TEST CASE DESIGN FOR THE VALIDATION OF COMPONENT-BASED EMBEDDED SYSTEMS

Wolfgang Fleisch

http://www.ias.uni-stuttgart.de

The validation of functional and real-time requirements of control software for embedded systems is a difficult task. It usually needs the electronic control unit (ECU) and the controlled hardware components. But very often the ECU or hardware components are not available for testing the control software at the beginning of the development. This paper presents how test cases can be designed from use cases and how component-based control software can be validated without ECU and hardware components by simulating the test cases in early development phases. For achieving a tool-based testable format, extended UML sequence diagrams are applied to formalise sequences of events, which have been specified in the use case scenarios. Provided that black box components are used for developing component-based applications, the monitoring of the dynamic behaviour inside the components is not possible during simulation. But the simulated dynamic behaviour is observable on the connections between the software components. In such a way monitored and recorded time stamp events are finally compared offline against the expected sequences of events specified in the test cases. The offline comparison validates the simulated behaviour by demonstrating the fulfilment of user requirements and by detecting errors in case of contradictions. An application example of an automotive wiper control system demonstrates the capacity of the presented test case design and validation process.

1. Introduction

1.1. Component-Based Embedded Systems

Embedded systems are control systems which are embedded in a technical system. They are designed for calculating actions as a response to characteristic input values. Usually this task is performed by a microcontroller based electronic control unit (ECU) which communicates with its environment by sensors and actuators. Distributed embedded systems consist of a network of ECUs, which exchange information via a communication network. A typical example for a distributed embedded system is a modern car. Nearly all kinds of control and supervision functionality, i.e. anti-blocking brake system, transmission control, fuel injection and different body electronics features, are controlled by distributed ECUs. It can be

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35409-5_23

B. Kleinjohann (ed.), *Architecture and Design of Distributed Embedded Systems*

presumed that almost the complete ECU hardware is developed by using components off the shelf. In opposite to this fact it is a relatively new approach to design also the control software with predefined software components [Goeh98].

1.2. Component-Based Control Software

Development of component-based control software means that new application software for an embedded system is composed of a set of configurable (with parameters) software components, which have been explicitly developed for multiple usage [Goeh98]. In the following, the term component always denotes a software component. Applying components and a component model promises many advantages as for example reduced development time and increased quality of the composed control software. The higher quality is reached by using prefabricated components, which have been extensively tested during their development. Applying uniform communication and execution mechanisms, provided by a component model, also reduces the number of possible design errors. The components exchange information by exchanging events and data on the connections between their interfaces.

