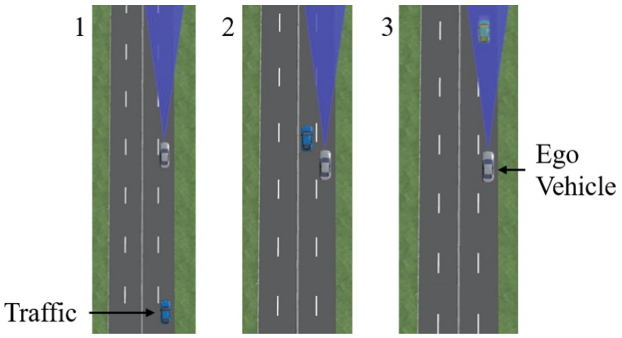

Fig. 5. Driving maneuver of the traffic object and the ego vehicle with its sensor view.

reaction time of the controller measured. The test case generation tool generated the following test chain:

1. *State*: Init_environment_and_vehicle_parameters
2. *State*: Init_velocity
3. *State*: ACC_OFF (VP)
4. *Transition*: ACC on via main switch (test step)
5. *State*: ACC_standby (VP)
6. *Transition*: ACC activation - no further activation needed in this example (test step)
   (a) *State*: ACC_speed_control (VP)
   (b) Traffic_moving_in_driving_lane (test step)
      i. *Transition*: relevant target detected (End condition test step)
   (c) *State*: ACC_time_gap_control (VP and starting the internal state test)
      i. Waiting for condition: acceleration of the ego vehicle less than $0.01\,\mathrm{m/s^2}$
      ii. Changing the parameter time_gap
      iii. Waiting for condition: acceleration of the ego vehicle less than $0.01\,\mathrm{m/s^2}$
7. *Transition*: ACC off via main switch (test step)

It is necessary to add an initial state *Init_velocity* for traffic vehicles in this tool set. Within this state, the velocity of the traffic vehicle will be initialized at the beginning of the test scenario. To reach the state *ACC_time_gap_control* an interaction with the environment is required. It is necessary to have another vehicle in front of the ego car. Therefore a driving maneuver of the traffic object is needed and modeled within the test model state *Traffic_moving_in_driving_lane*. Figure 5 shows the full scenario. The traffic object is overtaking the ego vehicle and drives into the sensor view while decelerating to a lower speed than the ego vehicle to activate the time gap controller.

5.3 Execution and Evaluation

The example test case was generated via guided path functions meaning that the test case was defined previously manually. As an example, a traffic car is overtaking the ego vehicle and reaching the sensor view of the ego vehicle, decelerating to trigger the time gap control function in the ego vehicle. The evaluation of the test case can be done automatically and offers visual evaluation of the timing of the state transitions and the parameters. Figure 6 shows the evaluation of the example scenario. The X axis shows the timing with a resolution of 1 ms and the Y axis on the left side the different possible states. On the right side, parameter values are scaled visualized. The top graph shows the states *ACC_OFF*, *ACC_standby* and *ACC_active*. The bottom graph visualizes the internal states of *ACC_active*, *ACC_speed_control* and *ACC_time_gap_control*. As an addition a None state, if *ACC_active* is not active. Starting from time 6 s, the test case triggers the transition from *ACC_OFF* to *ACC_active*. The enlargement in the red box shows the corresponding transitions. The green graph shows the expected transitions generated from the test model. The blue graph, the transitions within the system model (SUT) and the yellow one the signals in the simulation tool executing the test case. For that transitions, the system model did take 1 ms for every transition.

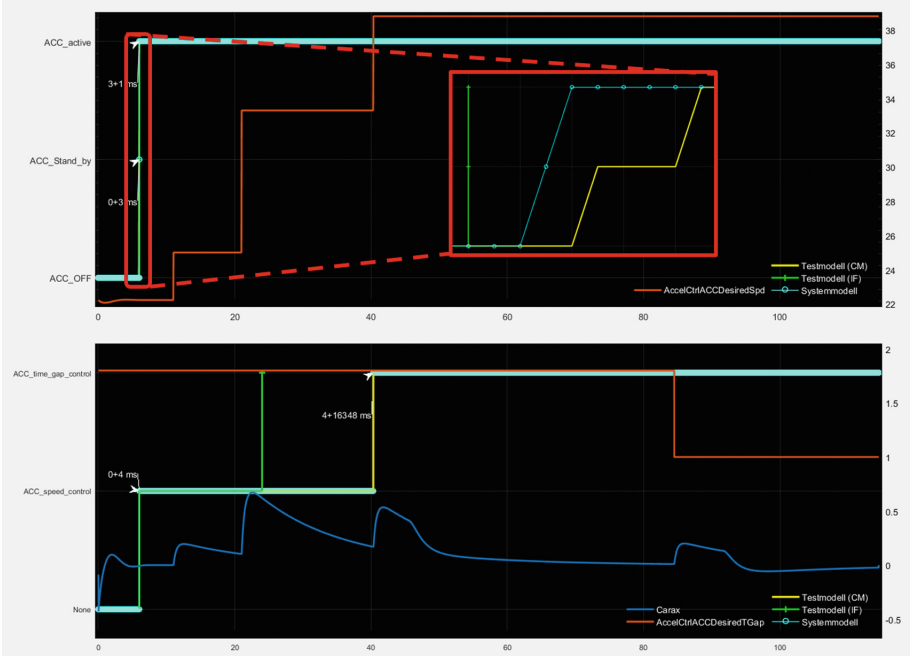

Fig. 6. Test evaluation through timing visualization of state transitions and parameter values. (Color figure online)

