

# Derivation of System Integration Tests from Design Models in UML

Oliver Alt<sup>1</sup> and Markus Schmidt<sup>2</sup>

<sup>1</sup> Robert-Bosch GmbH, CM-DI/ESI2  
Daimlerstr. 6

D-71226 Leonberg, Germany  
oliver.alt2@de.bosch.com

<sup>2</sup> Real-Time Systems Lab  
Darmstadt University of Technology  
D-64283 Darmstadt, Germany  
markus.schmidt@es.tu-darmstadt.de \*

**Abstract.** As model-driven software development is increasingly used, techniques to derive additional information from design models are rapidly explored. One kind of these additional information is test information. In this paper we present an approach of deriving a certain kind of test information for system integration test from a given design model. For a given domain with a generic behavioral model our approach has the ability to generate system integration tests that are specific for any design model of the same domain. The system integration tests are generated as sequence diagrams, where every diagram represents one specific test case for a concrete design model. The approach is of general use since all models are UML 2 models. We illustrate our approach using a case study for a simplified MOST audio system.

*Keywords:* System Integration Test, UML 2, Profiles, MOST

## 1 Introduction

As software systems became larger and more complex, some abstraction in form of models were used to cope with the complexity. Even these abstract description became too complex to include information for all aspects of a system. Non-functional information (e.g. dependability or testing) are typically not part of a system model. This is critical since testing is an important part of software development. Studies indicate that at least fifty percent of the overall costs of software development are devoted to testing [1]. As it is nearly impossible to test a complex system completely, it would be a great benefit if some tests could be generated from the system model itself. In the case of model-driven engineering (MDE) this means derivation of testing information from the developed system model.

---

\* Work supported in part by the European Community's Human Potential Programme under contract HPRN-CT-2002-00275, SegraVis

Since it is nearly impossible to develop an approach that covers all modeling languages, we focus our approach on the widely used UML [2]. Scenario-based techniques are frequently used in preliminary work on UML-based test synthesis. In [3] Pickin et.al. presented a method to generate system tests from test scenarios and design models. They need two external tools (UMLAUT and TGV) for the whole synthesis.

Another approach to derive integration tests was proposed by Hartmann et.al. [4]. They use communicating statechart to describe the interaction between system components.

The remainder of this paper is structured as follows. A simple MOST (Media Oriented System Transport) audio system that serves as our domain example is introduced in section 2. In section 3, we describe our approach to generate specific tests for integration test purposes. Generic behavioral models for MOST components are presented in section 4. Section 5 exemplifies the derivation of test sequences for our domain example. Section 6 concludes the paper with a discussion of further directions.

## 2 Domain

In modern cars the multimedia functions like driver entertainment, navigation and telematics are realized as a distributed system of electronic control units (ECUs). In the domain of car multimedia the ECUs are often connected via the MOST network. The complete car multimedia system in modern cars consists of a dozen ECUs from different vendors with thousands of syntactically correct communication messages. To verify these complex systems new model-based development and testing techniques are developed and evaluated for the development process of OEMs and suppliers like Robert-Bosch.

### 2.1 MOST

Media Oriented System Transport [5] is an object oriented network system, designed to transport multimedia data and control messages to control the connected devices. The device functions are organized as *function blocks*. A hardware MOST device can host one or more function blocks, e.g. a hardware device hosts function blocks for radio, CD player, amplifier, and navigation. But it is also possible to distribute these function blocks over many hardware devices. The

`DeviceID.FBlockID.InstanceID.FunctionID.OpType.Length(Data)`

**Fig. 1.** Definition of the MOST control message format

transport medium for MOST is a fiber optical cable and the topology is a ring. One device is the timing master, the other devices are slaves. The definition for a MOST message is given on figure 1. By using the specified operation types

(OpType) like `Set`, `Get`, `Start` or `StartResult` a controlling device is able to modify or request the state of functions in a function block. With the exception of `Set` and `Start` all request messages are responded by an associated response message. The syntax definition for the messages is given in the so-called *function catalog*. Examples of these function catalogs are [6] and [7].

