Model-based testing of car infotainment systems with SysML[*]

Oliver Alt
Robert-Bosch GmbH, CM-DI/EAA3
Daimlerstr. 6, D-71229 Leonberg
oliver.alt2@de.bosch.com

Abstract

New functions in car infotainment systems like internet connection or 3D navigation require new methods to ensure the quality of such systems. The system test is the last instance of quality insurance before the system software is delivered to the customer. Textual and informal graphical specifications are the base for system test development today. Human test developers read these specifications and develop test cases by hand. With an increasing part of new functions and software this method of test case development leads to increasing costs and needs of human resources. To improve this situation model-based testing is used. With model-based testing, test information can, in many cases automatically, be derived from the system model. The systems modeling language (SysML) is a new OMG standard and an UML2 profile for system modeling. In this paper the concepts of an approach to derive test information for the system test of car infotainment systems from a SysML model are presented. The approach allows for the derivation of automatable and reusable test cases for different projects from one domain specific system model for car infotainment systems. Functional system behavior is thereby separated in a functional and a product specific part. The new concepts from SysML, like extended activity modeling, definition of value types and allocations are applied and adapted to develop the system model. This model replaces or supplements the informal specification and allows derivation of test cases automatically. Activity diagrams are used to specify the system behavior. A realistic example of a MOST (Media Oriented System Transport) audio system illustrates the developed concepts.

1 Introduction

Nowadays car infotainment systems are specified using textual documents and in some cases informal graphical diagrams. These non-formal specifications are the input and base for the software system test. So the system test process is a human centric process in a high degree, because the test engineers have to read the textual specifications and than develop test cases to test the car infotainment system. Figure 1 illustrates the test process today. After the test specification, the next step is the test execution in the test environment. Finally, the results must be analyzed.

New requirements like internet connection, 3D navigation or connecting consumer electronic devices lead to an increasing complexity of such car infotainment systems. Furthermore a modern car infotainment system is a distributed system of several components from different

[*]Published in: R. Ester (ed.), Proceedings of the embedded world conference 2007, 2 2007; embedded world conference 2007 Proceedings and Conference materials, Design & Elektronik.

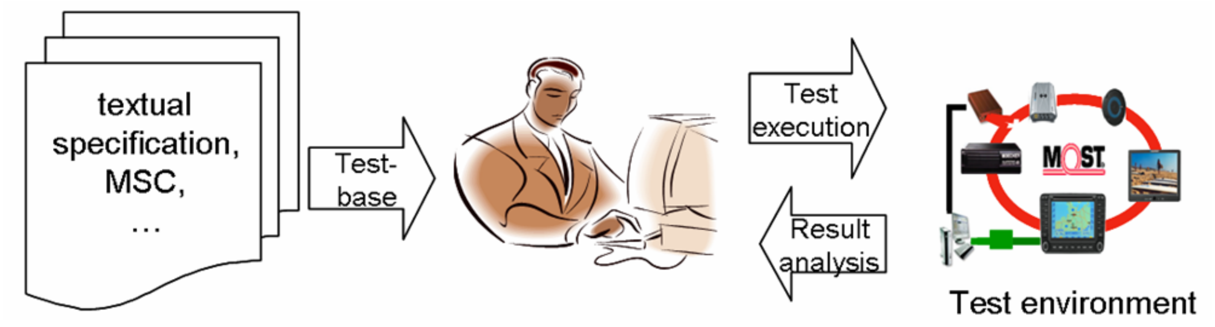

Figure 1: System test process today

vendors. These components (e.g. CD changer and audio amplifier) are typically interconnected using the Media Oriented System Transport (MOST) [1] network system. You can imagine the complexity of such a system if you know that for a typical MOST system in a modern car more than 32000 possible valid communication messages are specified.

To ensure the software quality of such a car infotainment system, more and more testing is required. By virtue of limited budget and project development time it is not possible to write all the required tests by hand. This leads to a very low test coverage and perhaps to undiscovered software bugs.

A second problem is that product specific informations are included in the test cases. For example to test the function *Start playback* of a CD player in one project you have to send a specific MOST message, but in another project you have to press the play-button. In the test case for the first project a test step *Send MOST message: AudioDiskPlayer.DeckStatus(Play).* is included and for the second project the test step is *Press play button*. Both test cases are testing the same function, but you can not use the same test case for both systems.

