Transforming Message Sequence Charts for Testing

Automated Test Code Generation for Different Abstraction Layers

Oscar Slotosch
Validas AG
slotosch@validas.de

Abstract: In this paper we describe experiences gained from using Message Sequence Charts (MSCs) for the specification and execution of tests. Using a test code generator (MSC2C) allows us to verify a system under test against the specified sequences. For executing one test specification on different test stages with different interfaces we applied transformation rules that transform abstract MSCs to concrete MSCs for different test stages. The rules are also formulated as MSCs. We demonstrate the approach using aspects from the AUTOSAR network management function.

1 Introduction

While many tools for model based development tools (including code generators) exists [Sch03], the model based testing approach is not so elaborated, even if there exists test generators that generate sequences from development models for the purpose of testing (see [Ha02]). In our industrial projects we formalized the requirements using the international standard of Message Sequence Charts [ITU04] and show how the MSCs can be reused for testing on different test stages.

The concretizations from the abstract MSCs to the test specifications or the different test stages are modelled using MSC transformation rules. MSCs are used as a formalism to express the MSC transformations.

2 Development Process

The typical process for embedded systems is a V-like process. Mostly all tests are manually created (e.g. by executing the tests manually, or by programming code for executing the tests). We suggest generating all test specifications, which dramatically reduces the test effort. Instead of programming a test suite for each test stage (software test, system test, acceptance test), we use an abstract test specification in form of Message Sequence Diagrams (MSCs) that captures the functional requirements. From this abstract MSC the test specifications for the test stages are generated using MSC transformations. The transformations correspond to the different phases and levels of abstractions in the typi-

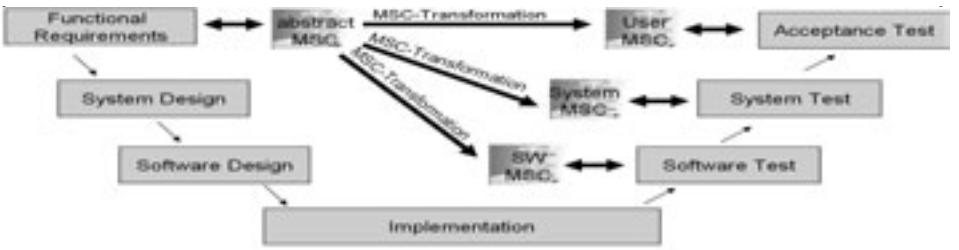

Figure 0: Development Process

cal embedded software design process (see Figure 1). We use the MSC2C test code generator for the automation of the tests (see [OS07]).

3 Example

In automotive applications with many electronic control units (ECUs) and devices it is important to coordinate startup and shutdown of the bus communication. This specific task is achieved by the network management (NM) module which deployed on each device connected to the network. The AUTOSAR project [Au07] standardized the functions of the network management (NM), which have been used as objective for the described test method.

One important requirement of the NM is: "After initialization the NM has to ensure that the devices after waking up remain active for a specified time". The NM is realized as a module of the operation system, which provides an API and functions for the NM. After integration in different ECU there are different scenarios for waking up the ECU, e.g. with an impulse at a digital input at the so-called "wakeup-line", or by sending a wakeup message at a bus. Using different bus technologies (CAN, Flexray) within one system makes testing more complex. Figure 2 shows a description of this requirement using MSCs.

