

**Forschungsbericht
Institut für Automatisierungs- und
Softwaretechnik**

Hrsg.: Prof. Dr.-Ing. Dr. h. c. P. Göhner

Wolfgang Fleisch

**Validierung
komponentenbasierter Software
für Echtzeitsysteme**

Band 1/2003

Universität Stuttgart

Validierung komponentenbasierter Software für Echtzeitsysteme

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Wolfgang Fleisch
aus Tübingen

Hauptberichter: Prof. Dr.-Ing. Dr. h. c. Peter Göhner
Mitberichter: Prof. Dr.-Ing. Alfred Storr

Tag der Einreichung: 22.05.2002
Tag der mündlichen Prüfung: 27.11.2002

Institut für Automatisierungs- und Softwaretechnik
der Universität Stuttgart

2002

IAS-Forschungsberichte

Band 1/2003

Wolfgang Fleisch

**Validierung komponentenbasierter Software
für Echtzeitsysteme**

D 93 (Diss. Universität Stuttgart)

Shaker Verlag
Aachen 2003

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Fleisch, Wolfgang:

Validierungskomponentenbasierter Software
für Echtzeitsysteme/Wolfgang Fleisch.

Aachen: Shaker, 2003

(IAS-Forschungsberichte; Bd. 2003, 1)

Zugl.: Stuttgart, Univ., Diss., 2002

ISBN 3-8322-1529-8

Copyright Shaker Verlag 2003

Alle Rechte, auch das des auszugsweisen Nachdruckes, der auszugsweisen
oder vollständigen Wiedergabe, der Speicherung in Datenverarbeitungs-
anlagen und der Übersetzung, vorbehalten.

Printed in Germany.

ISBN 3-8322-1529-8

ISSN 1610-4781

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen

Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9

Internet: www.shaker.de • eMail: info@shaker.de

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart.

Mein besonderer Dank gilt

Herrn Prof. Dr.-Ing. Dr. h. c. P. Göhner für die Übernahme des Hauptberichts, die Betreuung dieser Arbeit und die zahlreichen Anregungen, die es ermöglichten, dass die Arbeit in der vorliegenden Form durchgeführt werden konnte,

Herrn Prof. Dr.-Ing. A. Storr für die Übernahme des Mitberichts, die von Anfang an erfolgte Mitbetreuung der Arbeit im Rahmen des Graduiertenkollegs „Parallele und Verteilte Systeme“ und die hilfreichen Hinweise zum inhaltlichen Aufbau der Arbeit,

Herrn Prof. Dr.-Ing. Dr. h. c. mult. P. J. Kühn für die Aufnahme als Stipendiat in das Graduiertenkolleg „Parallele und Verteilte Systeme“ der Universität Stuttgart,

allen Kolleginnen und Kollegen am Institut für die gute Zusammenarbeit,

Michaela Knittel, Paul Linder, Helga Lunkenheimer, Susanne Manz und Thomas Ringler für die kritische Durchsicht des Manuskripts,

den vielen Studien- und Diplomarbeitern, die mit ihrem Engagement einen großen Beitrag zum Gelingen dieser Arbeit geleistet haben,

und schließlich meinen Eltern, für die langjährige Unterstützung während meiner gesamten Ausbildungszeit.

Stuttgart, im April 2002

Wolfgang Fleisch

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Tabellenverzeichnis	vii
Abkürzungsverzeichnis.....	viii
Begriffsverzeichnis	x
Zusammenfassung	xiii
Abstract	xiv
1 Einleitung.....	1
1.1 Bedeutung von Software für Echtzeitsysteme	1
1.2 Problemstellung bei der Validierung komponentenbasierter Software für Echtzeitsysteme	1
1.3 Zielsetzung der Arbeit.....	4
1.4 Gliederung der Arbeit.....	5
2 Grundlagen.....	6
2.1 Grundlagen und Definitionen aus dem Gebiet der Automatisierungstechnik.....	6
2.1.1 Aufbau von Prozessautomatisierungssystemen	6
2.1.2 Echtzeiteigenschaften von Prozessautomatisierungssystemen	8
2.2 Grundlagen und Definitionen aus dem Gebiet der Software-Qualitätssicherung	10
2.2.1 Software-Qualität	10
2.2.2 Fehlermöglichkeiten bei der Softwareentwicklung	10
2.2.3 Maßnahmen zur Software-Qualitätssicherung.....	12
2.2.4 Prüfverfahren zur Software-Qualitätssicherung.....	13
2.2.5 Grundsätzliche Vorgehensweise bei der dynamischen Prüfung von Software.....	15
2.2.6 Validierung und Verifikation von Software	17
3 Stand der Technik.....	20
3.1 Stand der Technik bei der komponentenbasierten Softwareentwicklung für Echtzeitsysteme	20
3.1.1 Anforderungen an die komponentenbasierte Softwareentwicklung für Echtzeitsysteme.....	20
3.1.2 Anforderungen an Beschreibungsmittel zur Entwurfsspezifikation komponentenbasierter Software für Echtzeitsysteme	21
3.1.3 Ansätze zur komponentenbasierten Softwareentwicklung für Echtzeitsysteme.....	26
3.2 Stand der Technik bei der Validierung von Software für Echtzeitsysteme	31

3.2.1	Anforderungen an Prüfverfahren zur Validierung von Software für Echtzeitsysteme.....	31
3.2.2	Ansätze zur Prüfung von Software für Echtzeitsysteme.....	31
3.3	Zusammenfassende Bewertung und Festlegung der Anforderungen.....	36
3.3.1	Bewertung der Ansätze zur komponentenbasierten Softwareentwicklung für Echtzeitsysteme.....	36
3.3.2	Bewertung der Ansätze zur Prüfung von Software für Echtzeitsysteme	36
3.3.3	Anforderungen an ein Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme	37
4	Konzeption eines Prüfverfahrens zur Validierung komponentenbasierter Software für Echtzeitsysteme.....	39
4.1	Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme	39
4.1.1	Übersicht über den Entwicklungsprozess	39
4.1.2	Anforderungsanalyse	40
4.1.3	Komponentenbasierter Entwurf	41
4.1.4	Implementierung auf Basis von Generierung	42
4.1.5	Komponenten-Integration und System-Integration	42
4.2	Fehlermöglichkeiten bei der komponentenbasierten Softwareentwicklung für Echtzeitsysteme	43
4.3	Grundkonzept des Prüfverfahrens.....	44
5	Spezifikation prüfbarer Anwendungsfälle	46
5.1	Anwendungsfälle zur Verhaltensspezifikation.....	46
5.1.1	Eigenschaften von Anwendungsfällen.....	46
5.1.2	UML Anwendungsfall-Diagramm.....	47
5.1.3	Spezifikationsschablone für Anwendungsfälle.....	48
5.2	Ermittlung einer geeigneten Notation zur formalisierten Verhaltensspezifikation von Anwendungsfällen	50
5.3	Sequenzdiagramme zur formalisierten Verhaltensspezifikation von Anwendungsfällen	52
5.3.1	UML Sequenzdiagramme	52
5.3.2	Erweiterung der Eigenschaften von UML Sequenzdiagrammen.....	53
5.4	Überführung der Anwendungsfälle in formalisierte Anwendungsszenarios	58
6	Erstellung des Simulationsmodells.....	61
6.1	Anforderungen an den Aufbau des Simulationsmodells.....	61
6.1.1	Realitätsnahes simuliertes Verhalten	61
6.1.2	Modell der komponentenbasierten Software	62
6.1.3	Modell der Umgebung	63
6.1.4	Simulationsmodell eines verteilten Echtzeitsystems	65
6.1.5	Simulationsablaufsteuerung.....	66
6.1.6	Generierung von Stimulations-Daten	66
6.1.7	Beobachtung und Aufzeichnung von Nachrichten im Simulationsmodell.....	68
6.2	Auswahl eines geeigneten Simulationswerkzeugs.....	69

6.3	Erstellung eines Simulationsmodells für einzelne Steuergeräte mit dem Werkzeug ASCET-SD	71
6.4	Erstellung eines Simulationsmodells für verteilte Steuergeräte mit den Werkzeugen ASCET-SD und CANoe	72
6.4.1	Übersicht über das Simulationswerkzeug CANoe.....	72
6.4.2	Vorgehensweise zur Erstellung eines Simulationsmodells mit ASCET-SD und CANoe	73
6.4.3	Aufbau eines CANoe-Simulationsmodells	74
7	Prüfung des simulierten Verhaltens der komponentenbasierten Software	77
7.1	Übersicht über das Prüfverfahren zur Validierung	77
7.2	Entwicklung von Prüffällen durch Ableitung von Stimuli-Sequenzen.....	78
7.2.1	Systematik zur Ableitung von Stimuli-Sequenzen	78
7.2.2	Aufbau einer Stimuli-Sequenz.....	79
7.3	Simulation der Prüffälle	80
7.4	Konformitätsprüfung.....	81
7.4.1	Prüfkriterien zwischen zwei Sequenzdiagrammen	81
7.4.2	Zuordnung der Nachrichten in Anwendungsszenarios auf Nachrichten im Simulationsmodell	84
7.4.3	Prüfung des Verhaltens eines gesamten komponentenbasierten Entwurfsmodells	87
7.5	Prüfergebnis	89
7.5.1	Prüfergebnis für einzelne Zeitstempel-Sequenzen.....	89
7.5.2	Fehlerursachen und Auswirkungen.....	90
7.5.3	Aufdeckung und Lokalisierung von Entwurfsfehlern.....	91
8	Realisierung einer Validierungsumgebung	93
8.1	Systemarchitektur der Validierungsumgebung	93
8.2	Validierungs-Projekt-Manager.....	94
8.3	Validierungs-Projekt-Datenbank.....	95
8.4	Anwendungsfall-Editor	96
8.5	Sequenzdiagramm-Editor und Interpreter.....	96
8.6	Zuordnungstabellen-Editor.....	98
8.7	Stimuli-Editor und Generator.....	98
8.8	Simulationsaufzeichnungs-Filter.....	100
8.9	Konformitäts-Prüfer	100
8.10	Prüfergebnis-Visualisierung.....	101
8.11	Bewertung der Werkzeugunterstützung	103
9	Anwendung des Prüfverfahrens zur Validierung der komponentenbasierten Software einer Kfz-Scheibenwischeranlage	105
9.1	Übersicht über die Scheibenwischeranlage.....	105
9.2	Anwendungsfälle.....	107
9.3	Anwendungsszenarios.....	108

