A Scenario-Centric Approach for the Definition of the Formal Test Specifications of Reactive Systems

Vladimir Entin, Sebastian Siegl, Klaus Meyer-Wegener
Universität Erlangen-Nürnberg
vladimir.entin|sebastian.siegl|klaus.meyer-wegener@cs.fau.de

Andreas Kern
Audi Electronics Venture GmbH
andreas.kern@audi.de

Michael Reichel
Technische Universität Braunschweig
extern.michael.reichel@audi.de

Abstract

Complex modern embedded automotive software systems require different test techniques in each of the development stages. Most common are Model in the Loop, Software in the Loop, and Hardware in the Loop. The majority of these test techniques are automated. Each uses different notations for test data, pass/fail criteria, system-under-test interface definition and test-platform-specific parameterization. This leads to a series of problems such as exchangeability of test-specification notations among different teams working on the same functional module, reusability of test cases and uniformity of test-specification representation. This contribution proposes an approach for the formal and test-platform-independent definition of the test specification of reactive systems. Additionally, the application of the approach in three concrete use-case scenarios elicited in a pre-development department of AUDI AG is shown.

1. Introduction

The development process of complex embedded automotive software systems such as e.g. driver assistance systems (DAS) consists of various stages. In each of these stages, different established test techniques such as Software in the Loop (SiL), Model in the Loop (MiL) and Hardware in the Loop (HiL) [11], or tests with actual measurement data are employed. Once a certain degree of maturity is achieved, the software module along with such artifacts as requirement document and test specification are passed to another development team for the next stage. At each stage requirements and test specification are further elaborated and completed. Although in [4, 10] approaches have been proposed to formalize the requirements definition, this is still lacking for the test specifications which are either informal or quite test-platform-specific (e.g. test scripts or XML documents). Thus they often cannot be used as they are by the next development team. That in turn leads to a loss of test knowledge.

For instance, the pre-development department of a car manufacturer and a supplier work together on the same functionality. The pre-development department initially develops first prototypes and specifies the requirements. The task of the supplier is to implement and optimize the Electronic-Control-Unit (ECU) software for use in the serial production. Finally the supplier submits the software module to the Original-Equipment-Manufacturer (OEM) department whose goal is the execution of system and acceptance tests according to the requirements.

Another important observation to be made is that, in practice, even within a single development team, there is often no standard representation of the test specification. For example, in the area of pre-development of DAS there are, on one side, maneuver catalogs for the actual test drives and, on the other side, a set of XML documents describing a drive scenario that allow its execution in the simulation (SiL). In the latter case the knowledge of the requirements an ECU software module has to fulfill (pass/fail criteria) is solely present in the mind of the module developer. The goal of this contribution is to propose a test-platform-independent test-specification description for the early development stages of reactive systems with complex environmental interaction such as e.g. DAS. It can be reused in all the following development stages.

 DOI: 10.1109/TAICPART.2009.21
The original publication is available at http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05381630

2. Approach

2.1. General Concept

The approach is conceptually divided into five levels (see Figure 1). The artifacts created and gradually refined at each level are represented either as models defined by the domain engineer or as their instances specified by the domain expert (section 2.2) which might be stored e.g. in a database. A domain expert is usually a person directly involved into the development of the SUT (whereas a domain engineer is concerned with the formalization of the test domain). The representation of each artifact does not change from level to level except the transformation which takes place at Level 5.

Level 1: scenario definition – Firstly, we introduce the notion of *scenario* which is a formal description of the complex interactions of the system under test (SUT) with its environment. The latter not only considers time aspects of the interaction, but also the absolute and relative position of the participants taking part in the interaction. The scenarios are not derived from the functional requirement specifications of a specific software module as in [1, 8], and they do not represent a desired behavior of the SUT. They are just a formal description of the complex interaction which takes place between the SUT and the actors which influence the behavior of the SUT. The data generated during the execution of such a formal scenario description can be employed as test data. In the context of DAS such test data comprises e.g. static and dynamic object lists detected by the sensor units, videos recorded either by virtual simulation or by a real camera during a measurement drive, implicitly recorded information about the lane, weather and light conditions. In practice, we have observed that the scenarios tend to be used for several projects within a single development stage. Moreover, they are oftentimes, at least partly, reused within a certain development stage for a new DAS. Hence, the scenarios basically do not depend on the functional requirements of a certain function. The basis for the formal description is considered to be mostly the knowledge of the developer or, in certain cases, electronic documents with informal scenario descriptions. For the area of DAS we have chosen the statechart method [6] as the formal description. It is created by the DAS developer, that is, domain expert, and does not refer to any specific test technique nor to any development project. The scenarios are described by a small predefined set of generic attributes which are defined in our meta-model-based approach (section 2.2). We define, for example, current velocity as a generic attribute for the acceleration and the lane change drive maneuvers.