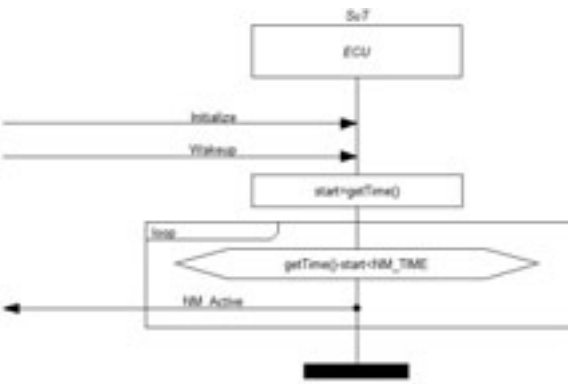

Figure 2: Abstract NM functional specification

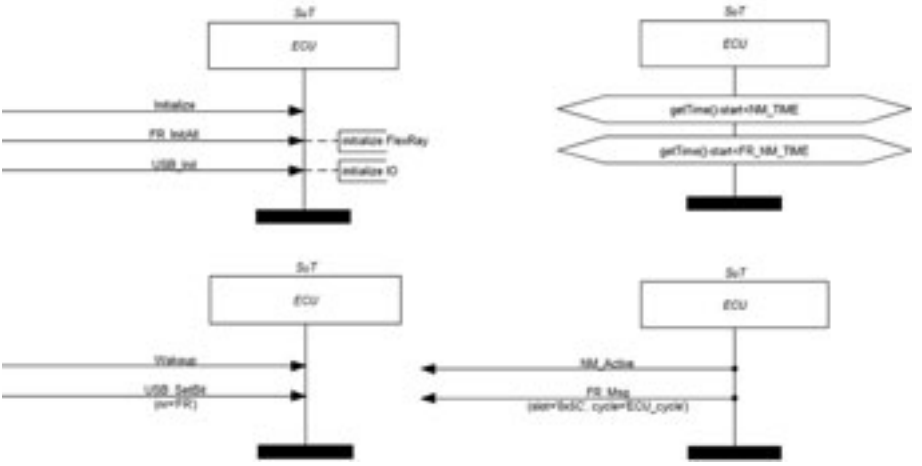

Figure 3: Transformation Rules for Flexray

To transform the abstract tests into executable test specification for the different test stages we used transformation rules, formalized by via pairs of MSCs. The transformation rule is applied to all elements in the MSC that match the first element in the rule. The matching element in the MSC is replaced by all following elements in the rule.

Figure 3 shows all rules that are required for transforming the abstract test specification in Figure 2 into an executable test specification. They represent the design decisions for implementing the NM on Flexray busses: the initialization prepares the Flexray bus and the USB IO card, which is used for triggering the wakeup line. This is achieved by replacing the Initialize message with two specific initialization messages. The active signal of the NM is a Flexray messages. Every ECU has a dedicated cycle for sending its NM message. This is represented by replacing the message NM_Active with a Flexray mes-

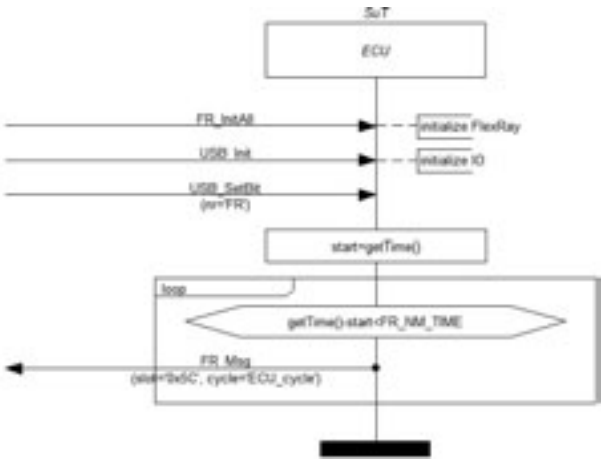

Figure 4: NM Test specification for the NM

sage sent in the slot 0x52 of the ECU specific cycle, which is here modeled using the parameter ECU_cycle.

With a small extension of the MSC2C test code generator [OS07] we could transform the abstract MSC automatically into the platform-oriented test specification (see Figure 4).

4 Summary

We modeled about 120 requirements of the NM. Most of them were software requirements, which have been tested directly at the software module. Further requirements, regarding timing, interaction and messages formats have been tested with different CAN and Flexray busses. We discovered many deviations in earlier versions of the NM.

One could argue, that transformation would not be necessary, if the interfaces of the system under test are identical for all tests. However, as different abstractions of the interfaces of the system under development are used at different design (and test phases), this is not practicable. Furthermore the functional specification uses an abstract and understandable interface with symbolic signals (e.g. NM_Active), which is easy to validate against the requirements, rather then their actual implementation (e.g., 0x52).

In general the approach saves much effort, since only one specification has to be created and not a specification for every test stage. However not all requirements have to be executed at every test stage (mostly due to an internal interface that cannot be stimulated). Therefore the MSC transformation is able to eliminate all MSCs that are not transformed. The basis for the approach is the ability to execute MSC test specifications by generating C code from it (see [OS07]).

Finally, the approach is very efficient in practical application since it sticks to the single source principle. For example, if a new requirement is added, it is added only to the abstract specification. By generating the other specifications automatically, the specifications of the test cases at different levels of abstraction are kept consistent.

Literature

[Ha02] AGEDIS: Model Based Test Generation Tools, Alan Hartman, see http://www.agedis.de/documents/ModelBasedTestGenerationTools.pdf, 2002.

[Au07] Automotive Open System Architecture, http://www.autosar.org

[Sch03] CASE Tools for Embedded Systems, Bernhard Schätz, Tobias Hain,Wolfgang Prenninger, Martin Rappl, Jan Romberg, Oscar Slotosch, Martin Strecker, Alexander Wisspeintner, et.al. Technical Report TUMI-0309, Fakultät für Informatik, TU München, 2003

[ITU04] Message Sequence Chart (MSC), International Telecommunication Number Z.120, http://www.itu.int/ITU-T/studygroups/com17/languages/Z120.pdf, 2004.

[OS07] Grzegorz Olender, Oscar Slotosch: Automatic Testing with Message Sequence Charts (MSCs), CONQUEST 2007