9.4	Simulationsmodell des Echtzeitsystems Scheibenwischeranlage	110
9.5	Stimuli-Sequenzen	111
9.6	Prüfergebnisse	112
9.7	Zusammenfassung der Ergebnisse für das Anwendungsbeispiel.....	116
10	Schlussfolgerungen und Ausblick	117
10.1	Zusammenfassung der Ergebnisse	117
10.2	Vorteile und Grenzen des entwickelten Prüfverfahrens.....	119
10.3	Ausblick	120
	Literaturverzeichnis.....	122

Abbildungsverzeichnis

Abbildung 1.1:	Verteilung der Fehlerentstehung und Fehlerbeseitigung in den verschiedenen Entwicklungsphasen nach Poston	2
Abbildung 2.1:	Prinzipielle Struktur eines Prozessautomatisierungssystems.....	6
Abbildung 2.2:	Beispiel für die Struktur eines verteilten Automatisierungssystems	7
Abbildung 2.3:	Übersicht über Produkt-bezogene Prüfverfahren zur Software-Qualitätssicherung.....	14
Abbildung 2.4:	V-Modell zur Softwareentwicklung	18
Abbildung 3.1:	Beispiel für ein Strukturmodell einer komponentenbasierten Software	23
Abbildung 3.2:	Beispiel für ein Strukturmodell einer objektorientierten Software	23
Abbildung 3.3:	Beispiel für ein Verteilungsmodell	24
Abbildung 4.1:	Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme	40
Abbildung 4.2:	Grundkonzept für ein Prüfverfahren zur Validierung des dynamischen Verhaltens von komponentenbasierter Software für Echtzeitsysteme.....	45
Abbildung 5.1:	Aufbau eines Anwendungsfall-Diagramms	47
Abbildung 5.2:	Aufbau eines UML Sequenzdiagramms	53
Abbildung 5.3:	Sequenzdiagramm mit Vor- und Nachbedingung.....	54
Abbildung 5.4:	Sequenzdiagramm mit Zeitbedingung	55
Abbildung 5.5:	Sequenzdiagramm mit periodischen Teil-Sequenzen	56
Abbildung 5.6:	Beispiel für die Herleitung von Anwendungsszenarios aus einem Anwendungsfall	59
Abbildung 6.1:	Wesentliche Bestandteile des Simulationsmodells	62
Abbildung 6.2:	Modell der komponentenbasierten Software im Simulationsmodell.....	63
Abbildung 6.3:	Modell der Umgebung im Simulationsmodell.....	64
Abbildung 6.4:	Simulationsmodell eines verteilten Echtzeitsystems	65
Abbildung 6.5:	Ableitung von Stimulations-Daten	66
Abbildung 6.6:	Generierung von Stimulations-Daten	67
Abbildung 6.7:	Zeitstempel-Aufzeichnung von Nachrichten im Simulationsmodell.....	68
Abbildung 6.8:	Aufbau eines Simulationsmodells für ein einzelnes Steuergerät mit ASCET-SD	71
Abbildung 6.9:	Erstellung der Windows-DLL zur Simulation der komponentenbasierten Software eines Steuergeräts in CANoe.....	74
Abbildung 6.10:	Aufbau eines Simulationsmodells für verteilte Steuergeräte mit CANoe	75
Abbildung 7.1:	Schritte des Prüfverfahrens zur Validierung	77
Abbildung 7.2:	Aufbau einer Stimuli-Sequenz	79
Abbildung 7.3:	Vollständige Abbildung der Nachrichten eines Anwendungsszenarios auf die Nachrichten einer Zeitstempel-Sequenz	82
Abbildung 7.4:	Periodisches Auftreten einer Nachricht in einer Zeitstempel-Sequenz	84
Abbildung 7.5:	Zuordnung von Nachrichten zwischen den Anwendungsszenarios und den Komponenten-Schnittstellen in einem Simulationsmodell.....	85
Abbildung 7.6:	Zuordnung von Nachrichten bei verteilten Echtzeitsystemen	87
Abbildung 7.7:	Prüfstrategie zur Prüfung des Verhaltens eines gesamten komponentenbasierten Entwurfsmodells	88
Abbildung 8.1:	Systemarchitektur der Validierungsumgebung AVE.....	93

Abbildung 8.2:	Benutzungsoberfläche der Validierungsumgebung AVE	94
Abbildung 8.3:	Validierungs-Projekt-Manager-Menü	95
Abbildung 8.4:	Anwendungsfall-Editor-Menü	96
Abbildung 8.5:	Sequenzdiagramm-Editor und Interpreter-Menü	96
Abbildung 8.6:	Grafischer Editor für erweiterte UML Sequenzdiagramme.....	97
Abbildung 8.7:	Dialog-Fenster bei automatischer Änderung der Zuordnungstabelle	97
Abbildung 8.8:	Stimuli-Editor und Generator-Menü.....	98
Abbildung 8.9:	Grafischer Stimuli-Editor.....	99
Abbildung 8.10:	Simulationsaufzeichnungs-Filter-Menü.....	100
Abbildung 8.11:	Auswahl-Dialog-Fenster für den Detaillierungsgrad des Prüfberichts	100
Abbildung 8.12:	Prüfbericht für eine einzelne Konformitätsprüfung	101
Abbildung 8.13:	Fehlervisualisierung im geprüften Anwendungsszenario	102
Abbildung 8.14:	Gesamtergebnis-Prüfbericht für ein gesamtes Validierungs-Projekt.....	102
Abbildung 9.1:	Scheibenwischeranlage im Modellprozess IAS-Cockpit.....	105
Abbildung 9.2:	Hardware-Komponenten der Scheibenwischeranlage	106
Abbildung 9.3:	Anwendungsfall-Diagramm für die Scheibenwischeranlage.....	107
Abbildung 9.4:	Anwendungsfall „W2 Schnell Wischen“	108
Abbildung 9.5:	Anwendungsszenario „W2_1 Scheibe Schnell Wischen“	110
Abbildung 9.6:	Simulationsmodell der Scheibenwischeranlage in ASCET-SD	111
Abbildung 9.7:	Stimuli-Sequenz zum Anwendungsszenario „W2_1 Scheibe Schnell Wischen“	112
Abbildung 9.8:	Fehlerhaftes Prüfergebnis der zugehörigen Zeitstempel-Sequenz zum Anwendungsszenario „W2_1 Scheibe Schnell Wischen“	114
Abbildung 9.9:	Fehlerhaftes Prüfergebnis der zugehörigen Zeitstempel-Sequenz zum Anwendungsszenario „W5_2 Überlaststrom Aufheben“	115

Tabellenverzeichnis

Tabelle 2.1: Beispiele für Qualitätssicherungsmaßnahmen	12
Tabelle 3.1: Beispiel für die konfigurierbaren Parameter und Attribute einer Komponente	22
Tabelle 3.2: Beschreibungsmittel komponentenbasierter Softwareentwicklungsmethoden für Echtzeitsysteme.....	26
Tabelle 3.3: Bewertung der Werkzeugunterstützungen	30
Tabelle 3.4: Ansätze zur Prüfung von Software für Echtzeitsysteme.....	32
Tabelle 5.1: Spezifikationsschablone für Anwendungsfälle	48
Tabelle 5.2: Bewertung unterschiedlicher Diagrammarten zur formalisierten Verhaltensspezifikation von Anwendungsfällen	51
Tabelle 6.1: Bewertung der Entwurfs-, Simulations- und Codegenerierungswerkzeuge ASCET-SD und ObjecTime Developer.....	70
Tabelle 7.1: Zuordnungstabelle für Nachrichten und Nachrichten-Bedingungen	86
Tabelle 7.2: Entscheidungstabelle zur Ermittlung des Prüfergebnisses.....	89
Tabelle 7.3: Charakteristische Fehlerursachen und deren Auswirkungen beim Entwurf komponentenbasierter Software für Echtzeitsysteme	90
Tabelle 9.1: Anwendungsszenarios für die Scheibenwischeranlage.....	109
Tabelle 9.2: Prüfergebnisse des Validierungs-Projekts „Scheibenwischeranlage“	113

Abkürzungsverzeichnis

ASCET-SD	A dvanced S imulation and C ontrol E ngineering T ool – S oftware D evelopment
ASCII	A merican S tandard C ode for I nformation I nterchange
ASPECT	A bteilungsübergreifende S teuerungs P rogramm E rstellung mit C ASE- T ool
AVE	A utomated V alidation E nvironment
AWL	A n W eisungs L iste
CAN	C ontroller A rea N etwork
CANoe	C AN o pen e nvironment
CAPL	C AN A pplication P rogramming L anguage
CASE	C omputer A ided S oftware E ngineering
CMM	C apability M aturity M odel
CORBA	C ommon O bject R equest B roker A rchitecture
DLL	D ynamic L ink L ibrary
EPOSIX	E xtended S pecification L anguage for P ortable O perating S ystem I nterface
ERCOS ^{EK}	E mbedded R eal- T ime C ontrol O perating S ystem based on the O SE K S tandard
ESDL	E mbedded S oftware D escription L anguage
FIFO	F irst I n F irst O ut
IAS	I nstitut für A utomatisierungs- und S oftwaretechnik der U niversität S tuttgart
IEC	I nternational E lectrotechnical C ommission
IEEE	I nstitute of E lectrical and E lectronics E ngineers
IPC	I ndustrie P ersonal C omputer
ISO	I nternational S tandardization O rganisation
Kfz	K raft f ahrzeug
LSC	L ive S equene C harts
MFC	M icrosoft F oundation C lasses

MSC	M essage S equence C hart
MS-COM	M icro S oft - C omponent O bject M odel
MSR	M anufacturer S upplier R elationship
OIL	O SEK I mplementation L anguage
OOA	O bject- O riented A nalysis
OOD	O bject- O riented D esign
OSEK/VDX	O ffene S ysteme und deren Schnittstellen für die E lektronik im K raftfahrzeug / V ehicle D istributed E xecutive
PLS	P rozess L eit S ystem
POSIX	P ortable O perating S ystem I nterface
ROOM	R eal- T ime O bject- O riented M odeling
SDL	S pecification and D escription L anguage
SPS	S peicher P rogrammierbare S teuerungen
TC-SL	T iming C onstraint – S pecification L anguage
UML	U nified M odeling L anguage
VME	V ersa M odule E urope