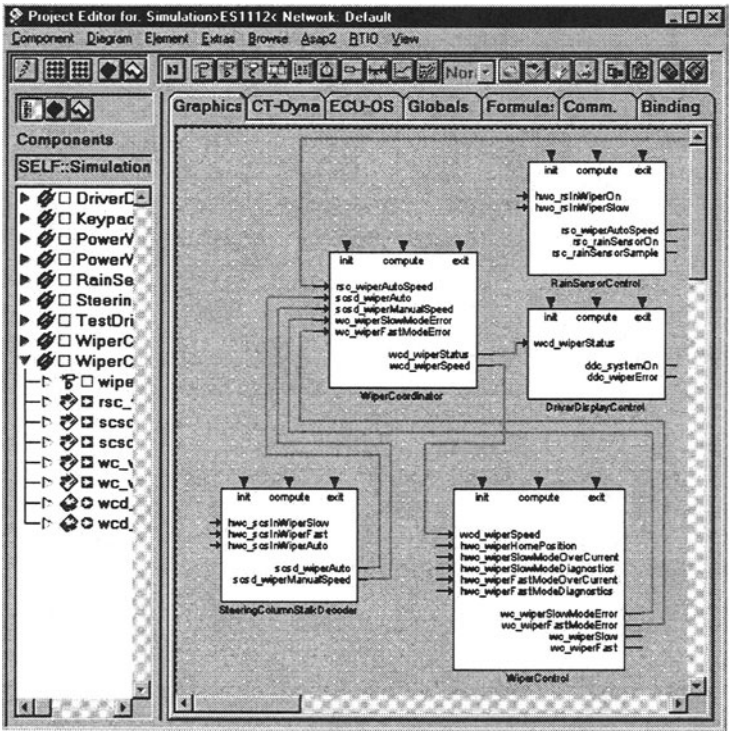

Figure 1: Example for modelling a component-based control software with ASCET-SD

Figure 1 presents an example of a component-based control software, which has been developed with the CASE tool ASCET-SD (Advanced Simulation and Control Engineering Tool – Software Development) from ETAS GmbH [ETAS00]. ASCET-SD supports modelling, simulation and target code generation of component-based automotive control software and has been applied in the context of this research

work. ASCET-SD offers different graphic and textual editors for modelling components and applications. The internal behaviour of components can be specified in different notations, i.e. control block diagrams, state machine diagrams, a Java-based specification language or standard C-code.

1.3. Difficulties in the Validation

Control software for embedded systems is characterised by real-time requirements, distribution and increasing complexity. The validation of functional and real-time requirements of a developed component-based control software is a difficult task [FlRi97], [GuNae99]. The testing of control software for example needs the ECU, the controlled hardware components and special debugging devices for real-time data monitoring. But very often the hardware components, i.e. the car or a part of it, and the ECUs are not available at the beginning of the software development, which leads to a late validation of the developed control software. The late validation results in a delayed detection of design errors in the control software, which causes increased fixing costs and delayed project schedules. For that reason simulation has been selected in this research work to enable the early validation of component-based control software during the early development phases [Flei99], [Flei00].

The quality of prefabricated components is verified by extensive testing during their development. But when composing a new control software with such components the level of correctness can not automatically be considered to be equal to the original single components. When applying such black box components for composing a new control software, the monitoring of the dynamic behaviour during simulation inside the components is supposed to be impossible. But the dynamic behaviour of the component-based control software is observable on the connections between the connected components. For these purpose, especially the exchanged events between the connected components are monitored in the simulation model when executing the simulation of the component-based control software.

2. Early Validation of Component-Based Control Software

The validation process for early development phases, presented in Figure 2, addresses two major goals:

- Demonstration of the fulfilment of user requirements
- Detection of design errors in the component-based control software

Both are checked by an offline comparison of the simulated dynamic behaviour of a simulation model against the specified required behaviour. It has to be noticed that design errors in component-based control software are mainly reduced to composition and configuration errors when using already verified components.

The component-based development and validation process consists mainly of the following 7 steps:

1. Use case analysis for gathering testable requirements from an users point of view
2. Transformation of use case scenarios into extended UML sequence diagrams
3. Design and modelling of the component-based control software and generation of an executable simulation model
4. Definition of stimuli sequences to complete the test cases

5. Simulation of the test cases for recording the communicated events between the components in the simulation model
6. Offline comparison between the 'Simulated Event Sequences' (SES) and the 'Required Event Sequences' (RES) for validation and error detection
7. Correction of the component-based control software or the use cases if a contradiction (possible error) has been detected

The test case design (see steps 1,2 and 4 in Figure 2) is part of the development and validation process and explained in more detail in the following sections.