## 2.2 UML 2 Profile for MOST audio system

Profiles offer the possibility to adapt UML to a certain domain. The primary extension mechanism is the stereotype which is a limited kind of a metaclass.

A simple part of the whole MOST specification is presented as a profile in figure 2. This profile contains hardware and software constructs, where every MOST construct is mapped to a stereotype of the same name. Every stereotype extends the metaclass `Classifier`, since we will model these MOST constructs as different UML model elements (e.g. components or objects). The stereotypes form a hierarchy both for hardware and software components. Every hardware construct has a device identifier `devID`, where there exists a specific stereotype for a MOST master device. The stereotypes for MOST software constructs form a hierarchy with the most general `FunctionBlock` as root and some specific audio function blocks (`AudioDiskPlayer`, `AmFmTuner`, `AudioAmplifier`) as leaves. An additional level separates audio sources from audio sinks.

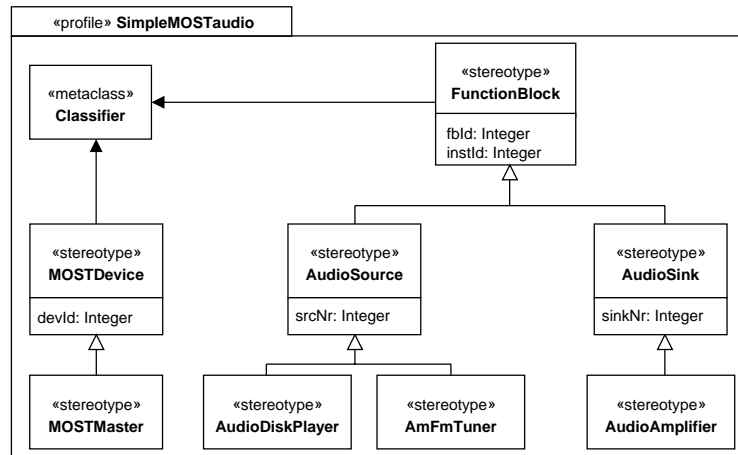


Fig. 2. Profile for a simple MOST audio system

Furthermore, the stereotype `FunctionBlock` contains attributes to specify the function block identifier (`fbId`) and the specific instance identifier (`instId`) of a certain MOST function block. As every MOST audio source has its own source number the associated stereotype contains the attribute `srcNr` to specify this number. A similar attribute `sinkNr` specifies the number of a audio sink.

### 2.3 An example MOST system

An example of a MOST system with three source function blocks and one sink function block, distributed on two hardware devices is given on figure 3. We apply here the stereotype definitions from our MOST-specific profile. The components are the physical MOST hardware ECUs including instances of function blocks. The connections between the component ports model the physical connections via the fiber optical cable. In the device `PlayerUnit` we have two similar function blocks of type `AudioDiskPlayer`. Therefore, they need different instance ids. The controlling device is the system master and controls the two slave devices `PlayerUnit` and `HifiAmplifier`.