To improve the situation described above two problems must be solved:

1. Separation of the product specific informations from the test cases to improve the reusability in similar products.

2. Replacement of the informal specification by a more formal one - a system model - to be able to derive test cases automatically (Model-based testing).

2 Modeling language

As modeling language the Systems Modeling Language (SysML) [2] was chosen, because it is a new OMG standard for system modeling, based on the Unified Modeling Language (UML) [3] and brings the required formality of a graphical specification language. SysML is defined as an UML profile. This means it is a domain specific extension of the UML for system modeling. UML is a widely accepted standard in industry and there are many tools available on the market.

Unlike UML, SysML knows only 9 diagram types, in opposite of 13 diagram types in UML. Furthermore some of these 9 diagrams are new types to support specific aspects of systems engineering. This makes the language easier to learn and to handle for system engineers. With own profiles it is possible to add other domain specific aspects to SysML. This makes it possible to use standard modeling concepts and tools and the advantages of domain specific modeling, too.

As modeling tool Enterprise Architect [4] is used, because it supports SysML and UML2 modeling and has an add-in interface to extend the functionality with own tools and programs.

3 Model-based testing concept

To solve the problem of the included product-specific information in the test cases, the model is structured in several, different parts. One part contains the functional aspects of the system. This part includes the system description from a product unspecific point of view. System descriptions from the functional part are valid for similar systems from the same domain. The functional model part describes *what* the system is able to do, resp. what functionality the system has, but not *how* this functionality is realized for the concrete product.

The description how the functionality is realized is described in a separated product-specific model part. So we have one functional part, and for every product a product-specific model part.

The functional model part is a little bit similar to the platform independed model (PIM), known from the Model Driven Architecture (MDA) [5] approach from OMG. The main difference is, that in MDA the platform specific model (PSM) is completely generated by a transformation. In opposite of that the product-specific model part used and described here is modeled by the system designer.