Begriffsverzeichnis

Akteur: Eine Rolle bei der Spezifikation von Anwendungsfällen, die eine Person oder technisches System im Bezug auf eine bestimmte Angelegenheit spielt

Aktor: Einheit zur Umsetzung von Stellinformation tragenden Signalen geringer Leistung in leistungsbehafete Signale einer zur Prozessbeeinflussung notwendigen Energieform

Anwendungsfall: Die Beschreibung einer Menge von sequenziellen Aktionen, einschließlich Alternativen und Varianten, die ein System ausführt, um ein erkennbares, für einen externen Akteur nützliches Ziel zu erreichen

Anwendungsszenario: Eine Sequenz von Nachrichten, die eine Anforderung an das Verhalten einer komponentenbasierten Software darstellt

Automatisierungsrechner: Ein Rechner mit der Eigenschaft Daten zeitgerecht zu erfassen, zu verarbeiten und auszugeben. Der Automatisierungsrechner verfügt über Prozessperipheriegeräte zur Ein- und Ausgabe von Prozess-Signalen (elektrische Signale)

Daten: Eine wieder interpretierbare Darstellung von Information in formalisierter Weise, geeignet zur Übermittlung, Deutung oder Verarbeitung

Echtzeitsystem: Ein Rechnersystem, das mit externen technischen Systemen in einer Wechselwirkung steht und bei dem die Verarbeitung der Programme zeitlich mit den in externen Systemen ablaufenden Vorgängen Schritt halten muss

Implementierung: Das Ergebnis der Übersetzung eines Entwurfs in ein Programm. Ebenso wird die Entwicklungsaktivität zur Erstellung eines Programms als Implementierung bezeichnet

Kommunikationssystem: Gesamtheit aller Einrichtungen (Hardware und Software) in einem Rechnernetz, die für die physikalische Übertragung von Nachrichten zwischen räumlich getrennten Rechnern und damit zwischen verteilten Programmteilen notwendig sind

Komponente (Software-Komponente): Eine in sich abgeschlossene funktionale Einheit mit vollständig spezifizierten Schnittstellen, die gut dokumentiert, qualitativ hochwertig und unverändert mehrfach verwendbar ist. Eine Komponente ist außerdem unabhängig von anderen Komponenten, an gegebene Aufgabenstellungen anpassbar und bezüglich bestimmter Eigenschaften konfigurierbar

Komponentenbasierte Software: Eine aus Komponenten zusammengesetzte Software

Nachbedingung: Ein logischer Ausdruck von Systemgrößen bei einem Anwendungsfall, der erfüllt sein muss, damit das Ziel eines Anwendungsfalls erreicht ist

Prüffall: Ein Prüffall beschreibt, welche Funktion des Prüfgegenstands mit welcher Genauigkeit zu prüfen ist, welche Ausgangssituation hierfür erforderlich ist, welche stimulierenden Referenz-Eingaben (Daten und Signale mit allen für die Prüfung ausschlaggebenden Eigenschaften wie Zeitbedingungen) notwendig sind und welche Referenz-Ergebnisse (Ausgabedaten und Reaktionen/Effekte) zu erwarten sind

Prüfung: Eine Tätigkeit, wie Messen oder Untersuchen von einem oder mehreren Merkmalen eines Prüfgegenstands sowie das Vergleichen mit in einer Referenz festgelegten Anforderungen, um festzustellen, ob Konformität für jedes Merkmal erzielt ist

Sensor: Einheit zur Umsetzung von physikalischen oder chemischen Größen in eine elektronische Größe

Simulation: Bei der Simulation wird ermittelt, wie ein vorgegebenes Simulationsmodell eines Systems auf bestimmte Eingaben reagiert. Das Simulationsmodell wird dabei zum Prüfgegenstand. Analog zum Testen kann das Simulationsmodell unter spezifizierten Bedingungen kontrolliert ausgeführt werden, mit dem Ziel, Fehler zu entdecken

Spezifikation: Eine detaillierte Beschreibung der Teile eines Ganzen und ihrer Eigenschaften bezüglich Größe, Qualität, Leistungsfähigkeit usw. sowie ihrer Beziehungen untereinander

Steuergerät: Eine spezielle Art von Automatisierungsrechner, der neben einem Mikrocontroller zur Bearbeitung arithmetischer Operationen und logischer Verknüpfungen auch Prozessperipherie zur Ein- und Ausgabe von Prozess-Signalen (elektrische Signale) besitzt

Technischer Prozess: Ein Prozess, dessen physikalische Größen mit technischen Mitteln erfasst und beeinflusst werden

Testen: Die kontrollierte Ausführung eines Prüfgegenstands, z.B. ein Programm oder eine Funktion eines Programms, unter spezifizierten Bedingungen mit dem Ziel, Fehler zu entdecken. Bei der Ausführung werden die Ergebnisse beobachtet oder aufgezeichnet und anhand dieser eine Bewertung des Prüfgegenstands unter ausgewählten Prüfkriterien durchgeführt

Validierung: Die Nachweisführung, bei der gezeigt wird, dass ein Produkt die Erwartungen des Anwenders erfüllt

Verifikation: Die Nachweisführung, bei der gezeigt wird, dass ein Produkt die Anforderungen erfüllt, die während vorhergehender Aktivitäten erstellt wurden

Vorbedingung: Ein logischer Ausdruck von Systemgrößen bei einem Anwendungsfall, der erfüllt sein muss, damit ein Szenario eines Anwendungsfalls ablaufen kann

Zeitstempel-Sequenz: Sequenz von beobachteten Nachrichten, die jeweils mit der aktuellen Simulationszeit und dem Nachrichten-Bezeichner protokolliert wurden

Zusammenfassung

Der Anteil und die Bedeutung von Software bei der Entwicklung von Echtzeitsystemen wächst sehr stark. Gleichzeitig werden die Entwicklungszyklen für Software-Produkte zunehmend kürzer, was sich immer häufiger in ausgelieferter fehlerhafter Software widerspiegelt. Durch die Verwendung vorgefertigter Software-Komponenten kann nach dem Baukastenprinzip bei der Konstruktion neuer Produkte die Produktivität und die Qualität erheblich gesteigert werden. Aber auch wenn durch den Einsatz geprüfter, betriebsbewährter Software-Komponenten viele Fehler vermieden werden, muss eine aus Komponenten zusammengesetzte Software auch in Zukunft mit geeigneten Prüfverfahren validiert werden, weil beim Zusammenspiel der Komponenten weiterhin Fehler entstehen können.

In der vorliegenden Arbeit wird dazu ein Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme entwickelt, das das dynamische Verhalten von komponentenbasierter Software gegenüber den Anforderungen aus Anwendersicht überprüft. Die Anforderungen werden zu Beginn der komponentenbasierten Softwareentwicklung als Anwendungsfälle spezifiziert. Mit Hilfe von erweiterten UML Sequenzdiagrammen werden die Anwendungsfälle systematisch in eine formale, automatisiert prüfbare Notation überführt, die auch die Spezifikation von Echtzeitanforderungen explizit unterstützt. Da beim Entwurf von komponentenbasierter Software bereits vollständig spezifizierte Software-Komponenten verwendet werden, kann das Verhalten schon in der frühen Entwurfsphase mit Hilfe von Simulation validiert werden. Die Verwendung von Simulation hat den Vorteil, dass auch ohne die in der Praxis meist spät verfügbaren Bestandteile des technischen Prozesses und des Automatisierungssystems eine Prüfung des simulierten Verhaltens der komponentenbasierten Software durchgeführt werden kann. Anhand festgelegter Prüfkriterien wird dann das simulierte Verhalten der komponentenbasierten Software gegenüber dem in den Anwendungsfällen spezifizierten Verhalten geprüft, um damit frühzeitig Fehler aufzudecken.

Eine im Rahmen der Arbeit realisierte Validierungsumgebung unterstützt den Softwareentwickler dabei umfassend, angefangen bei der Spezifikation von prüfbaren Anwendungsfällen, bis hin zur automatisierten Prüfung des simulierten Verhaltens. Die Leistungsfähigkeit des Prüfverfahrens und der Validierungsumgebung werden abschließend am Beispiel der Entwicklung und Validierung einer komponentenbasierten Software für eine Kfz-Scheibenwischeranlage demonstriert.

Abstract

Software has become a crucial factor in the development of real-time systems. At the same time the development cycles of software products are increasingly shortened, which results more and more in erroneous software delivered to customers. The usage of prefabricated software components for the development of new products significantly improves the development productivity and product quality. Although the usage of already tested and proven in use software components avoids many errors in advance, the dynamic behaviour of a composed software still has to be validated by a suitable testing method because new errors can arise from the interaction of the components.

For that reason a testing method for the validation of component-based software for real-time systems is developed in this thesis. The testing method checks the dynamic behaviour of a component-based software against user requirements. Such user requirements are specified by use cases at the beginning of the component-based software development. For achieving a testable notation, the use cases are systematically transferred into the formal language of extended UML sequence diagrams. The extensions of the UML sequence diagram syntax provide the additional specification of enhanced real-time requirements. With that preparation, the dynamic behaviour of a component-based software can be validated by simulation in the early design phase, because all prefabricated software components used for the design have already a completely specified behaviour. Applying simulation for the validation of the dynamic behaviour of a component-based software for real-time systems is profitable, because the simulation can be processed earlier without the real components of the technical process and the automation computer system. These components are usually not available during the early software development phases of a real-time system. The developed testing method finally checks the simulated behaviour of the component-based software against the specified behaviour of the use cases employing defined comparison criteria. In that way the testing method enables the early detection of errors in component-based software for real-time systems.

The validation environment, which has been also developed in the context of this thesis, provides a comprehensive assistance for the validation of component-based software for real-time systems. It supports the specification of testable use cases up to an automated conformance check of the simulated behaviour against the specified behaviour from the use cases. The capability of the testing method and of the validation environment are exemplary demonstrated at the development and validation of a component-based software for an automotive wiper system.

1 Einleitung

1.1 Bedeutung von Software für Echtzeitsysteme

Echtzeitsysteme sind in vielen Bereichen unseres täglichen Lebens, wie z.B. in Haushaltsgeräten, Telekommunikationsanlagen, Produktionsanlagen, Kraftfahrzeugen sowie in der Luft- und Raumfahrttechnik, anzutreffen. Sie unterliegen strengen Anforderungen in Bezug auf ihr zeitliches Verhalten und übernehmen häufig sicherheitsrelevante Aufgaben [Thal97]. Bei Ausfällen oder Versagen können neben Maschinen im schlimmsten Fall sogar Menschenleben gefährdet sein.