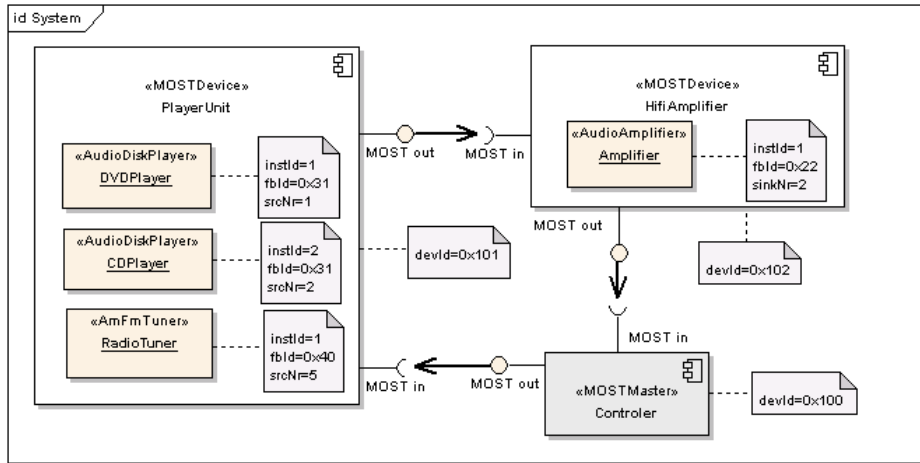


Fig. 3. Structure of example MOST system

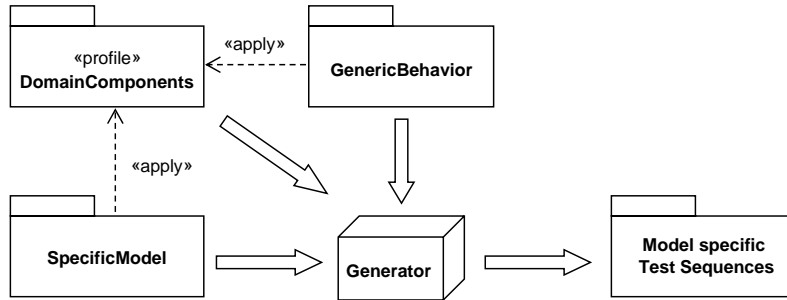
## 3 Approach

We chose UML 2 [2] as our modeling language since it is an open standard and has thirteen different diagram types for both structure and behavior. An important benefit is the built-in extension mechanism of UML - the profiles. This offers the possibility to tailor UML models to domain specific needs. In addition to this application we use profiles to map generic behavioral models to a concrete design model.

### 3.1 Derivation of test sequences

Developing domain specific models in UML is done using a specific profile. As it is shown in figure 4 the design model `SpecificModel` uses the profile

`DomainComponents` that contains domain specific information. This profile is reused to describe the general behavior of components from the given domain. Both packages `DomainComponents` and `SpecificModel` are specific for a certain domain but not for a concrete model of this domain. As such they can be used for any concrete model of the same domain.



**Fig. 4.** General approach to derive specific test information

The generator uses these three models to create integration tests for the specific system. This is done using the stereotypes as links between the design model and the generic behavior. The design model contains stereotypes with concrete attribute values. These values are used to create the concrete behavior from the given generic behavioral model. This concrete behavior model is further transformed into a set of sequence diagrams where each diagram represents a single test case. All sequence diagrams together represent specific system integration test sequences.

## 4 Behavior

To describe the dynamic behavior of our system, especially the MOST function blocks, we can use the UML 2 behavioral diagrams such as statecharts, activity or sequence diagrams. In the domain of MOST typically Message Sequence Charts<sup>3</sup> are used to specify and describe the system behavior [8]. So we chose UML sequence diagrams to specify the MOST communication behavior.

Table 1 gives an overview of the MOST functions used in our examples in the next sections.

### 4.1 Generic (MOST-)behavior

To define the behavior in a generic way, we replace specific diagram information by wild-cards in square brackets ([ ]), corresponding to our profile definitions.

<sup>3</sup> telecommunication standard (ITU-T Z.120), nearly identical with UML 2 sequence diagrams