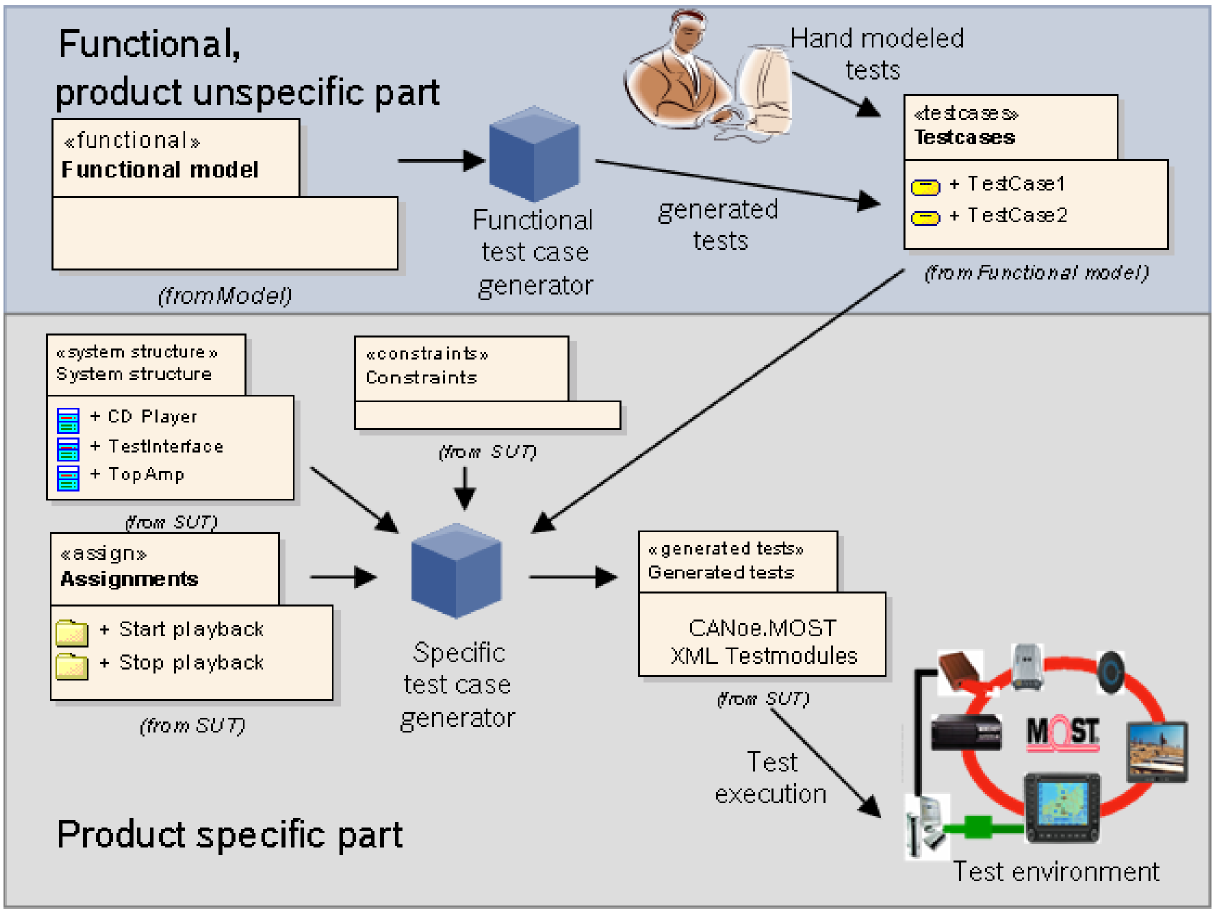

Figure 2: Model-based testing process

Figure 2 illustrates the changed working process, while applying the model-based concepts for the system test. Now the process is model-centric in opposite to the human-centric process in figure 1. The cubic nodes symbolize generation steps for test cases on the functional level and the translation from functional to product-specific test cases processes.

Beside the automated generation of test cases it is also possible to specify tests by hand and supplement the test pool. These hand modeled test cases are also specified on the functional

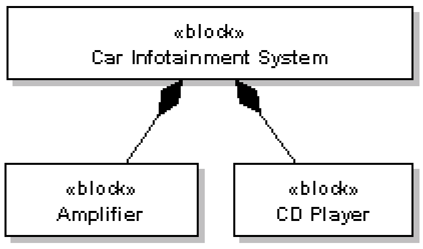

Figure 3: Functional system structure as SysML block diagram

level and can be reused to test different similar products. So the model-based testing concept is a hybrid process and is able to combine test case generation and traditional manual test specification.

3.1 Behavior modeling

To be able to derive system test information from the model, it is required to design the model structure with regard to this goal (*Design for testability*).

The main goal is to model the system specification and not the tests (system modeling vs. test modeling). To define system requirements the usage of use cases is a widely accepted approach. Use cases describe system use scenarios as a series of user and system activities, resp. actions.

Another question to answer is, what is the essence of a system test case. A system test case normally consists of two steps:

1. Trigger the system from the user point of view (*trigger action*).
2. Validate the system reaction by comparing the behavior of the System Under Test (SUT) with the specified behavior (*validation action*).

You can see, that use cases and test cases have a common ground - usage of activities, resp. actions. Thus for modeling the system behavior, activity modeling from SysML was chosen as main behavior modeling concept. Activity diagrams in UML2 and SysML are extended with new elements and a more precise semantic definition (based on Petri-nets). Furthermore SysML extends activity modeling with the possibility to change the run-state of an activity (*control operator*) - a concept missing in UML2 and often needed in system design.

4 Functional part

The functional model part comprises the domain specific aspects of the system specification. In this paper an example audio system with CD player and audio amplifier is used. Figure 3 shows the logical structure of the audio system as an SysML block diagram.

Beside the structural description, the specification of the dynamic behavior is the main part of the model. The packaged containing the dynamic, behavioral part are named equally to blocks used to describe the system structure. The first step in the behavior definition is to define the user and system activities. Examples for user activities are *Start playback* or *Set volume*. User activities are all activities, a user can start to interact with the system. They describe the system from a user point of view. Figure 4 gives further examples for user activities.

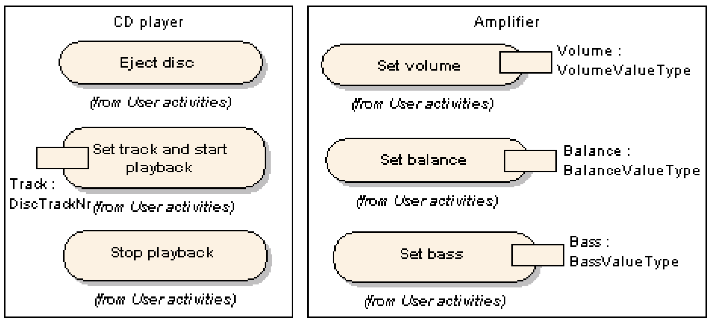

Figure 4: Example for user activities

System activities are activities from the system point of view. An example for such a system activity is *Playback* for the CD player subsystem.