Level 2: project and testing technology selection – Usually each development stage (section 1) is subdivided into various projects. At this level the developer has to decide for which project and which specific test technique within the project the test specification has to be created. He further chooses a subset of pre-defined generic scenarios (Level 1) for further refinement in a specific project and test technique. As mentioned before, various test techniques are usually applied within single projects. In the early development stages of the DAS typically MiL and SiL are employed. Furthermore, the so-called "open loop" SiL automated test run uses very large files with data recorded previously during measurement drives and feeds them into the DAS module. This test method is called "open loop" because the module output has no influence on the environment nor on the module itself. Lastly, actual test drives are of great importance.

Level 3: scenario completion – According to the chosen project and test technique the developer completes the selected generic scenario description with the project-specific attributes. For example, projects dealing with image-based lane-detection algorithms will need such information as lane marking or the light conditions whereas systems supporting the driver during the lane change will need such attribute as lateral acceleration of the the vehicle in which the SUT has been installed (EGO-vehicle).

Level 4: test specification creation – At this level the developer completes a test specification by including the project-specific scenario description (Level 3) as well as the following information: The set of attributes which specifies the desired SUT behavior is determined. They have to be further analyzed by the domain engineer whether they are generic or specific to a certain project or test technique. Then they are added to the corresponding part of the model so that the developer during the test-specification definition only has to define specific values. Due to the meta-model-based approach (section 2.2) these attributes can easily be managed and classified. We consider the desired SUT behaviour as being part of a test specification created for a certain project and a test technique (see Figure 1) and not as a part of a project-specific scenario. Thus, it is a separate artifact. Since in the early development stages of reactive systems primarily black-box testing of complex handwritten software modules is applied, we do not specify a behavior model of a DAS module as is done in the area of model-based testing [9, 2]. Instead the abstract interface of the SUT under a specific test technique as well as its actual use have to be defined.

Level 5: test specification transformation – The goal of the transformation is the creation of a test-case descrip-

tion and the information needed to start an automated or, in some cases, manual test process. The automated test process is basically concerned with the automated test execution whereas the manual one might involve not only the execution but also the creation of the test data such as e.g. measurement data recorded during a drive. At this level the entire set of artifacts created and refined at the Level 4 is employed. The transformation process is triggered by the domain expert however certain preparative steps have to be taken previously by the domain engineer (section 2.2). The result of the transformation is a platform specific representation of the test specification defined at all the previous levels. Additional information might be created during the transformation (section 2.2) as well.

A particular role in the transformation is reserved to the formal scenario description. Depending on the test technique and, above all, on the test platform, a scenario description can be transformed into a test-data description. This could be a complex representation of the traffic and the behavior of the vehicle under test for a SiL run. But in the case of actual test drives, it could just as well be a document containing a set of guidelines for the test driver.

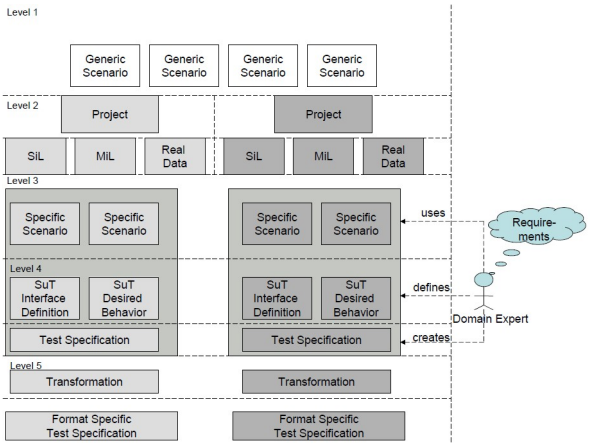

Figure 1. Approach for platform-independent test specification

2.2. Process

In this section we present steps to be completed for the application of the approach.

Step 1: meta-model definition – In this step a thorough evaluation of the already productively used scenario-description formats such as e.g. drive-maneuver catalogs, XML-based SiL traffic descriptions or videos of already completed measurement drives has to be accomplished. Since most formal specification techniques, including statecharts [6], are too generic for a representation of the scenarios, one has to restrict them by defining an appropriate meta-model. Moreover, in order to improve the comprehensibility of the chosen formal method, modeling guidelines have to be elaborated by the domain engineer. They give specifics e.g. on how to model concurrent or synchronized execution of certain actions. For the DAS area an exemplary guideline would specify the lane numbering and the vehicle positions relatively to the EGO-vehicle.