Der Anteil und die Bedeutung von Software bei der Entwicklung von Echtzeitsystemen wächst sehr stark. Immer komplexere Automatisierungsaufgaben steigern gleichermaßen die Komplexität der Software. In der Produktautomatisierung kann die rapide wachsende Komplexität eindrucksvoll am Beispiel der Innovationen in der Kraftfahrzeug-Elektronik untermauert werden. Während in den siebziger Jahren die Kraftfahrzeuge nur über elektrische und mechanische Automatisierungssysteme gesteuert und geregelt wurden, erfolgt dies heute zunehmend über elektronische Systeme und die darin enthaltene Software [Seif01]. Laut Angaben eines führenden deutschen Automobilherstellers [Daim98] entfallen bereits heute ungefähr 30 Prozent der Herstellungskosten eines Kraftfahrzeugs auf elektronische Systeme, obwohl auf dem Elektroniksektor Preissenkungen von jährlich bis zu 50 Prozent zu verzeichnen sind. Die Tendenz ist weiterhin stark ansteigend. In den neuen Modellen der PKW-Oberklasse sind derzeit zwischen 50 und 100 elektronische Steuergeräte integriert, welche zum Großteil über Kommunikationssysteme miteinander verbunden sind. Die in den verteilten Steuergeräten ausgeführte Automatisierungssoftware erbringt dabei die eigentliche Systemfunktionalität, um den Fahrer beim Führen des Fahrzeugs zu unterstützen. Weiterhin sind viele Automatisierungsfunktionen, wie beispielsweise die Optimierung von Abgasemissionen bei Verbrennungsmotoren, erst durch den Einsatz von Software kostengünstig machbar.

1.2 Problemstellung bei der Validierung komponentenbasierter Software für Echtzeitsysteme

In der industriellen Praxis werden die Entwicklungszyklen für Produkte immer kürzer bei gleichzeitig knappen Budgets. Dies geht oft zu Lasten der Qualität, was sich immer häufiger in ausgelieferter fehlerhafter Software widerspiegelt. Fehler in der Software können im Rahmen der Produkthaftung enorme Folgekosten verursachen, wie z.B. durch Rückrufaktionen bei Massenprodukten. Die General Motors Corp. [Beus96] musste 1996 beispielsweise 292.860

Fahrzeuge der Marken Pontiac, Oldsmobile und Buick der Baujahre 1995 und 1996 in die Werkstätten zurückrufen, weil ein Softwarefehler beim Motorstart einen Brand verursachen konnte. Eine fehlerhafte Sequenz in der Software des Motorsteuergeräts konnte zu Fehlzündungen beim Starten des Motors führen. Dabei entstanden teilweise Risse im Ansaugstutzen, wodurch unter ungünstigen Umständen Feuer ausgelöst wurde. Ein weiteres eindrucksvolles Beispiel für nicht rechtzeitig aufgedeckte Softwarefehler war der Verlust der Ariane 5 Rakete auf ihrem Jungfernflug. Die daraufhin eingesetzte Untersuchungskommission stellte in ihrem Bericht [ESA96] fest, dass durch Spezifikations- und Entwurfsfehler in der Software des Trägheitssensorsystems das gesamte Flugsteuerungssystem der Rakete sämtliche Navigations- und Höheninformationen verlor. Die Rakete geriet dadurch 30 Sekunden nach dem Abheben vollständig außer Kontrolle und stürzte ab. Der Bericht schließt unter anderem mit dem Hinweis, dass durch zusätzliche Prüfmaßnahmen bei der Entwicklung des gesamten Flugsteuerungssystems der Verlust der Rakete hätte verhindert werden können.

Da unter Software nicht nur ein ausführbares Programm verstanden wird, sondern auch alle dazugehörigen Entwicklungsdokumente, Quellcode, Prüffälle usw., beschränken sich die Fehlermöglichkeiten bei der Entwicklung nicht nur auf die Programmierung, sondern auf alle Tätigkeiten bei der Entwicklung. Nach den Untersuchungen von Poston [Post85] entstehen bei der Softwareentwicklung bereits während der frühen Anforderungs- und Entwurfsphase rund 55 Prozent aller Entwicklungsfehler. Auf der anderen Seite wird ein Großteil der Entwicklungsfehler erst beim Abnahmetest und in der Betriebsphase der Software aufgedeckt und beseitigt. Abbildung 1.1 zeigt die Verteilung der Fehlerentstehung und Fehlerbeseitigung in den verschiedenen Entwicklungsphasen.

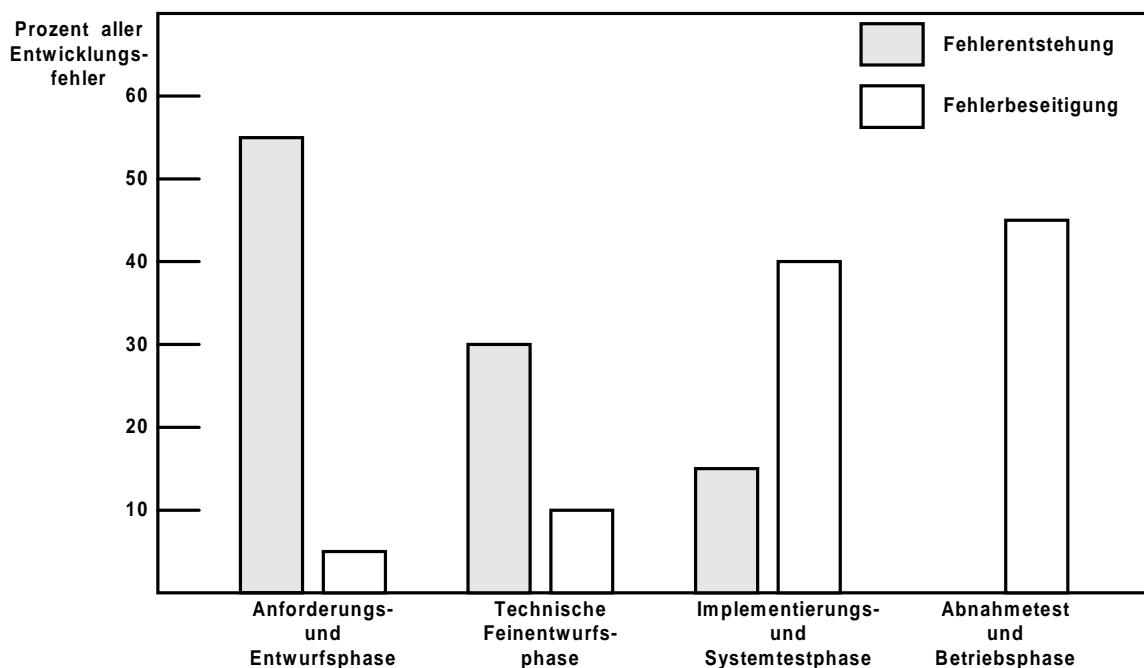


Abbildung 1.1: Verteilung der Fehlerentstehung und Fehlerbeseitigung in den verschiedenen Entwicklungsphasen nach Poston

Die Erkenntnisse von Poston besitzen noch heute ihre Gültigkeit [Balz98, WaCo96]. Dies gilt in besonderem Maße auch für Automatisierungssoftware in Echtzeitsystemen, weil bei diesen aus der Sicht des Auftraggebers bzw. des Anwenders die Fehler in der Software oft erst nach der Inbetriebnahme erkennbar sind. Bei der Entwicklung von Automatisierungssystemen kommt noch erschwerend hinzu, dass die Bestandteile des Automatisierungssystems und des technischen Prozesses häufig erst sehr spät für eine Software-Prüfung verfügbar sind. Teilweise ist die Software zur Erfüllung ihrer Funktionalität zusätzlich noch auf mehrere Automatisierungsrechner verteilt. So passiert es in der Praxis sehr häufig, dass z.B. nicht berücksichtigte Anforderungen des Auftraggebers in der Software erst nach dem Abnahmetest aufgedeckt und beseitigt werden. Dies ist jedoch die teuerste Form der Fehlerbeseitigung. Verschiedene Studien [StWi00, Zale01] zeigen, dass die Beseitigung von aufgedeckten Fehlern in der Software nach der Abnahme um ein Vielfaches teurer ist als direkt nach der Implementierung.

Das primäre Ziel muss also darin bestehen, Fehler durch den Einsatz konstruktiver Maßnahmen erst gar nicht zu machen. Die komponentenbasierte Softwareentwicklung wird in vielen Quellen [Göhn98, HoRi99, KoBo98, SLSW99, Stan92, Szym98] als vielversprechende konstruktive Maßnahme zur Verbesserung der heute existierenden Zeit-, Kosten- und Qualitätsprobleme bei Softwareentwicklungsprojekten in den unterschiedlichsten Anwendungsbereichen gesehen. In Anlehnung an die Vorgehensweisen in anderen Ingenieurdisziplinen kann durch den Einsatz von Software-Komponenten die Komplexität der Softwareentwicklung stark reduziert werden. Außerdem können die Software-Komponenten in verschiedenen Anwendungen mehrfach verwendet werden, was die Produktivität erheblich steigert. Die Idee der Software-Mehrfachverwendung ist nicht neu. Während aber früher bei Funktionen, Modulen oder auch bei Klassen die Implementierungssicht im Vordergrund stand, orientieren sich heute Software-Komponenten stärker an den zu realisierenden Anwendungen [Göhn98]. Doch auch durch den Einsatz von Software-Komponenten lassen sich Fehler nicht vollständig vermeiden. Da konstruktive Maßnahmen zur Fehlervermeidung, wie z.B. die komponentenbasierte Softwareentwicklung, nicht geeignet sind, um die Qualität eines entwickelten Produkts nachzuweisen, muss beispielsweise die Einhaltung der Anforderungen des Auftraggebers auch zukünftig weiter validiert werden. Die Validierung ist dabei eine Prüfkativität, bei der gezeigt wird, dass das Produkt die Erwartungen des Anwenders erfüllt.