Use cases [6] are a mechanism, widely used to describe behavioral aspects of systems. A use case is a flow of such user and system activities similar to the activities described above.

Test cases are sequences of user activities and subsequent validation actions to check the right or wrong system reaction. A simple example of such a test case modeled as activity diagram is given in figure 5. The system activities are not a part of system test cases, because the system is seen as a black box and for a black-box-test the internal realization, described with the system activities irrelevant. System activities may be used for test generation purposes and for using the model as test oracle (see description of test generation idea in section 7.2).

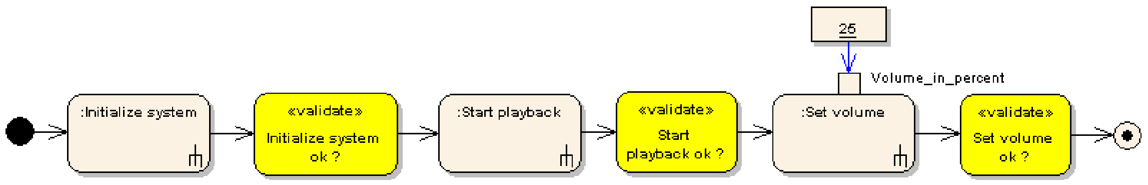
25
Volume_in_percent
:Initialize system
«validate» Initialize system ok ?
:Start playback
«validate» Start playback ok ?
:Set volume
«validate» Set volume ok ?

Figure 5: Example for simple test case

The automated validation of the system reaction for a car infotainment system is hard to realize, because there are things like dynamic HMIs, navigation maps and audio, resp. video signals. Thus in the first iteration of the model-based testing approach the validation is done half automatically by the test engineer. Every trigger action in the test case is followed by a subsequent validation question (Stereotype *validate*).

Functional test cases are applicable to all similar systems of the same domain, because they include only functional aspects of the system. With the test case from figure 5 you can test all systems with the capability to start a playback operation and set a volume. This improves the reusability for the test cases and the reuse for quality insurance of many similar products.

5 Product specific part

To automate the execution of the trigger activities in the test cases, an assignment must be defined between the user activities and product-specific actions, e.g. send specific bus message sequence. Activity modeling is used for this purpose. For every user activity in the functional model part, an activity sequence is defined, which describes the product specific action sequence that must be executed to achieve the desired behavior for the system under test. To model the domain specific aspects of the car infotainment domain, a UML profile was defined.

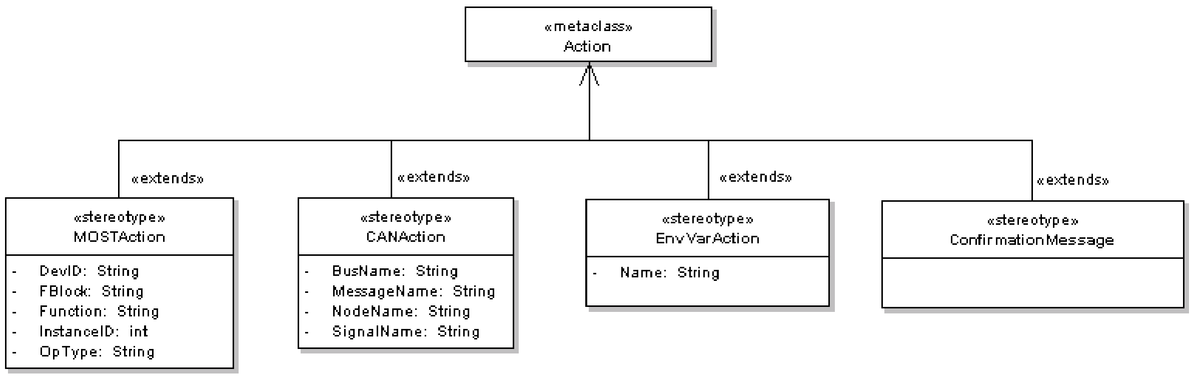

Figure 6: Profile to model car infotainment specific behavioral aspects

Figure 6 shows a part of this profile definition. For the automotive specific bus systems (CAN, MOST) and test specific aspects (EnvironemntVariables, ConfirmationMessage) stereotypes were defined. An application example for the profile is given in figure 7. The activity diagram defines the MOST messages, necessary to realize the user activity *Initialize system*. A more detailed explanation of the allocation mechanism between the functional activities and the product-specific aspects was given in an earlier paper [7].