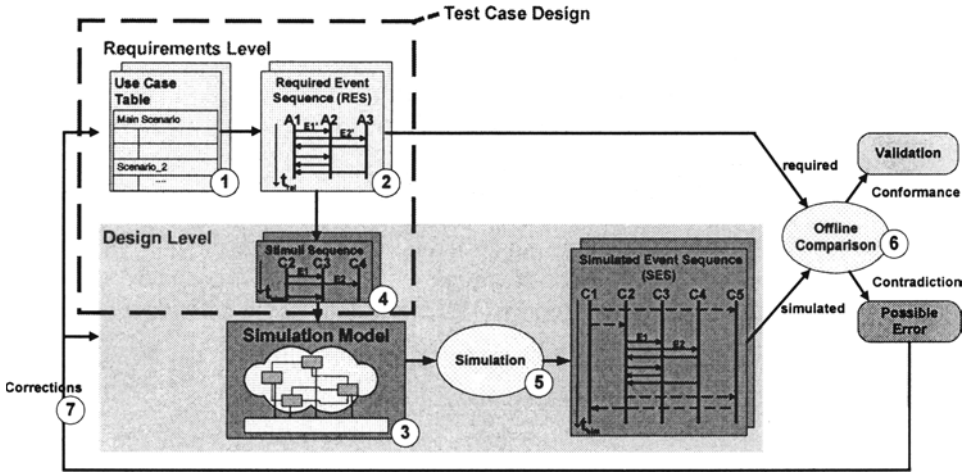

Figure 2: Validation Process

3. Test Case Design

3.1. Test Case

In the context of this work, a single test case consists of a stimuli sequence and a expected output sequence of events, which is denoted as 'Required Event Sequence (RES)'. The stimuli sequence defines a sequence of input events which trigger the simulation model. During simulation the monitored sequence of events is recorded in a 'Simulated Event Sequence' (SES). A test case is validated if the SES is conform with the RES. A test case detects a possible error if a contradiction between SES and RES has been detected. In the following it is described how test cases are designed from use cases.

3.2. Gathering Testable Requirements with Use Cases (Step 1)

According to Jacobson, the originator of use case analysis, "A use case is a specific way of using the system by performing some part of the functionality. Each use case constitutes a complete course of events initiated by an actor and the system. ... The collected use cases specify all the existing ways of using the system" [Jaco92]. The main concepts of use case modelling are actors and use cases. An actor represents an entity (a human being, a machine or a computer system) external to the system under development, that communicates with the system in order to achieve certain goals. A use case is a generalisation of a usage situation where one or many actors interact with the system to accomplish specific goals. One use case may cover several

sequences of events - so called scenarios. A use case may be described either from an external (black-box) point of view or from an internal (white-box) point of view.

This scenario driven modelling view of the use cases has been the main reason for selecting them for describing testable requirements for the dynamic behaviour of a component-based control software. Actors on the requirements level are not equal to components on the design level, but the sequences of events (scenarios) between the actors can be compared against sequences of events between the components in the simulation model, which is generated from the component-based design model. Also an important advantage of use cases in general is their suitability for the software design as well as for the validation. The requirements do not come from the blue but are systematically gathered by writing use cases. Use cases are not qualified to describe requirements completely, but when testing a designed control software against all specified use cases by simulation later on, it can be stated at least that the validation is complete from an user's point of view.

Cockburn [Cock97] has introduced a helpful template for writing use cases. Based on this template we have developed similar use case template, which is presented in Figure 3 [Flei99]. Additionally we have defined some writing rules guiding the developer to write precise use case scenarios [Flei00]. The structured format of a table and the writing rules improve the specification of precise and meaningful requirements.

USE CASE <No>	<name>	
Goal	<>	
Preconditions	<>	
Postconditions	<>	
Actors	<>	
Main scenario	Step	Action
	1.1	<>
	1.2	<>
	1.3	<>
	...	
Scenario 2	Step	Action
	2.1	<>
	....	<>
Variations	<>	
Exceptions	<>	
Notes	<>	

Figure 3: Use case template

3.3. Transformation to Required Event Sequences (Step 2)

Although the use case tables are already in a structured format, they are not suitable for a tool-based automatic testing. For achieving a testable format of the requirements, all use case scenarios are transferred manually into extended UML sequence diagrams, denoted as 'Required Event Sequences' (RES) as presented in Figure 4. Use cases which have been written according to the writing rules, relieve this manual transformation process. The graphical specification of the RES is supported by a self-developed sequence diagram editor. The RES are based on the UML sequence diagram notation [UML00] and have been extended for specifying additional real-time conditions. Timing conditions between two events can be specified relatively as maximum, minimum or interval time.