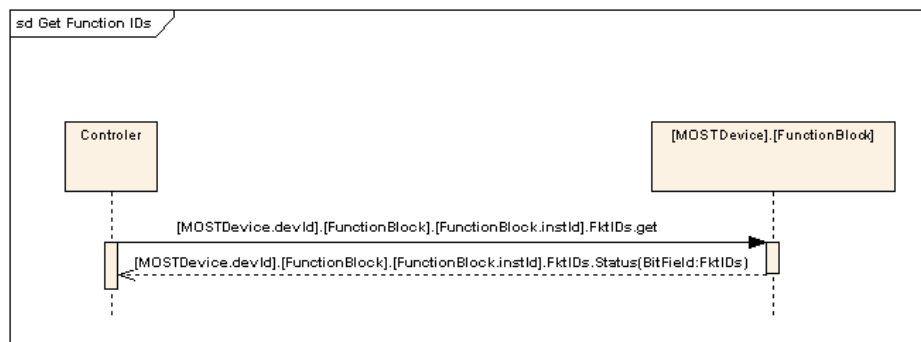
FunctionBlock	Function.OpType	Data	Meaning
All	FktIds.Get	-	Returns list of function Ids for function block
AudioSource	Allocate.StartResult	SrcNr	Reserve audio channel for source
AudioSink	Connect.StartResult	ChannelList, SrcDelay, SinkNr	Connect sink to audio channel
AudioSource	SourceActivity.StartResult	SrcNr, Activity=On	Activate audio source
AudioDiskPlayer	DeckStatus.Set	Play	Start playback
AudioDiskPlayer	DeckStatus.Set	Stop	Stop playback

**Table 1.** MOST functions used in our example

To derive a concrete behavioral scenario we must just replace the generic notation by the concrete values, defined in the system design model.

The generic sequence diagram to request the function Id list for a certain function block is given in figure 5. A MOST master device may use this functions - available in all function blocks - to scan the network and get the available functions in the devices and their function blocks (plug and play functionality). Following the MOST specification we use the message notation introduced in figure 1 to model the communication flow in the sequence diagram.

You can see the application of our generic notation, where e.g. [FunctionBlock.instId] references the attribute instId of the stereotype FunctionBlock.



**Fig. 5.** Get function IDs as sequence diagram

## 4.2 Generic source-sink behavior

To transmit audio data between source and a sink, the source at first has to allocate a channel on the MOST and then the sink can connect to this channel to receive the data.

The generic behavior sequence for building such an audio connection is given in figure 6. In the example sequence diagram we have interactions between two different function blocks and the controller.

Sources and sinks are special function blocks and thus they are a refinement, corresponding to the generalization in our profile. In the sequence diagram we use therefore the wild-cards `[AudioSource]` and `[AudioSink]` as replacement of the specific MOST parameters.

A more complex scenario is the change of the source with more than two involved function blocks (at least two sources and one sink). In such a case we add a numeric value to the generic identifiers (e.g. `[AudioSource1]`). For lack of space we can not present this example here.

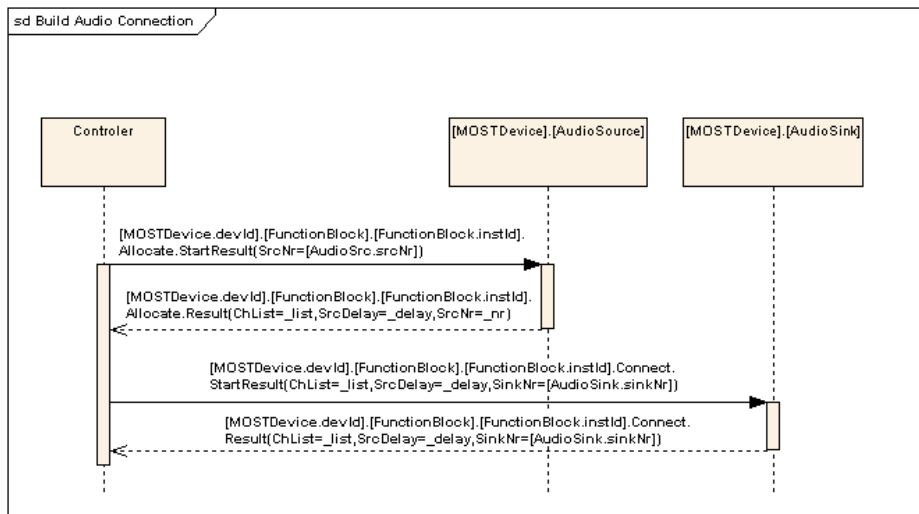
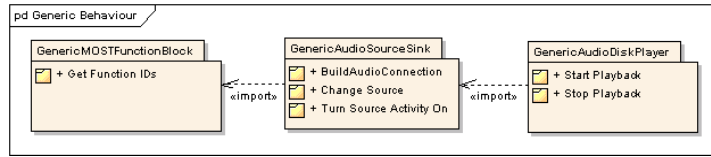