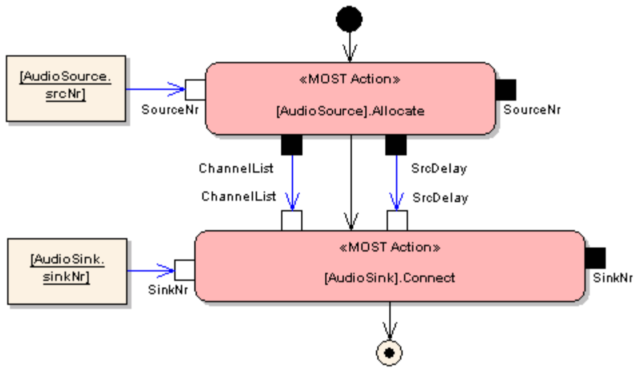

Figure 7: Product specific definition of user activity *Initialize system*

Beyond the dynamic aspects, the product specific part also includes structural information about the system. A top level system structure diagram is given in figure 8. In this diagram the standard notation for the model elements is replaced by alternate images. This simplifies the configuration of the real test environment for the test engineer, because he is able to recognize the devices and necessary connectors from the SysML diagram. The approach of this MOST specific profile for the system structure definition and modeling similar aspects of MOST devices was presented in [8].

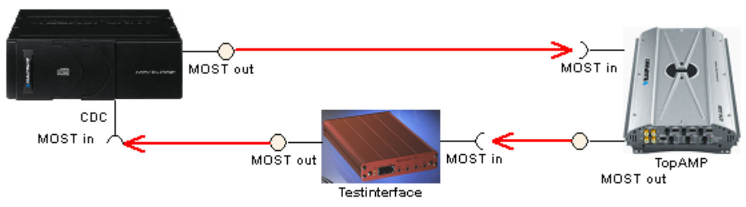

Figure 8: Top level system structure for a MOST system using profiles and alternate images

6 Specification of test input parameters

As you can see in figure 4 some of the functional activities have parameters, e.g. *Set volume*. These parameters are instances of specific value type definitions and contained in a special *value type definitions* package of the respective subsystem. SysML introduces new stereotypes to define unit, dimension and value types. These concepts are applied to specify the behavior activity parameter classifiers. An example of such a parameter definition for the amplifier is given in figure 9. To generate executable test cases from the model, the definition of concrete values for the activity parameters is necessary.

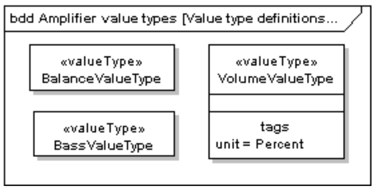
bdd Amplifier value types [Value type definitions...
«valueType» BalanceValueType
«valueType» VolumeValueType
«valueType» BassValueType
tags
unit = Percent

Figure 9: Value type definition for amplifier

For testing and specification purposes it is required to partition these values in equivalence classes. For all values of such an equivalence class as test input, the system reacts in the same way. In model based testing the Classification Tree Method [9] and the Classification Tree Editor (CTE) [10] is a widely used tool to partition test input data. The Classification Tree Editor allows for the definition and editing of equivalence classes, respectively data partitions in a simple and concise way. Thus a classification tree is derived from the SysML model, especially from the value type definitions. The data exchange between the SysML model and the CTE is easy to realize, because the CTE data format is XML.