Das sekundäre Ziel muss deshalb darin bestehen, Fehler, die dennoch gemacht wurden, so früh wie möglich aufzudecken und zu beseitigen. Dies kann mit Hilfe analytischer, prüfender Maßnahmen erreicht werden. Die Durchführung analytischer Qualitätssicherungsmaßnahmen ist ein sehr zeitaufwändiger Teil der Softwareentwicklung. Nach einer Studie von Stolze und Willert [StWi00] werden bereits heute in den meisten Softwareentwicklungsprojekten ungefähr 60 Prozent der Entwicklungszeit für analytische Qualitätssicherungsmaßnahmen, schwerpunktmäßig für das Testen und die Fehlerbeseitigung, aufgewendet. Trotz des großen Aufwands haben die Qualitätsprobleme in der Praxis nicht merklich abgenommen.

Während sich die Software-Entwicklungsmethoden und damit die konstruktiven Qualitätssicherungsmaßnahmen in den letzten Jahren sehr stark weiterentwickelt haben, hinkt die Entwicklung neuer Methoden für die analytische Qualitätssicherung, insbesondere von geeigneten Software-Prüfverfahren, deutlich hinterher. In der industriellen Praxis besteht das Prüfen von Software für Echtzeitsysteme oftmals nur in der Durchführung einiger unsystematischer und nicht reproduzierbarer Prüffälle [Ligg93, StWi00].

Zusammenfassend ergeben sich für die Validierung von komponentenbasierter Software für Echtzeitsysteme derzeit folgende Probleme:

- Wachsende Anforderungen an die Funktionalität und die Qualität der Software für Echtzeitsysteme stehen im Widerspruch zu kürzeren Entwicklungszeiten und knappen Entwicklungsbudgets.
- Ein Großteil der Fehler entsteht bereits in den frühen Entwicklungsphasen, wird aber erst in den späten Abnahmetests aufgedeckt.
- In der Praxis kann die Prüfung der Software oftmals erst in den späten Entwicklungsphasen durchgeführt werden, da die Bestandteile des technischen Prozesses und des Automatisierungrechnersystems zuvor nicht verfügbar sind.
- Spät aufgedeckte Fehler können zu enormen Folgekosten bei der Fehlerbeseitigung führen.
- Konstruktive Qualitätssicherungsmaßnahmen, wie z.B. die komponentenbasierte Software-Entwicklung, sind nicht geeignet um Software-Qualität nachzuweisen.
- Software-Prüfungen werden in der industriellen Praxis häufig unsystematisch und dadurch wenig effizient durchgeführt.

1.3 Zielsetzung der Arbeit

Im Rahmen dieser Arbeit soll ein praxisnahes Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme entwickelt werden. Bei der Entwicklung des Prüfverfahrens werden folgende Ziele verfolgt:

- Mit Hilfe des Prüfverfahrens soll die Einhaltung von Anforderungen durch die Software aus der Sicht des Auftraggebers bzw. der zukünftigen Anwender systematisch validiert werden.
- Die Validierung soll Fehler, wie z.B. nicht berücksichtigte Anforderungen, in der komponentenbasierten Software eines Echtzeitsystems möglichst frühzeitig aufdecken.
- Das Prüfverfahren soll die spezifischen Eigenschaften von komponentenbasierter Software und von Echtzeitsystemen gleichermaßen berücksichtigen.

- Der Aufwand bezüglich Zeit und Kosten für die Validierung soll nicht größer sein als bei bisher eingesetzten Prüfverfahren.

1.4 Gliederung der Arbeit

In Kapitel 2 werden für das Verständnis der Arbeit notwendige Begriffe aus den Gebieten der Automatisierungstechnik und der Software-Qualitätssicherung eingeführt.

Der Stand der Technik bei der komponentenbasierten Softwareentwicklung für Echtzeitsysteme und bei der Validierung von Software für Echtzeitsysteme wird in Kapitel 3 behandelt. Anhand festgelegter Kriterien werden die in der Literatur verfügbaren Lösungsansätze vorgestellt und bezüglich der Kriterien bewertet.

In Kapitel 4 wird ein Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme konzipiert. Dazu wird zunächst ein verbindlicher Entwicklungsprozess für die Entwicklung komponentenbasierter Software für Echtzeitsysteme festgelegt und verbleibende Fehlermöglichkeiten bei der komponentenbasierten Softwareentwicklung analysiert. Auf Basis dieser Ausgangssituation wird dann das Grundkonzept für ein Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme entwickelt.

Die Kapitel 5 bis 7 erläutern die einzelnen Bestandteile des konzipierten Prüfverfahrens. In Kapitel 5 wird auf die Spezifikation des geforderten Referenz-Verhaltens auf Basis von Anwendungsfällen eingegangen und in Kapitel 6 wird die Erstellung eines Simulationsmodells zur Simulation des Verhaltens der komponentenbasierten Software im Kontext des gesamten Echtzeitsystems vorgestellt. In Kapitel 7 wird das konzipierte Prüfverfahren zur Validierung des simulierten Verhaltens der komponentenbasierten Software gegenüber dem Referenz-Verhalten erläutert.

In Kapitel 8 wird die im Rahmen der Arbeit entwickelte Validierungsumgebung vorgestellt, welche die einzelnen Bestandteile des Prüfverfahrens mit prototypischen Werkzeugen unterstützt.

Die Anwendung des Prüfverfahrens wird beispielhaft anhand der Entwicklung einer komponentenbasierten Software für eine Kfz-Scheibenwischeranlage in Kapitel 9 demonstriert.

In Kapitel 10 werden abschließend die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf zukünftige Erweiterungen aufgezeigt.

2 Grundlagen

In der Literatur werden viele für diese Arbeit relevante Begriffe uneinheitlich verwendet. Deshalb werden in diesem Kapitel die für das Verständnis der Arbeit notwendigen Grundlagen und Begriffe aufgeführt und definiert. Im ersten Teil werden zunächst der Aufbau und die Echtzeiteigenschaften von Prozessautomatisierungssystemen erläutert sowie daraus resultierende Schwierigkeiten aufgezeigt. Anschließend werden im zweiten Teil dieses Kapitels wichtige Begriffe aus dem Gebiet der Software-Qualitätssicherung definiert.

2.1 Grundlagen und Definitionen aus dem Gebiet der Automatisierungstechnik

2.1.1 Aufbau von Prozessautomatisierungssystemen

Ein Prozessautomatisierungssystem ist ein Automatisierungssystem zur Automatisierung technischer Prozesse. Im Allgemeinen setzt sich ein Prozessautomatisierungssystem aus folgenden Bestandteilen zusammen:

- Benutzer
- Technischer Prozess
- Automatisierungsrechnersystem

Abbildung 2.1 zeigt die prinzipielle Struktur eines Prozessautomatisierungssystems [LaGö99].

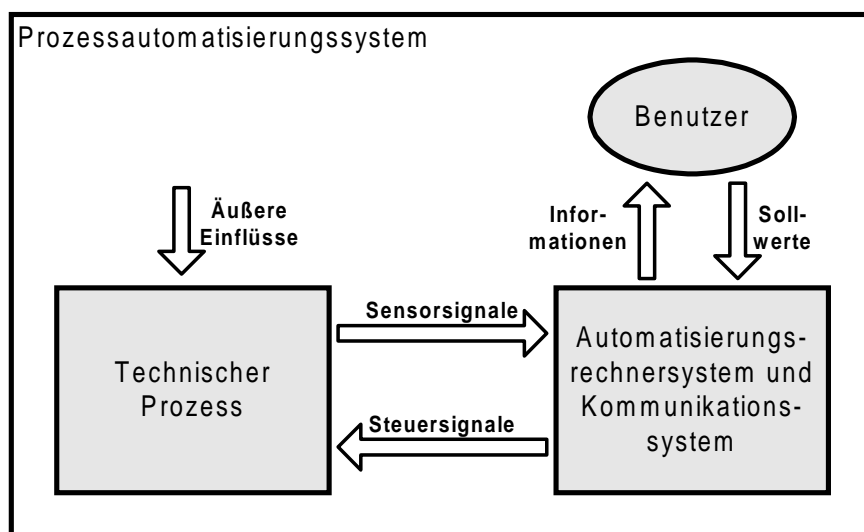


Abbildung 2.1: Prinzipielle Struktur eines Prozessautomatisierungssystems

Der Benutzer gibt dem Automatisierungsrechnersystem Sollwerte vor und wird von diesem mit Information versorgt. Auf den technischen Prozess wirken äußere Einflüsse aus der Umwelt, die indirekt auch durch den Benutzer beeinflusst werden können. Das Automatisierungsrechnersystem bekommt vom technischen Prozess Sensorsignale, verarbeitet diese und liefert Steuer-signale zurück.

Die heute in der Praxis verwendeten Automatisierungsrechnersysteme lassen sich in vier charakteristische Entwicklungslinien einteilen [LaGö99]:

- Speicherprogrammierbare Steuerungen (SPS)
- Steuergeräte (Mikrocontroller-basierte Ein-Chip-Computer)
- Personal Computer (PC) bzw. Industrie Personal Computer (IPC)
- Prozessleitsysteme (PLS)

Bei verteilten Automatisierungssystemen ist zusätzlich ein Kommunikationssystem erforderlich, das die verteilten Automatisierungsrechner miteinander verbindet. Abbildung 2.2 zeigt ein charakteristisches Beispiel für die Struktur eines verteilten Automatisierungssystems. Die Steuergeräte sind dabei über ein Bussystem miteinander verbunden und tauschen mit Hilfe der Automatisierungssoftware Informationen aus.