Fig. 6. Generic behavior for building an audio connection between source and sink

## 4.3 Generic cd-player behavior

As further refinement we define the generic behavior of special sources and sinks. Such a source for example may be an audio disc player with sequences defining the control messages to start and stop playback. The generic definition is done in a similar way as described for sources and sinks, but in this case we use wild-cards corresponding to our refinement of `AudioSource` and `AudioSink` in our profile (e.g. `[AudioDiskPlayer]`).

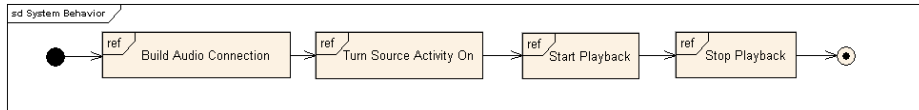


**Fig. 7.** Structure of behavioral packages

The relation between our behavioral packages is given in figure 7 and reflects the hierarchy of common behavior of MOST function blocks from our example. Due to of space we have omitted a specific audio sink (e.g. amplifier).

## 5 Deriving test sequences

To derive system test sequences we need additional behavioral information. We have to define the order of execution for our generic sequences. This could be done using UML interaction overview diagrams, similar to the use of activity diagrams described in [9].



**Fig. 8.** System behavior sequence

The interaction sequence for our example system is given in figure 8. To keep it simple we show only a part of the whole system description.

With all these information we are now able to generate a permutation of sequences for our example system. From our system model and the profile definition we get all necessary informations for replacing the generic wild-cards with the system specific data. A test sequence that is automatically generated from the activity diagram in figure 8 and the design model in figure 3 is shown in figure 9. All generic information is replaced by specific information from the design model.

Sequence diagrams are only one possible view of test sequences. To get an automated execution of the test sequences we can generate e.g. XML for the commercial CANoe.MOST tool from Vector Informatik [10] or TTCN-3 [11] code and use the test-environment for MOST systems as described in [12] and [13] to execute the tests.

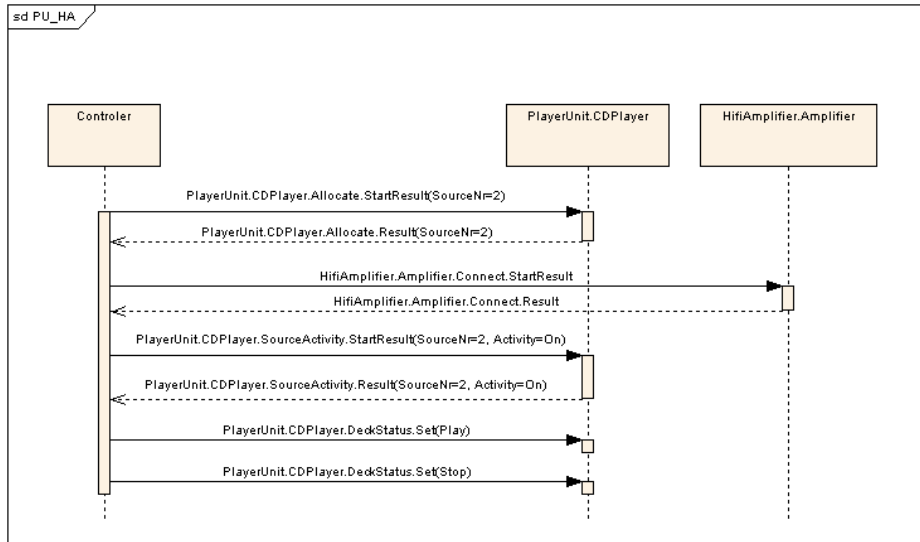


Fig. 9. Test sequence with PlayerUnit.CDPlayer and HifiAmplifier.Amplifier

## 6 Conclusions and Outlook

In this paper, we introduced an approach to automatically derive test information for any design model of a certain domain. Starting from a given domain with a generic behavioral model, our approach has the ability to generate specific system integration tests for any design model. These integration tests are depicted as a set of sequence diagrams. Every sequence diagram represents a specific test case for a given design model. The description of a test case as a sequence diagram is an approved method. But instead of manually writing a new sequence diagram for any possible combination from the system model, we generate these concrete sequence diagrams.