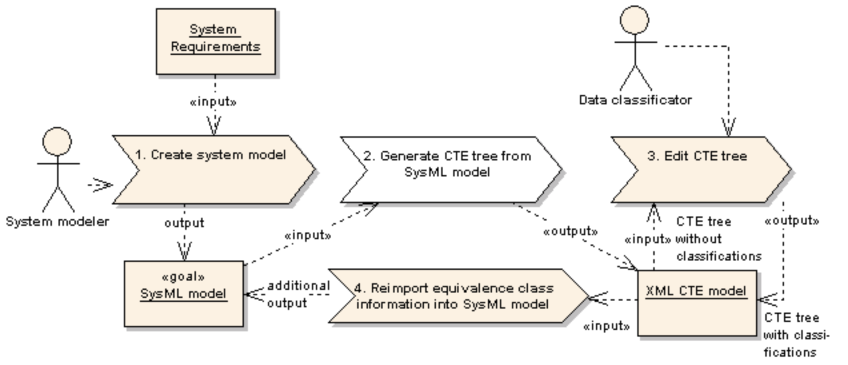

Figure 10: Workflow for combining SysML and CTE

Figure 10 illustrates the process workflow of the SysML and CTE combination. After generating the CTE tree the equivalence classes are added and the specified values are reimported to the SysML model.

Beyond usage of the data partitions as input for the test case generator, another idea is to use the *SysML Allocation* mechanism to map data partitions to model elements of the behavioral model. Allocations are introduced with SysML and allow mapping of elements within the various structures or hierarchies of a model [2, p.123].

For every data partition the system behavior, respectively the system reaction is different. As the system reaction is modeled as activity diagram, it is possible to model the different behaviors within this diagram. Depending on what kind of input parameter in the trigger action is used, a different control and object flow in the activity diagram is executed. The allocation between the input value data partition and the control-, resp. object flow is done using the SysML allocation mechanism. Thus the system behavior for input values of different equivalence classes can be modeled explicitly.

Among deriving test cases to test the normal system behavior, a test case generator is now able to derive robustness tests also, by analyzing the equivalence class allocations.

7 Test execution and test derivation

7.1 Generating product specific test modules for Vector CANoe

The comprehensive information in the SysML specification model provides a derivation of executable test cases to test the car infotainment system. At Robert Bosch the test cases are executed using the simulation and test tool CANoe.MOST (CANoe) [11] from Vector Informatics. CANoe is widely used in the automotive industry for simulation and test during the development of automotive electronic components. It supports the simulation of different automotive specific bus systems, like CAN, MOST or FlexRay. Since version 5 CANoe is able to execute test cases described in an XML format. For that purpose the tester can use - currently 14 - predefined, so-called *test patterns* to trigger and check the state of the system under test. Parts of such an XML test module are given in figure 11. This test module was generated from the functional test case introduced in section 4 and can be executed using the CANoe application.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<testmodule title="testmodule" version="Mon Dec 11 23:30:32 CET 2006">
  <description />
  <engineer>
    <info>
      <name>Oliver Alt</name>
      <description>CM-DI/EAA3</decription>
    </info>
  </engineer>
  <testcase title="Start playback and set volume" ident="1165875911160281">
    <description />

    ...

    <initialize title="Start playback - CDC.AudioDiskPlayer.DeckStatus.Set(Play)" wait="0"
        <mostmsg fblock="0x31" function="0x200" optype="0x0" instance="0x2"
              name="CDC" tellen="1" dir="tx" sa="0x100" da="0x101" ams="ams" spy="node">
            <byte pos="0">0x01</byte>
        </mostmsg>
    </initialize>

    <testerconfirmation title="Start playback - validate" wait="0" passedbutton="yes"
                        timeoutresult="failed">
        Start playback ok ?
    </testerconfirmation>

    ...