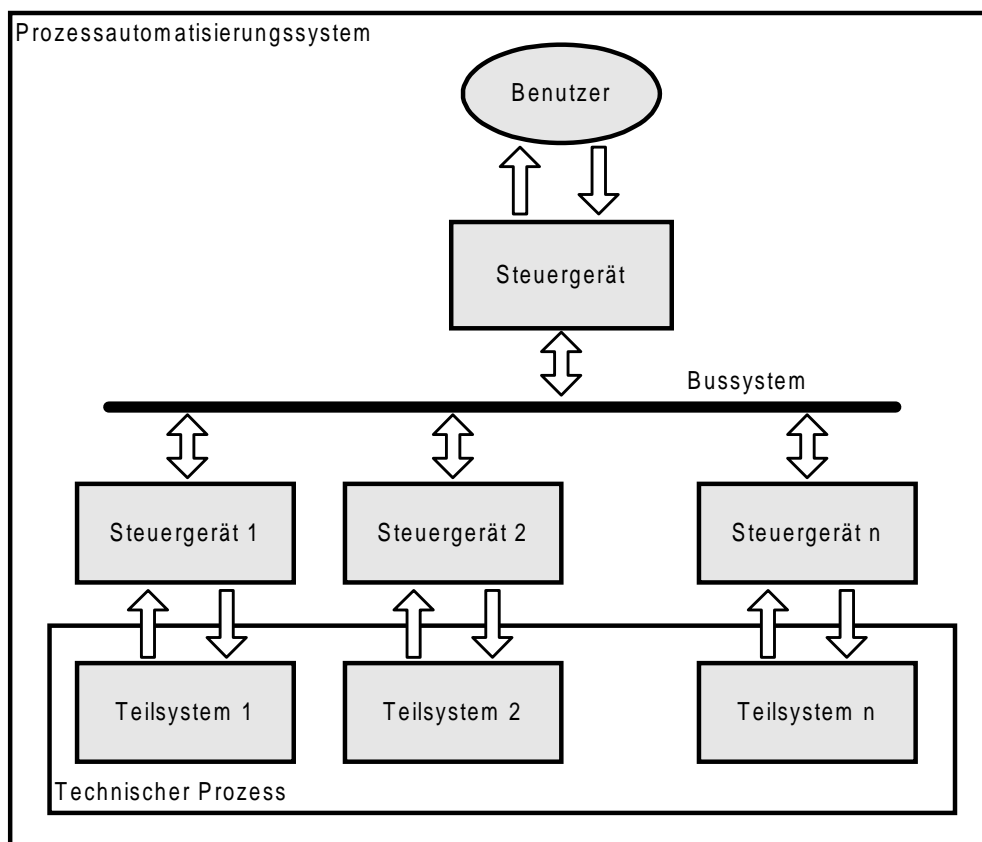


Abbildung 2.2: Beispiel für die Struktur eines verteilten Automatisierungssystems

Das Automatisierungsrechnersystem besteht aus einem Hardware-System und einem Automatisierungssoftware-System. Das Automatisierungssoftware-System wird im Folgenden als Automatisierungssoftware bezeichnet und wie folgt definiert:

Automatisierungssoftware:

Die Automatisierungssoftware ist die Menge von Programmen, die zum Betrieb eines Automatisierungsrechnersystems und zur Ausführung der Automatisierungsaufgaben erforderlich sind, inklusive ihrer Dokumentation.

Um eine sinnvolle Trennung zwischen ausführenden und organisatorischen bzw. verwaltenden Aufgabenbereichen zu erreichen, wird die Automatisierungssoftware im Allgemeinen in folgende zwei Bereiche aufgeteilt:

- Anwendungssoftware
- Systemsoftware

Anwendungssoftware:

Zur Anwendungssoftware werden alle Programme gezählt, die die Automatisierungsfunktionen ausführen, wie z.B. Messwerte einlesen sowie nach Regelungs- und Steuerungsalgorithmen Stellgrößen berechnen und ausgeben.

An Stelle von Anwendungssoftware wird in der Literatur auch die angelsächsisch geprägte Bezeichnung Applikationssoftware (engl. application software) verwendet.

Systemsoftware:

Zur Systemsoftware werden alle Programme gezählt, die die Ausführung der Anwendungsprogramme steuern, verwalten und unterstützen, wie z.B. Betriebssysteme oder Treiberprogramme. Sie sind universell einsetzbar und werden in der Praxis oft von Herstellern eines Automatisierungsrechnersystems mit entwickelt.

2.1.2 Echtzeiteigenschaften von Prozessautomatisierungssystemen

Prozessautomatisierungssysteme sind Echtzeitsysteme. In Anlehnung an [HHL93, LaGö99, Stan92] ist ein Echtzeitsystem wie folgt definiert:

Echtzeitsystem:

Als Echtzeitsysteme (engl. real-time systems) werden Rechnersysteme bezeichnet, die mit externen technischen Systemen in einer Wechselwirkung stehen und bei denen die Verarbeitung der Programme zeitlich mit den in externen Systemen ablaufenden Vorgängen Schritt halten muss.

Echtzeitsysteme besitzen als charakteristische Eigenschaft den Echtzeitbetrieb. Die Definition des Echtzeitbetriebs wird aus [DIN44300] übernommen.

Echtzeitbetrieb:

Der Echtzeitbetrieb ist der Betrieb eines Rechnersystems, bei dem Programme zur Bearbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zufälligen, zeitlichen Verteilung oder zu bestimmten Zeitpunkten auftreten.

Die wesentlichen Anforderungen an Echtzeitsysteme in der Prozessautomatisierung sind nach [LaGö99]:

- Rechtzeitigkeit
- Gleichzeitigkeit
- Verlässlichkeit
- Vorhersehbarkeit

Die Forderung nach Rechtzeitigkeit bedeutet, dass die Erfassung, Verarbeitung und Ausgabe von Prozessdaten sowie daraus abgeleitete Reaktionen pünktlich ausgeführt werden müssen, das heißt nicht zu früh und nicht zu spät. Dabei steht nicht die Schnelligkeit der Bearbeitung im Vordergrund, sondern die Reaktion innerhalb vorgegebener und vorhersagbarer Zeitschranken. Die funktionale Korrektheit der Informationsverarbeitung eines Echtzeitsystems hängt nicht nur vom Resultat der Berechnung, sondern auch davon ab, wann dieses Resultat produziert und ausgegeben wird.

Die Forderung nach Gleichzeitigkeit besagt, dass im technischen Prozess mehrere Vorgänge (Teilprozesse) gleichzeitig zu steuern sind und dass das Automatisierungssystem dieser Anforderung durch die nebenläufige Ausführung von Programmen entsprechen muss.

Die Forderung nach Verlässlichkeit ist von besonderer Bedeutung bei solchen Prozessautomatisierungssystemen, bei denen ein Ausfall des Automatisierungssystems oder des Kommunikationssystems zu einer Gefährdung von großen Sachwerten oder gar von Menschenleben führen würde. Im letzteren Fall, den sogenannten sicherheitsrelevanten Prozessautomatisierungssystemen, muss zusätzlich gewährleistet sein, dass bei einem Hardware- oder Softwarefehler das Prozessautomatisierungssystem in einen sicheren Zustand überführt wird.

Die Forderung nach Vorhersehbarkeit besagt, dass alle durch das Automatisierungssystem auszuführenden Reaktionen exakt geplant und deterministisch sein müssen. Insbesondere bei der Automatisierung sicherheitskritischer technischer Prozesse muss das Verhalten des gekoppelten Echtzeitsystems auch bei Überlast- und Fehlersituationen vorhersehbar sein.

Neben der Bezeichnung Echtzeitsystem wird in der Informatik auch die Bezeichnung „Reaktives System“ verwendet, was zum Ausdruck bringt, dass das Automatisierungssystem auf Informationen von außen reagieren muss.

2.2 Grundlagen und Definitionen aus dem Gebiet der Software-Qualitätssicherung

2.2.1 Software-Qualität

In der Einleitung wurde aufgezeigt, dass die Qualität der Software bei der Entwicklung von Echtzeitsystemen mittlerweile eine entscheidende Rolle einnimmt. Zur Definition des Begriffs Software-Qualität werden die Ausführungen der [DIN12119, ISO9126-1] übernommen.

Software-Qualität:

Die Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen. Die Software-Qualitätsmerkmale sind im Einzelnen:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

2.2.2 Fehlermöglichkeiten bei der Softwareentwicklung

Eine wichtige Zielsetzung der Software-Qualitätssicherung ist die Aufdeckung von Fehlern. Dazu ist es notwendig zu wissen, was ein Fehler ist und wo Fehler bei der Softwareentwicklung für Echtzeitsysteme entstehen können. Die nachfolgenden Definitionen basieren auf den Quellen [Balz98, BeEG86, DIN12119, DIN40041, IEEE610.12, ISO9126-1, LaGö99, Ligg93, Ligg00, PoRi96].

Fehler:

Ein Fehler ist

- jede Abweichung von den Anforderungen des Auftraggebers,
- jede Inkonsistenz in den Anforderungen,
- jede Abweichung der tatsächlichen Ausprägung eines Qualitätsmerkmals von der vorgesehenen Soll-Ausprägung,
- jede Inkonsistenz zwischen der Spezifikation und der Implementierung und
- jedes strukturelle Merkmal eines Quellcodes, das ein fehlerhaftes Verhalten des Programms verursacht.

Ein Fehler ist demnach nicht nur eine Abweichung vom geforderten funktionalen Umfang, sondern auch von den ausgewählten Qualitätsmerkmalen.

Abweichung:

Die Abweichung ist der Unterschied zwischen einem Merkmalswert oder dem Merkmal zugeordneten Wert und einem Bezugswert. Der geforderte Wert kann ein bestimmtes Verhalten der Software sein, die Abweichung damit ein vom geforderten Verhalten abweichendes Verhalten.

Da Software ein immaterielles Produkt ist, kann sie nicht durch „physikalische oder chemische Ausfallmechanismen“ fehlerhaft werden. Als Ursachen für Fehler in Software kommen nur „menschliche Fehlermechanismen“ in Frage, wie z.B.

- Denkfehler
- Verständnisfehler
- Kommunikationsfehler
- Interpretationsfehler
- Flüchtigkeitsfehler

Fehler in der Software sind immer inhärent, das heißt, sie wirken sich bereits bei der Entwicklung der Software aus. Bei Automatisierungssoftware werden sie in der Praxis jedoch oft erst bei der Inbetriebnahme oder während der Betriebsphase des Automatisierungssystems entdeckt. Zu den inhärenten Fehlerarten gehören z.B.

- Pflichtenheftfehler (Anforderungsfehler)
- Entwurfsfehler
- Programmierfehler
- Dokumentationsfehler

Der Begriff Mangel sollte mit äußerster Vorsicht benutzt werden. Die in [DIN8402] herausgestellte Unterscheidung zwischen Mangel und Fehler ist entscheidend, weil sie im Zusammenhang mit Produkthaftungsfragen rechtliche Bedeutung hat.

Mangel:

Einen Mangel weist nach § 459 BGB Abs. 2 [BGB459] eine Sache auf, wenn sie „mit Fehlern behaftet ist, die den Wert oder die Tauglichkeit zu dem gewöhnlichen oder dem nach dem Verträge vorausgesetzten Gebrauch aufheben oder mindern“.

Im Gegensatz zum Fehler hebt ein Mangel immer auf eine Beeinträchtigung der Verwendbarkeit der betrachteten Einheit ab.