Finally, the meta-model must provide the possibility to define such important parts of the test specification as the desired behavior (pass/fail criteria) and the SUT parameterization. Clearly, their representation, similar to that of a scenario, must allow a distinction between generic and project-specific attributes.

Step 2: partial instantiation of the meta-model – In this step the domain engineer elaborates in cooperation with the domain experts the following information: concrete categories for the elements of the test specification, concrete elements for these categories such as states, events, guards, actions and finally generic, project-specific and test-technique-specific attributes for these elements.

At last he instantiates the meta-model with the gathered elements.

Step 3: creation of the test-specification model and its instantiation – By means of an editor a domain expert creates a model of the test specification and then instantiates it with the concrete values. Instead of creating a completely new test-specification model, he can select an existing one and instantiate it with the appropriate values. In the area of reactive systems, test specifications, especially test data, often show the same structure but different parameter values, so the separation into test-specification model and test-specification instance increases the reusability of the test specification.

Step 4: transformation – Usually, before the transformation process starts, the domain expert has to create a transformation template. Per test technique there is one template to be created. As regards the specification of the test environment some approaches [3, 7] suggest the definition of this at the model level. Our experience has shown that information related to the test environment (mostly call parameters of the execution and evaluation environment, the configuration parameters of a database or configuration of a test runner) is altered in rare cases, so it is advisable to create it in the transformation step, that is, in the template.

An example of the described process is shown in Figure 2.

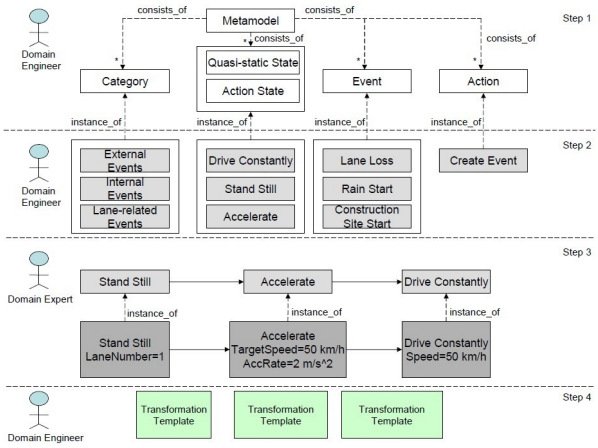

Figure 2. Example of test specification definition process

3. Use-Cases

Our approach is currently being evaluated at a DAS pre-development department of AUDI AG. During the evaluation phase we have identified three concrete use-cases which are presented in the following.

3.1. Use Case 1: Actual Measurement Drives

Measurement drives are often carried out at a dedicated test site. A thorough preparation for the latter is indispensable. By means of the graphical editor the developer creates the scenario and the corresponding test-equipment descriptions. In the next step they are stored in a drive scenario database [5]. During the transformation phase an electronic document is created with the description of the scenario in a table form. A driver performs the measurement drive according to the document and stores the recorded data in the database. The developer also has to assign the data file to a concrete instance of the scenario which has previously been saved in the database by the developer. In order to execute the automated SiL open loop test run, the DAS developer has to complete a test specification in which he specifies the data needed for the automated test run (section 2.1). Next he selects the appropriate transformation template which creates a set of configuration files so that in a couple of mouse clicks he is able to start the automated test run.

3.2. Use Case 2: Variation of Scenario and Test Technique

The developer would like to start a test run with the scenario which has been already recorded at the test site however with slightly different parameters that would make its execution under real-life conditions impossible. He might, for instance, change the distance of the tailgating vehicle under test to the target vehicle to a very small one so that the danger of collision under real-life circumstances would be extremely high. By applying our approach the DAS developer simply creates a new test specification. He might also use the existing one in case SUT parameters and pass/fail criteria are the same as in the SiL open loop test run with the measurement data. The test specification includes a slightly changed scenario instance which is completed with the project specific attributes. The former is consequently transformed into the file set necessary to execute the SiL test run. This use case demonstrates a multi-platform application of a scenario.

3.3. Use Case 3: Transfer of the Test Knowledge

So far the application of the approach within one automotive software development phase i.e. pre-development was described. The question which arises is how to transfer the test knowledge from this phase to further development stages such as e.g. serial development and especially the department concerned with the system and acceptance test. They receive an informal functional specification and a formal test specification with scenarios from the pre-development. The developers refine the functional specification. In the next step the given formal test specification is completed. Thus, the definition of the scenarios is reused in the creation of the test specification in the serial development (Figure 3).