After specifying the required order and timing of a sequence of events, three additional test case attributes are set for every RES. The attribute *precondition* is taken over from the use case scenario. The attributes *sequence type* (normal, mandatory or forbidden) and *comparison type* (event repetition allowed or not) are set according to the selected test strategy for the offline comparison. If *sequence type* normal is selected, the RES is only compared against the respective SES. If *sequence type* mandatory or forbidden is selected, the RES is compared against all SES if the sequence of events occurs after the precondition has become true. If a forbidden RES occurs after the precondition has become true, it is an error. Depending on design decisions from the component-based development, components may send the same event periodically. Therefore the *comparison type* defines if a required event in the RES is allowed to occur repeatedly in the recorded SES or not. The attribute *comparison type* has been introduced after first practical experiences with the offline comparison algorithms.

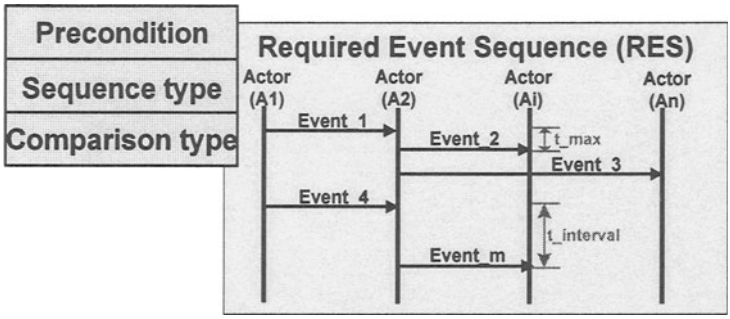

Figure 4: 'Required Event Sequence' (RES)

3.4. Deriving Stimuli Sequences (Step 4)

After the component-based design model has been developed and an executable simulation model has been generated from it, the test cases can be completed. The expected output sequence of events is already defined in a RES. A stimulating input sequence of events has to be derived from the respective RES to complete a test case. The first event of the RES usually becomes the first stimuli event. Alternatively other events which trigger this first event can become stimuli events. Additionally all events in the RES, which are sent from an external actor (i.e. switch or sensor) are also candidates to become a stimuli event in the stimuli sequence.

4. Test Case Simulation

4.1. Preparation and Execution of the Simulation Model (Step 5)

The generated simulation model of the component-based control software has to be prepared for the test case simulation. The 'Test Driver' component as shown in Figure 5 includes the necessary supplements.

The missing hardware components usually have a reactive behaviour, as for example an electrical wiper motor. For that reason a simple environment model of the hardware components is supplemented to the simulation model to simulate the closed loop control behaviour of the overall system. The environment model is one

part of the 'Test Driver'. The other part of the 'Test Driver' contains the stimuli sequences, which inject the stimulating events for the simulation runs of different test cases.