2.2.3 Maßnahmen zur Software-Qualitätssicherung

Die Software-Qualitätssicherung verfolgt in Anlehnung an [Göhn01, Lude97] zwei wesentliche Ziele:

- Unmittelbar: Vertrauen in ein Produkt erhöhen
- Mittelbar: Qualitätsniveau erhöhen

Die Maßnahmen zur Software-Qualitätssicherung können in

- konstruktive Maßnahmen und
- analytische Maßnahmen

unterteilt werden. Konstruktive Maßnahmen wirken sich direkt auf die Qualität bei der Softwareentwicklung aus, während analytische Maßnahmen prüfenden Charakter haben. Konstruktive und analytische Maßnahmen ergänzen sich gegenseitig. In Anlehnung an [Ligg90, Stra97, BMI92, Wall90] ergeben sich nachfolgende Definitionen.

Konstruktive Qualitätssicherung:

Die konstruktive Qualitätssicherung umfasst alle Maßnahmen und Hilfsmittel, die im Laufe der Software-Entwicklungsphasen eingesetzt werden, um Fehler zu vermeiden.

Analytische Qualitätssicherung:

Die analytische Qualitätssicherung umfasst alle Maßnahmen und Hilfsmittel, die zum Entdecken und Beheben von Fehlern in bereits erstellten (Teil-)Produkten der Software-Entwicklungsphasen eingesetzt werden.

Bei den Qualitätssicherungsmaßnahmen kann außerdem in Entwicklungsprozess-bezogene und Produkt-bezogene Maßnahmen unterschieden werden. Tabelle 2.1 enthält einige Beispiele für Qualitätssicherungsmaßnahmen.

Tabelle 2.1: Beispiele für Qualitätssicherungsmaßnahmen

	Konstruktive Maßnahmen	Analytische Maßnahmen
Entwicklungsprozess-bezogen	<ul style="list-style-type: none"> • Vorgehensmodelle (z.B. V-Modell) • Standards und Normen (z.B. ISO9000, CMM) 	<ul style="list-style-type: none"> • Audits • Metriken • Zertifizierungen
Produkt-bezogen	<ul style="list-style-type: none"> • Entwicklungsmethoden (z.B. OOA, OOD) • Standards und Normen (z.B. IEC61508) • CASE-Werkzeuge (z.B. ASCET-SD, Rose) 	<ul style="list-style-type: none"> • Reviews • Testen • Simulationen • Formale Verifikationen • Rapid Prototyping

Für die vorliegende Arbeit sind vorrangig die Maßnahmen und Hilfsmittel zur analytischen Qualitätssicherung von Software-Produkten von Interesse, da diese helfen Fehler aufzudecken. Auf die dazu erforderlichen Grundlagen wird in den nächsten Abschnitten ausführlicher eingegangen.

2.2.4 Prüfverfahren zur Software-Qualitätssicherung

Die Aktivitäten der analytischen Qualitätssicherung basieren allesamt auf dem Prinzip der Nachweisführung. Die Nachweisführung zeigt, dass ein Prüfgegenstand alle an ihn gestellten Anforderungen erfüllt. Die Nachweisführung muss so durchgeführt werden, dass sie objektiv nachzuvollziehen ist. Die Aktivität bei der Nachweisführung ist die Prüfung [PoRi96].

Prüfung:

Die Prüfung ist eine Tätigkeit, wie Messen oder Untersuchen von einem oder mehreren Merkmalen eines Prüfgegenstands sowie das Vergleichen mit in einer Referenz festgelegten Anforderungen, um festzustellen, ob Konformität für jedes Merkmal erzielt ist.

Im Allgemeinen ist die Prüfung auf ein bestimmtes Ziel abgestimmt. Sie zielt dann auf die Nachweisführung einzelner Qualitätsmerkmale ab, wie z.B. Funktionsprüfung, Zuverlässigkeitsprüfung oder Benutzbarkeitsprüfung.

Für die Prüfung stehen zwei unterschiedliche Prinzipien zur Verfügung:

- Stichproben-Prinzip
- Logisches Nachweis-Prinzip

Beim Stichproben-Prinzip werden eine Referenz und der Prüfgegenstand unter gleichen Randbedingungen und Eingaben betrachtet. Liefern Referenz und Prüfgegenstand für alle gewählten Eingaben die gleichen Ausgaben bzw. Ergebnisse, so genügt der Prüfgegenstand seiner Referenz. Durch die stichprobenhafte Prüfung ist es bei komplexeren Prüfgegenständen schwierig Aussagen über die Vollständigkeit der Prüfung zu machen. Beispiele für Stichprobenbasierte Prüfverfahren sind Audit, Inspektion, Review, Testen und Simulation.

Beim logischen Nachweis-Prinzip werden die in der Referenz geforderten Eigenschaften am Prüfgegenstand aufgrund von Axiomen und Annahmen nachgewiesen. Der logische Nachweis erfolgt vollständig und allgemeingültig im Rahmen der Annahmen. Beispiele für Prüfverfahren basierend auf logischem Nachweis sind Symbolisches Testen und Formale Beweistechnik (Formale Verifikation).

Die Produkt-bezogenen Prüfverfahren für Software lassen sich in statische Prüfung und dynamische Prüfung klassifizieren. Auf Basis von [IEEE610.12] werden sie wie folgt definiert:

Statische Prüfung:

Die statische Prüfung ist die Prüfung eines Systems oder einer Komponente basierend auf deren Form, Struktur, Inhalt oder Dokumentation.

Dynamische Prüfung:

Die dynamische Prüfung ist die Prüfung eines Systems oder einer Komponente basierend auf deren Verhalten während der Ausführung.

Als Synonym für dynamische Prüfung werden in der Literatur oft auch die Begriffe „dynamische Analyse“ oder „dynamisches Testen“ verwendet. Abbildung 2.3 zeigt eine Übersicht über die wichtigsten Produkt-bezogenen Prüfverfahren zur Software-Qualitätssicherung.

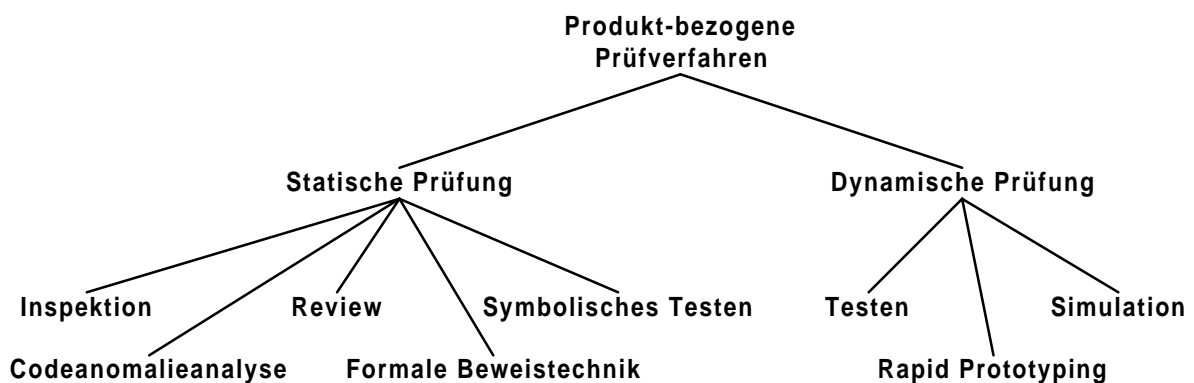


Abbildung 2.3: Übersicht über Produkt-bezogene Prüfverfahren zur Software-Qualitätssicherung

Bei der Qualitätssicherung von Software für Echtzeitsysteme ist das dynamische Verhalten der Software von besonderem Interesse, da neben der reinen Funktionalität auch Echtzeitanforderungen geprüft werden müssen. Bei der dynamischen Prüfung wird der Prüfgegenstand zur Ausführung gebracht und demonstriert dabei sein Verhalten. Deshalb ist die dynamische Prüfung beim Aufdecken von Fehlern im dynamischen Verhalten prinzipiell erfolgreicher als die statische Prüfung.

Im Folgenden werden die für diese Arbeit relevanten Prüfverfahren Testen und Simulation in Anlehnung an [Balz96, IEEE610.12, MaMe88, Schn91, Stra97] definiert. Das Testen ist das am häufigsten eingesetzte Prüfverfahren der dynamischen Prüfung.

Testen:

Das Testen bezeichnet die kontrollierte Ausführung eines Prüfgegenstands, z.B. ein Programm oder eine Funktion eines Programms, unter spezifizierten Bedingungen mit dem Ziel, Fehler zu entdecken. Bei der Ausführung werden die Ergebnisse beobachtet oder aufgezeichnet und anhand dieser eine Bewertung des Prüfgegenstands unter ausgewählten Prüfkriterien durchgeführt.

Das sogenannte „Debugging“, das heißt das Lokalisieren und Beheben bzw. Beseitigen eines entdeckten Fehlers, wird nicht zum Testen gerechnet, sondern als eine Tätigkeit gesehen, die sich an das Testen anschließt. Es gehört gleichwohl zur analytischen Qualitätssicherung.

Simulation:

Bei der Simulation wird ermittelt, wie ein vorgegebenes Simulationsmodell eines Systems auf bestimmte Eingaben reagiert. Das Simulationsmodell wird dabei zum Prüfgegenstand. Analog zum Testen kann das Simulationsmodell unter spezifizierten Bedingungen kontrolliert ausgeführt werden, mit dem Ziel, Fehler zu entdecken. Bei den Simulationsuntersuchungen werden die Ergebnisse beobachtet oder aufgezeichnet und anhand dieser eine Bewertung des Prüfgegenstands – in diesem Fall das Simulationsmodell – unter ausgewählten Prüfkriterien durchgeführt.

Der Unterschied zwischen Testen und Simulation von Software liegt darin, dass beim Testen die Implementierung der Software kontrolliert ausgeführt wird, wo hingegen bei der Simulation ein Simulationsmodell der Software kontrolliert ausgeführt wird. Zur Prüfung des Echtzeitverhaltens muss beim Testen die Software auf dem Automatisierungsrechner selbst ausgeführt werden, bei der Simulation hingegen muss das dynamische Verhalten der Software durch ein Simulationsmodell auf einem Entwicklungsrechner realitätsnah nachgebildet werden.

Die bei der Simulation häufig angebotene Animation des Verhaltens ermöglicht eine visuelle Validierung des beobachteten dynamischen Verhaltens.