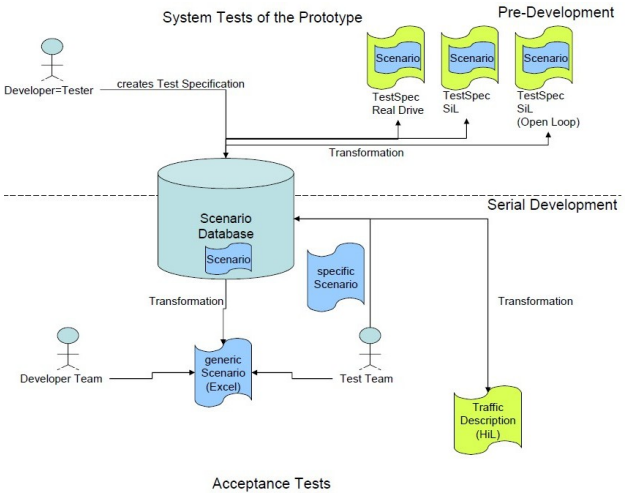

Figure 3. Re-use of scenarios across development phases

Due to our transformation approach the formal generic

scenario descriptions could be used as table documents (Figure 3) for the communication between the developer team and the test team. The test team could then complete them with the project specific information and employ it e.g. for the automated creation of the virtual traffic description for the HiL test run.

4. Implementation

We are currently implementing a set of tools for the approach. The modeling hierarchy is stored in the drive-scenario database. A .NET-based graphical user interface allows the definition of the test specification and its storage in the database. The transformation templates are managed in the same database. During the transformation phase a test specification is being extracted from the database and transformed to the .ecore format by a dedicated tool. In the next step the Open Architecture Ware is triggered by the aforementioned tool. The former takes as input the meta-model definition of the test specification, the instance of the latter and the transformation template. We are using XPand as transformation language. The scenario database is currently being used by four different departments at AUDI AG.

5. Discussion

It is essential to introduce the presented approach into the daily development process of the departments and to consider the cost of the introduction. On the one side there is the effort to gain acceptance of the formal scenario notation from the domain experts. This effort can be reduced by choosing the notation which the majority of the DAS developers know from the modeling environments already in use, e.g. MatLab/SimuLink. The modeling guidelines prove to be of great help to improve the comprehensibility of the chosen notation and as consequence the successful introduction of the approach.

The second aspect to be taken into consideration is the time invested in the completion of a scenario description or a test specification. Until now the scenario descriptions have been created and stored locally by each DAS developer so that a variety of them existed often duplicated. By introducing a clear separation between the test specification model and its instance our approach allows the developer to avoid the duplication of the test-specification description and improves the reusability of the latter. Clearly, the cost of manual creation remains, although it is possible to complete manually created scenario descriptions e.g. by the data recorded during a measurement drive in order to further reduce it.

6. Acknowledgement

We would like to express our deepest gratitude to Dr. Karl-Heinz Siedersberger for the continuous support and help.

References

[1] S. Baerisch. Model-driven test-case construction. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 587–590, New York, NY, USA, 2007. ACM.

[2] S. Benz. AspectT: aspect-oriented test case instantiation. In *AOSD '08: Proceedings of the 7th international conference on Aspect-oriented software development*, pages 1–12, New York, NY, USA, 2008. ACM.

[3] L. Borner, T. Illes-Seifert, and B. Paech. The testing process - a decision based approach. In *ICSEA '07: Proceedings of the International Conference on Software Engineering Advances*, page 41, Washington, DC, USA, 2007. IEEE Computer Society.

[4] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 285–294, New York, NY, USA, 1999. ACM.

[5] V. Entin. A database for the management of drive scenarios during the development of driver assistance systems. Master's thesis, Friedrich-Alexander University Erlangen-Nuremberg, Audi Electronics Venture GmbH, 2006.

[6] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[7] Object Management Group. UML Testing Profile. Version 1.0 formal, 2007.

[8] C. Pfaller, A. Fleischmann, J. Hartmann, M. Rappl, S. Rittmann, and D. Wild. On the integration of design and test: a model-based approach for embedded systems. In *AST '06: Proceedings of the 2006 international workshop on Automation of software test*, pages 15–21, New York, NY, USA, 2006. ACM.

[9] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner. One evaluation of model-based testing and its automation. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 392–401, New York, NY, USA, 2005. ACM.

[10] M. Satpathy and S. Ramesh. Test case generation from formal models through abstraction refinement and model checking. In *A-MOST '07: Proceedings of the 3rd international workshop on Advances in model-based testing*, pages 85–94, New York, NY, USA, 2007. ACM.

[11] F. van Meel, G.-P. Duba, T. Bock, and B. Strasser. Developing properties for driver assistance systems by means of an inovative and constant development process. *FISITA 2008 World Automotive Congress - Springer Automotive Media*, II, 2008.