Corresponding to the state transitions, the parameters *desired speed* of the ego vehicle (top), *desired time gap* (bottom) and *acceleration* of the ego vehicle (bottom). At 24 s, it is noticeable that the test model defines a transition to *ACC_time_gap_control*. In this situation, the traffic vehicle begins with its overtaking maneuver. At about 40 s, the traffic vehicle reaches the range of the sensors of the ego vehicle and the state transition will be triggered. As a precondition for the internal state test of *ACC_time_gap_control*, the acceleration of the ego vehicle has to be less than $\pm 0.01\,\mathrm{m/s^2}$. Triggered through that condition, the desired time gap will be changed and the resulting acceleration of the ego vehicle can be analyzed. Summarized, the test case reaches the expected state within about 40 s. The parameter changes from time 80 s to test end are the test of the internal state function. It is important that no other state transitions will be triggered within that test. With the focus on the state transition test, there are several possible errors to be detected:

- Timing delays of worst case execution time overflow within the system model
- Wrong state transitions because of errors within the transition conditions
- Unexpected parameter values that lead to unexpected transitions.

6 Conclusion and Forecast

Within this first attempt in building simulation based driving scenarios from test models, it was shown, that the method can be very helpful in efficiently finding errors within a system model. It could be proved, that the driving scenarios can be built from the test model and shown, how the framework is working together. Through the driving simulator, environmental sensors can be included for testing advanced driver assistance systems and highly automated driving functions in the future. For this implementation, the atomic scenarios for state transitions and internal state tests were made manually and could be complex considering pre- and post-conditions for the chained atomic scenarios. At this point, further work will be done in integrating evolutionary optimization methods already done in earlier projects [11]. Through describing driving scenarios with the test model, a gap can be closed between the real driving tests and the component test. Through the test model, the driving scenarios will trigger every internal state transition of the SUT without differentiation between small component tests, or high level system tests. It is usable under both conditions.

Acknowledgements. This work is supported under the funding program Forschung an Fachhochschulen of the German Federal Ministry of Education and Research (BMBF), contract number 13FH7I01IA (SAFIR).

References

1. Roehrleef, M., Deutsch, V., Ackermann, T.: Scenarios for autonomous vehicles - opportunities and risks for transport companies. In: Position Paper, November 2015, Verband Deutscher Verkehrsunternehmen e.V. (VDV), Cologne, Germany (2015)
2. Kalra, N., Paddock, S.M.: Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? RAND Corporation (2016). http://www.rand.org/content/dam/rand/pubs/research_reports/RR1400/RR1478/RAND_RR1478.pdf
3. Winner, H., Hakuli, S., Lotz, F., Singer, C. (eds.): Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort. ATZ/MTZ-Fachbuch, pp. 126–138. Springer, Wiesbaden (2015). https://doi.org/10.1007/978-3-658-05734-3
4. Bach, J., Otten, S., Sax, E.: Model based scenario specification for development and test of automated driving functions. In: IEEE Intelligent Vehicle Symposium (IV), Gothenburg, Sweden (2016)
5. Sahin Tas, O., Kuhnt, F., Zoellner, J.M., Stiller, C.: Functional system architectures towards fully automated driving. In: 2016 IEEE Intelligent Vehicles Sysmposium (IV), Gothenburg, Sweden (2016)
6. Yves, P., Fahrenkrog, F., Fiorentino, A., Gwehenberger, J., Helmer, T., Lindman, M., op den Camp, O., Rooij, L., Puch, S., Fränzle, M., Sander, U., Wimmer, P.: A comprehensive and harmonized method for assessing the effectiveness of advanced driver assistance systems by virtual simulation: the P.E.A.R.S. initiative. In: 24th International Technical Conference on the Enhanced Safety of Vehicles, Gothenburg, Sweden (2015)
7. Siavashi, F., Truscan, D.: Environment modeling in model-based testing: concepts, prospects and research challenges: a systematic literture review. In: EASE 2015 Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, New York, USA (2015)
8. Zofka, M.R., Kuhnt, F., Kohlhaas, R., Rist, C., Schamm, T., Zoeller, J.M.: Data-driven simulation and parametrization of traffic scenarios for the development of advanced driver assistance systems. In: 18th International Conference on Information Fusion, Washington, USA (2015)
9. Prialé Olivares, S., Rebernik, N., Eichberger, A., Stadlober, E.: Virtual stochastic testing of advanced driver assistance systems. In: Schulze, T., Müller, B., Meyer, G. (eds.) Advanced Microsystems for Automotive Applications 2015. LNM, pp. 25–35. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-20855-8_3
10. Bringmann, E., Kraemer, A.: Model-based testing of automotive systems. In: 2008 International Conference on Software Testing, Verification and Validation, Lillehammer, Norway (2008)
11. Sattler, K.: Methodik fuer den Systemtest in der integralen Fahrzeugsicherheit. Dissertation, Otto-von-Guericke University Magdeburg, Germany (2015)
12. Riener, A., Wintersberger, Ph., Hempen, T., Lauerer, Ch., Hasirlioglu, S., Reway, F.: A flexible mixed reality test environment to improve simulation-based testing for highly automated driving. In: Aktive Sicherheit und Automatisiertes Fahren. Expert Verlag, Germany (2016)
13. Dirndorfer, T., Roth, E., von Neumann-Cosel, K., Weiss, Ch., Knoll, A.: Simulation environment for the development of predictive safety systems. In: Proceedings of the FISITA 2010 - World Automotive Congress, Budapest, Hungary (2010)