  </testcase>
</testmodule>
```

Figure 11: Generated XML test module for execution with CANoe.MOST

Two test patterns are used here to test the example system. To do the trigger operation the *initialize*-pattern is used. This pattern is able to set values or send bus messages to the system under test.

The validation actions for the half-automatic tests are implemented using the *tester confirmation*-pattern. This pattern shows a dialog box with text and asks the user for a Yes/No decision.

7.2 Derivation of functional test cases

Concepts to derive functional test cases from the SysML model are currently still under development, but I will give a short overview of the main idea.

In the work of Belli et. al. [12] an approach for deriving test cases from a, so-called Event-Sequence-Graph (ESG) is presented. The idea is now to generate such ESGs from the SysML model automatically and use the developed algorithms for ESGs to derive the test information.

For that purpose the user and system activities are assigned using activity modeling (events, decision nodes, value specification actions and the new SysML control operator). An execution of a user activity can influence the system activity in four ways:

1. No change.
2. Change only the run state of the system activity (*control operator*).
3. Change only the internal state of the system activity (object state change).
4. Change both states.

With additional modeling of preconditions for the user activity execution, e.g. *You must first insert a CD and than you can start playback*, using decision nodes, a generator should be able to derive the desired event sequences by using the following algorithm:

A simulator tool simulates the execution of the activities included in the system model. The simulator tests for every user activity, if an execution is possible and if the execution leads to a state change in the system activities, respectively in the system. In that case the user activity is added to the activity sequence chain. This step is repeated with the possible predecessors, and so on. If no predeceasing activity can be found, the activity sequence ends. The resulting ESG is the cut of all found activity sequences.

Such a simulator tool is currently under development to be able to evaluate the approach on a real example.

8 Conclusion and Outlook

In this paper an overview about the developed concepts in model-based testing for car infotainment systems was given. By replacing the informal specifications in the development process by a more formal model, the test process and test coverage can be improved by using model-based testing. As modeling language SysML extended with further domain specific profiles was used to model the system specification.

Many of the new concepts from the SysML language were used and adapted to achieve the goal of test process and coverage improvement. The *parametric constraints* were adapted to map parameters between the functional and product specific part, presented earlier in [7]. *Allocations* are used to map value types to the behavior model and the *control operator* is used in the allocation of user and system activities.

With separation of the functional and product specific system aspects in the model the reusability of the test cases could be improved. Thereby activity modeling was chosen for describing the behavior of the system. Derivation of test cases from these functional activity model is currently developed and evaluated. The model is used there as a test oracle.

Both, new approaches are developed, like the profiles for car infotainment systems and the separation of the behavior information, and well known concepts for model-based testing are reused, like the classification tree editor for specifying equivalence classes of test input data or the Event-Sequence-Graph approach for test case derivation. For better user acceptance the concepts are implemented with standard tools like Enterprise Architect, CANoe or CTE.

The approach of test case reusablitiy improvement is currently evaluated in a prototype project. The next step will be the integration of functional test case derivation, after finishing the evaluation of the approach to generate Event-Sequence-Graphs from the SysML model.

References

[1] MOST Cooperation. *MOST Specification Rev. 2.3 08/2004*. MOST Cooperation, rev. 2.3 edition, August 2004.

[2] OMG. OMG Systems Modeling Language (OMG SysML^{TM}) - Final AdoptedSpecification. ptc/06-05-04, Object Management Group, 5 2006.

[3] Mario Jeckle, Chris Rupp, Jürgen Hahn, Barbara Zengler, and StefandQueins. *UML 2 glasklar*. Hanser, 2004.

[4] Sparx Systems Pty Ltd. *Enterprise Architect 6.x*, 2006. http://www.sparxsystems.com, zuletzt besucht am 22.2.07.

[5] Joaquin Miller and Jishnu Mukerji. MDA Guide Version 1.0.1. Technical Report omg/2003-06-01, OMG, Juni 2003.

[6] Peter Hruschka and Chris Rupp. *Agile Softwareentwicklung für embedded real time systems mit der UML*. Hanser, 2002.

[7] O. Alt. Generierung von Systemtestfällen für Car Multimedia Systeme aus domänenspezifischenUML Modellen. In C. Hochberger and R. Liskowsky, editors, *INFORMATIK 2006 Informatik für Menschen Band 2*. Lecture Notes in Informatics, 10 2006.

[8] Oliver Alt and Markus Schmidt. Derivation of System Integration Tests from Design Models in UML. In Hajo Eichler and Tom Ritter, editors, *Proceedings of the ECMDA Workshop on Integration of Model Driven Developmentand Model Driven Testing*. IRB Verlag, 7 2006.

[9] M. Grochtmann and K. Grimm. Classification Trees for Partition Testing. Software testing, Verification and Reliability, 3 1993.

[10] Mirko Conrad. *Modell-basierter Test eingebetteter Software im Automobil: Auswahl undBeschreibung von Testszenarien*, volume 1. Auflage. Deutscher Universitäts-Verlag, Oktober 2004.

[11] Vector Informatik. CANoe.MOST Version 5.2 - Simulation- and testtool for CAN and MOST systems,`www.vector-informatik.de`, 2005.

[12] Fevzi Belli, Nimal Nissanke, Christof J. Budnik, and Aditya Mathur. Test Generation Using Event Sequence Graphs. Technical reports and working papers, Universität Paderborn, `http://adt.upb.de`, 9 2005.