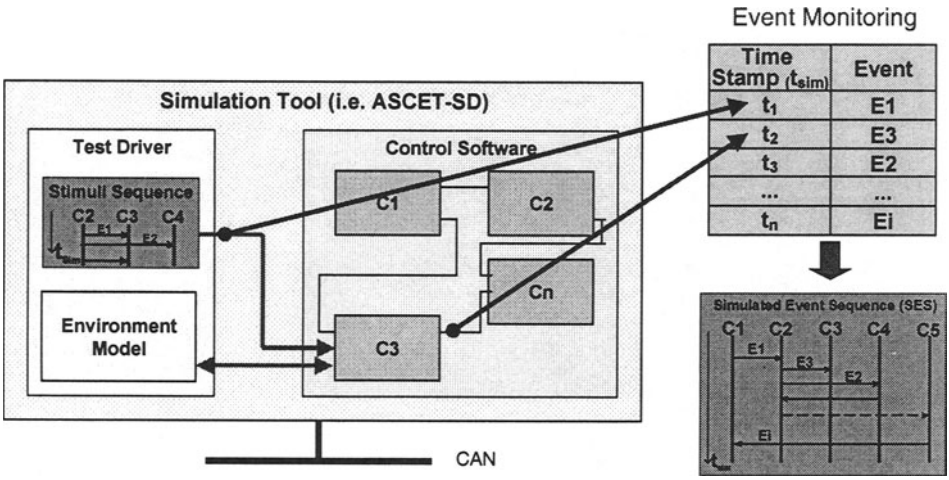

Figure 5: Simulation of test cases involving a single ECU

For the simulation of test cases involving single ECUs, we have selected the tool ASCET-SD for component-based modelling and real-time simulation [ETAS00]. ASCET-SD provides component-based modelling of control software as well as a comfortable simulation environment for injecting stimuli and for recording real-time data during simulation runs. Time stamp events can be reconstructed offline from the recorded real-time data of the different simulation runs.

4.2. Offline Comparison and Corrections (Step 6 + 7)

The offline comparison tests the conformance between the SES and the corresponding RES for every simulated test case. Conformance is achieved if all events in the RES can be found in the respective SES in the right order and if all timing constraints are met. Additionally, each RES of the sequence type *mandatory* or *forbidden* is tested against all SES. A mapping table for the event names between the use case model and the simulation model is necessary to enable a tool-based automatic checking. If the offline comparison detects contradictions, the possible errors in the component-based design model or in the specified use cases have to be investigated by the developer and corrections have to be made. Typical design errors are missing connections between components, wrong parameter values of components or improper dynamic behaviour on the system level.

5. Application Example: Test Case Design and Validation of an Automotive Wiper Control System

The following application example is part of a distributed body electronics system and demonstrates the test case design as well as the capacity of the validation process when modelling and simulating component-based control software using ASCET-SD. The application example is an automotive wiper control system. The

component-based software (see Figure 1) controls the hardware components rain sensor (for rain strength measurement), steering column stalk (for driver's wishes), wiper motor and dashboard display. A photo of the laboratory prototype is presented in Figure 6.

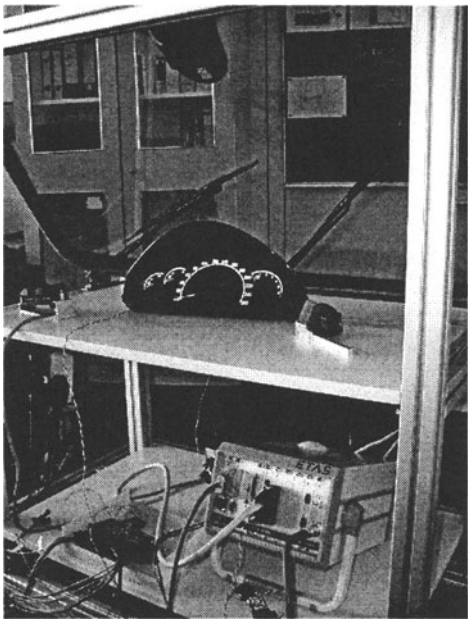

Figure 6: Automotive wiper control system

During the early development phases most of the hardware components have not been available. For that reason the component-based control software has been validated by real-time simulation and offline comparison. Later on, when the real hardware components have been available except the ECU, a hardware in the loop simulation of the component-based control software has been executed. 4 use cases with 16 scenarios have been gathered as initial requirements in the use case analysis as a starting point for the test case design. Figure 7 shows exemplary the main scenario of the use case 'Standard wiping front window'.