To be as general as possible, we chose UML 2 as our modeling language. Therefore, our approach can be used with any UML 2 compliant modeling tool. The approach uses the following groups of models.

1. A profile containing stereotypes that specify certain elements of the domain.
2. A behavioral model that describes the behavior of domain components in a generic manner.
3. A specific system model that uses the profile to specify system components.

Models of the first two groups are specific for a certain domain (as in our example for a simplified MOST audio system) and can be reused for every concrete model of the third group.

The fundamental basis of our approach is the extension mechanism of UML2-the profiles. These stereotypes are used as links between elements of a design

model and elements of the generic behavioral model. Our approach can be used with every UML 2 compliant tool. Only one additional plug-in is necessary.

We are currently implementing the described approach. As such we are working on an implementation of the generator as a plug-in for the UML 2 tool Enterprise Architect [14]. After the completion of this plug-in we will evaluate our approach with more realistic examples. As such we have to extend the MOST profile to include more MOST devices both in the design models and the behavioral models and to model extended test oracle information. Another research direction is the description of behavior with activity diagrams instead of sequence diagrams. Activity diagrams offer constructs to specify behavior in a more concise way (e.g. explicit modeling of object flow), since it is possible to describe a more general behavior beyond plain communication.

## References

1. Harrold, M.J.: Testing: a roadmap. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, ACM Press (2000)
2. OMG: UML 2.1 Superstructure Specification. OMG. (2006) <http://www.omg.org/docs/ptc/06-01-02.pdf>.
3. Pickin, S., Jard, C., Traon, Y.L., Jéron, T., Jézéquel, J.M., Guennec, A.L.: System test synthesis from uml models of distributed software. In: FORTE '02, London, UK, Springer-Verlag (2002)
4. Hartmann, J., Imoberdorf, C., Meisinger, M.: Uml-based integration testing. In: ISSSTA '00: Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis, New York, NY, USA, ACM Press (2000) 60–70
5. MOST Cooperation: MOST Specification Rev. 2.3 08/2004. MOST Cooperation. Rev. 2.3 edn. (2004)
6. MOST Cooperation: MOST FunctionBlock AudioDiskPlayer. MOST Cooperation. Rev. 2.4 edn. (2003)
7. MOST Cooperation: MOST FunctionBlock Audioamplifier. MOST Cooperation. (2003)
8. MOST Cooperation: MOST Dynamic Specification. MOST Cooperation. 2.1 edn. (2005)
9. Briand, L., Labiche, Y., eds.: A UML-Based Approach to System Testing, Software Quality Engineering Laboratory Carleton University, Carleton University (2002)
10. Vector Informatik: CANoe.MOST Version 5.2 - Simulation- and testtool for CAN and MOST systems, [www.vector-informatik.de](http://www.vector-informatik.de) (2005)
11. ETSI: The Testing and Test Control Notation version 3 Part 1: TTCN-3 Core Language. ETSI. Etsi es 201 873-1 v2.2.1 edn. (2003)
12. Burton, S., Baresel, A., Schieferdecker, I., eds.: Automated testing of automotive telematics systems using TTCN-3, 3rd Workshop on system testing and validation, Fraunhofer IRB Verlag (2004)
13. Schmidt, M., Herrmann, J., Baresel, A., eds.: Domain Specific Test Systems Automotive Case Study Description, 3rd release, TT-MEDAL Consortium (2005)
14. Sparx Systems Pty Ltd.: Enterprise Architect 6.0. (2006) <http://www.sparxsystems.com>.