USE CASE W.1	Standard wiping front window	
Goal	clean windscreen	
Precondition	a) wiper motor(A3) off (wiper motor park position sensor(A2) in state ‚on') b) wiper motor(A3) on (wiper motor park position sensor(A2) in state ‚off')	
Postcondition	steering column stalk(A1) in position ‚wiper_off', wiper motor park position sensor(A2) is ‚on'	
Actors	driver(A0), steering column stalk(A1), wiper motor park position sensor(A2), wiper motor(A3), wiper control(A4)	
Main Scenario	Step	Action
	1.1	driver(A0) switches steering column stalk(A0) in position ‚wiper_slow' with precondition a).
	1.2	steering column stalk(A0) sends its position ‚wiper_slow' to wiper control(A4).
	1.3	wiper control(A4) starts wiper motor(A3) with velocity ‚slow' after maximum delay of 100 ms.
	1.4	driver(A0) switches steering column stalk(A1) in position ‚wiper_off'.
	1.5	steering column stalk(A1) sends its position ‚wiper_off' to wiper control(A4).
	1.6	wiper motor park position sensor(A2) sends ‚on' to wiper control(A4).
	1.7	wiper control(A4) stops the wiper motor(A3) in park position with velocity ‚off' after maximum delay of 20 ms.

Figure 7: Main scenario of the use case 'Standard wiping front window'

For achieving a testable format, the main scenario of the use case 'Standard wiping front window' is transformed to the RES in Figure 8. The test case attributes of the RES are set as following:

Precondition:	Wiper motor 'off'
Sequence_type:	Normal
Comparison_type:	Event repetition allowed

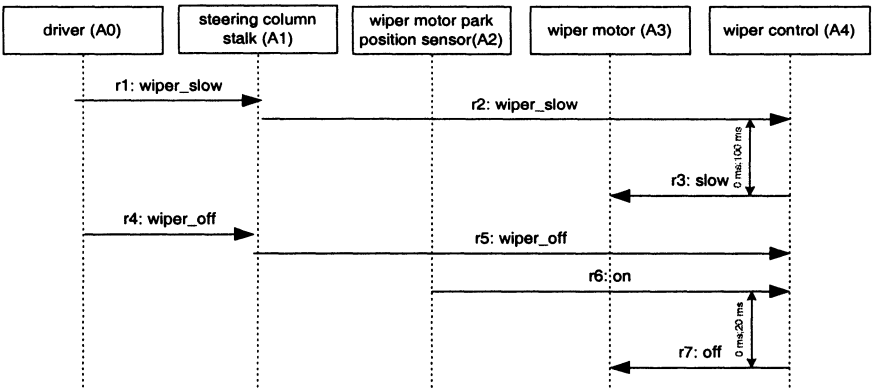

Figure 8: RES – Use case 'Standard wiping front window', main scenario

For completing the test case belonging to the RES, the stimuli sequence is derived from the RES after the component-based control software has been modelled in ASCET-SD. Especially the events, which are sent from the actor driver, could not be mapped to a software component. These events come from external hardware components and become a stimuli event in the stimuli sequence. The events r1.wiper_slow and r4.wiper_off, which can be seen in Figure 8, have become stimuli events because the come from the external actor driver. All stimuli events are sent by the 'TestDriver' component during simulation execution. The 'TestDriver' component also encapsulates the environment model of the missing hardware components. The injected event s6.current_error for example, which can be seen in Figure 9, comes from the model of the wiper motor, which simulates the dynamic behaviour of it.

Figure 9 shows one SES from the simulation results. It has been stimulated by the exemplary test case. The test case has been simulated under the assumption that a mechanical blocking of the wiper motor during the standard wiping has caused a current error, which has been detected by the hardware diagnosis component of the control software sending the event s6.current_error.

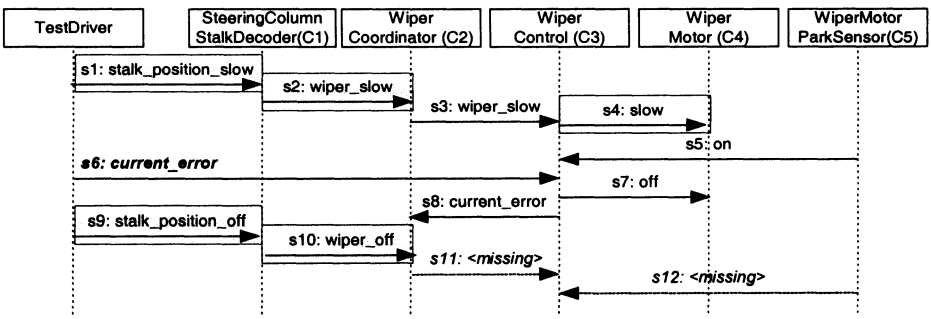

Figure 9: SES – Use case 'Standard wiping front window', main scenario

The offline comparison of the RES against the corresponding SES has delivered the result that the SES is incomplete. The first five events, which are highlighted in Figure 10, have been found, but the last two required events have not been found. The stimulated mechanical blocking of the wiper motor causes a deadlock situation

between the component 'WiperCoordinator' and the component 'WiperControl' in the component-based control software. A simple reconfiguration of the component 'WiperCoordinator' has solved the problem. The example demonstrates how errors in the component-based control software are detected by test case simulation.

6. Conclusions

Component-based development of control software for embedded systems promises overall improved quality and reduced time to market. But it can not be concluded from the higher quality of single components to the correctness of a composed component-based control software. Although design errors are limited to the composition and configuration of components and the overall number of design errors will be less, the validation of a component-based control software is still necessary. Simulation has been selected to achieve an earlier validation of component-based control software and the early detection of design errors. The use case-based systematic gathering of required sequences of events from an users point of view is suitable for the development as well as for the test case design. The transformation of the use case scenarios to extended UML sequence diagrams enables an automatic tool-based offline comparison of the simulated software.

Using a professional case tool like ASCET-SD is a big benefit for modelling and simulation of component-based control software. ASCET-SD enables the simulation of component-based control software for single ECUs respecting real-time behaviour. Additionally it supports real-time data monitoring and recording of time stamp events, which is the necessary input for the offline comparison against the required sequences of events for the validation of the component-based control software in early development phases.

The effort for the test case design and simulation is profitable because error detection and correction in later ECU testing would cost at minimum the same amount of time. Additional benefit is gained by reusing the test cases for later ECU integration tests.

References

[Cock97] Cockburn, Alistair: *Structuring Use Cases with Goals,* The Journal of Object-Oriented Programming, Vol. 10 (5/6), Sept.-Dec. 1997.

[ETAS00] ETAS GmbH: *ASCET-SD 4.0 Manual,* http://www.etas.de, 2000.

[FlRi97] Fleisch, W., Ringler, Th., Belschner, R.: *Simulation of application software for a TTP real-time subsystem,* In Proceedings of European Simulation Multiconference 1997, Istanbul, Turkey, p. 553-558, 1997.

[Flei00] Fleisch, W.: *Simulation and Validation of Component-Based Automotive Control Software,* In Proceedings of 12th European Simulation Symposium, Hamburg, Germany, p. 417-421, 2000.

[Flei99] Fleisch, W.: *Applying Use Cases for the Requirements Validation of Component-Based Real-Time Software,* In Proceedings of 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC99), Saint Malo, France, p. 75-84, 1999.

[Goeh98] Göhner, Peter: *Komponentenbasierte Entwicklung von Automatisierungssystemen,* GMA-Kongreß'98, VDI-Berichte Nr. 1397, p. 513-521, 1998.

[GuNae99] Gunzert, M., Nägele, A.: *Component-Based Development and Verification of Safety-Critical Software for a Break-by-Wire System with Synchronous Software Components,* In Proc. of Int. Symposium on Parallel and Distributed Systems Engineering (PDSE), Los Angeles, CA, USA, p. 134-145, 1999.

[Jaco92] Jacobsen, Ivar et al.: *Object-Oriented Software Engineering: A Use Case Driven Approach,* Addison-Wesley, Reading, Mass., 1992.

[UML00] OMG: *UML v.1.3,* http://www.rational.com/uml/resources/documentation/index.jtmpl, 2000.