2.2.5 Grundsätzliche Vorgehensweise bei der dynamischen Prüfung von Software

Für eine dynamische Prüfung von Software basierend auf Testen oder Simulation müssen prinzipiell die folgenden wesentlichen Schritte durchgeführt werden:

- (1) Prüffallentwicklung
- (2) Erstellung der Prüfumgebung
- (3) Stimulation des Prüfgegenstands
- (4) Beobachtung des Prüfgegenstands
- (5) Bewertung

Die sicherlich schwierigste Aktivität bei der Prüfung ist die Prüffallentwicklung (1), da sie ein kreativer Prozess ist. Bei der Prüffallentwicklung werden geeignete Stichproben ausgewählt, das heißt, es werden geeignete Prüffälle zur Erreichung der Prüfziele entwickelt. Die Definition des Begriffs Prüffall erfolgt in Anlehnung an [BMI92, PoRi96].

Prüffall:

Ein Prüffall beschreibt,

- welche Funktion des Prüfgegenstands mit welcher Genauigkeit zu prüfen ist,
- welche Ausgangssituation hierfür erforderlich ist,
- welche stimulierenden Referenz-Eingaben (Daten und Signale mit allen für die Prüfung ausschlaggebenden Eigenschaften wie Zeitbedingungen) notwendig sind und
- welche Referenz-Ergebnisse (Ausgabedaten und Reaktionen/Effekte) zu erwarten sind.

Beim Einsatz der zuvor vorgestellten Prüfverfahren werden grundsätzlich folgende zwei Prüfziele verfolgt:

- Fehlerfindung im Prüfgegenstand
- Nachweis¹, dass ein Prüfgegenstand die in einer Referenz festgelegten Anforderungen erfüllt

Mit Prüfstrategien werden Handlungsweisen und Auswahlkriterien bereit gestellt, mit deren Hilfe abhängig vom Prüfziel geeignete Prüffälle hergeleitet und entwickelt werden können. Die Informationsquelle (Referenz), aus welcher Prüffälle für Programme hergeleitet werden, stellt nach [FLS95] einen wichtigen Ausgangspunkt für die Prüfstrategien dar: „Der Programmierer hat drei Informationsquellen für die Auswahl der Prüfdaten: Das Programm, seine Spezifikation und seine Kenntnis der häufigsten Programmfehler.“

Die Prüffallentwicklung (1) für Software kann daher prinzipiell in drei Arten eingeteilt werden:

- Programmstrukturbasierte Prüffallentwicklung (White-Box)
- Spezifikationsbasierte Prüffallentwicklung (Black-Box)
- Wissensbasierte Prüffallentwicklung

Die programmstrukturbasierte Prüffallentwicklung [Balz98, FLS95, Myer91] orientiert sich an den Programmstrukturelementen, das heißt, die Prüffälle werden z.B. aus der Kenntnis der Pfade, Anweisungen, Bedingungen oder Datenstrukturen des Quellcodes hergeleitet.

Bei der spezifikationsbasierten Prüffallentwicklung [Balz98, FLS95, Myer91] hingegen werden die Prüffälle aus Spezifikations-Dokumenten, z.B. aus einer Anforderungsspezifikation oder Entwurfsspezifikation, ohne notwendige Kenntnis der Programmstruktur hergeleitet.

Bei der wissensbasierten Prüffallentwicklung [Ligg93, Ligg00] werden die Prüffälle unter Kenntnis der speziellen Fehlermöglichkeiten eines spezifischen Problembereichs entwickelt. Das dazu notwendige Wissen kann beispielsweise aus den Ergebnissen einer Sicherheitsanalyse stammen.

Bei der Erstellung der Prüfumgebung (2) wird eine geeignete Prüfinstrumentierung für die

¹ Beim Testen bzw. bei der Simulation findet kein mathematisch korrekter Nachweis statt, sondern ein bedingt gültiger Umkehrschluss, dass durch das Nicht-Auffinden von Fehlern die Anforderungen erfüllt sind.

Stimulation (3) und Beobachtung (4) des Prüfgegenstands zur Ausführung der Prüffälle aufgebaut. Vergleichbar mit einer messtechnischen Instrumentierung in physikalischen Experimenten muss sie das Anlegen von Stimulations-Daten sowie die Beobachtung und Aufzeichnung von Daten an den Schnittstellen des ausgeführten Prüfgegenstands ermöglichen.

Bei der abschließenden Bewertung (5) werden die beobachteten Ergebnisse mit den Referenz-Ergebnissen verglichen.

Bei komplexeren Prüfgegenständen ist es schwierig, Aussagen über die Vollständigkeit der Prüfung zu machen, da die entwickelten Prüffälle nur Stichproben sind. Zur Bewertung der Vollständigkeit der Prüfung existieren Vollständigkeits-Metriken, die sich an der Art der Prüffallentwicklung orientieren.

Orientiert sich die Prüffallentwicklung an den Programmstrukturen der Implementierung, existieren als Vollständigkeits-Metriken sogenannte Testabdeckungskriterien. Man unterscheidet zwischen den Testabdeckungen C0 bis C7 [Balz98, BrDr95]. Die gebräuchlichsten Testabdeckungen sind C0, C1 und C2:

- C0 Jede Anweisung wird mindestens einmal durchlaufen.
- C1 Jeder Programmzweig wird mindestens einmal durchlaufen.
- C2 Bei zusammengesetzten Bedingungen in bedingten Anweisungen wird jede Bedingung (wahr, falsch) getestet.

Orientiert sich die Prüffallentwicklung an der Spezifikation, orientieren sich auch die Vollständigkeits-Metriken daran. Bei einer Spezifikation auf Basis von Zustandsautomaten [Bind99, LiRü96] müssen die Prüffälle beispielsweise alle Zustände und Zustandsübergänge vollständig abdecken.

Bei der wissensbasierten Prüffallentwicklung orientieren sich die Vollständigkeits-Metriken in der Regel ebenfalls an der Spezifikation.

2.2.6 Validierung und Verifikation von Software

Die Softwareentwicklung wird üblicherweise in verschiedene Entwicklungsphasen eingeteilt. Für diese Einteilung existieren eine Vielzahl von unterschiedlichen Entwicklungsprozess-Modellen [Balz98]. An dieser Stelle wird nur das bundeseinheitliche Vorgehensmodell² V-Modell [BrDr95, BMI92] für Entwicklungen im Bereich der Informationstechnik vorgestellt, welches dieser Arbeit zugrunde gelegt wird. Das V-Modell besteht aus den 4 Submodellen Softwareerstellung, Qualitätssicherung, Konfigurationsmanagement und Projektmanagement.

² Das V-Modell wurde speziell für die Entwicklung von Informationssystemen und eingebetteten Systemen (engl. embedded systems) entwickelt [BrDr95]. Ein eingebettetes System ist eine spezielle Art von Prozessautomatisierungssystem, das dadurch gekennzeichnet ist, dass das Automatisierungssystem im technischen Prozess integriert und damit für den Benutzer nicht direkt sichtbar ist.

Die im Submodell Qualitätssicherung festgelegten analytischen Maßnahmen gewährleisten dabei die Erfüllung der Qualitätsanforderungen. Die anderen 3 Submodelle unterstützen durch konstruktive und organisatorische Maßnahmen die Vermeidung von Fehlern bei der Entwicklung. Abbildung 2.4 zeigt die Entwicklungsphasen und die vorgesehenen Prüfkativitäten zwischen den Teilprodukten der Softwareentwicklung.

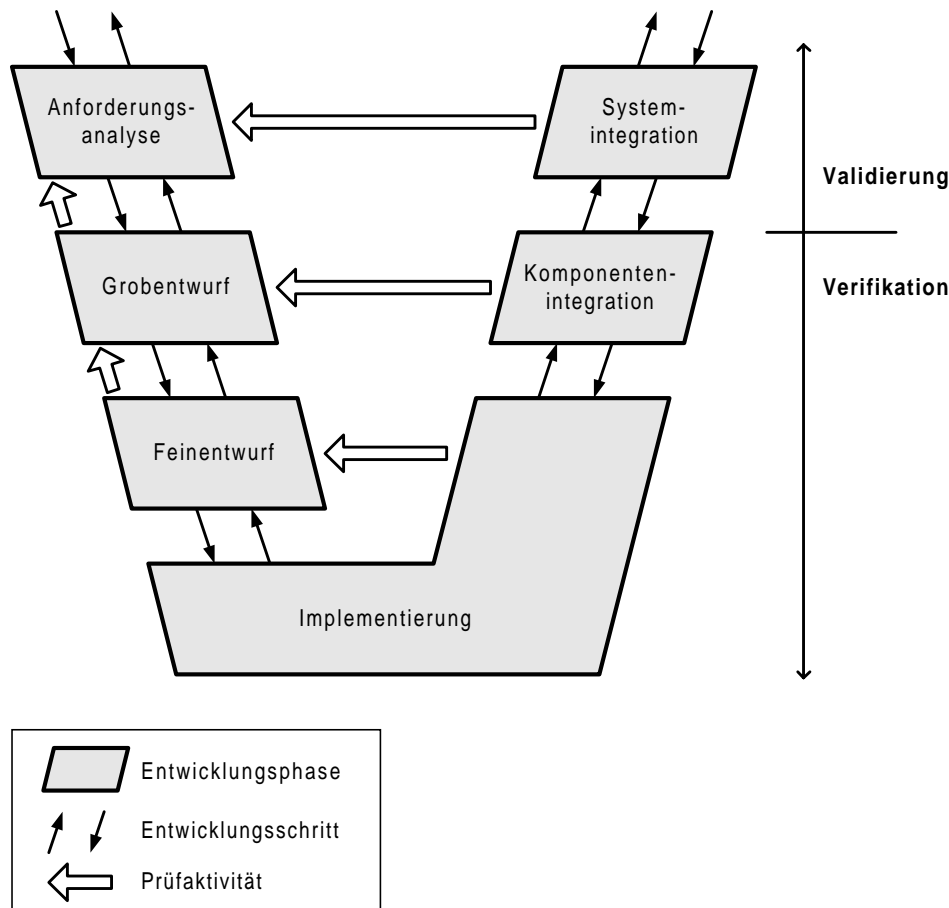


Abbildung 2.4: V-Modell zur Softwareentwicklung

Die ergänzenden Entwicklungsphasen für die (Echtzeit)-Systementwicklung im V-Modell wurden in Abbildung 2.4 zur Vereinfachung bewusst weggelassen. In der Praxis werden die einzelnen Entwicklungsphasen bei der Softwareentwicklung mehrfach durchlaufen. Die Pfeile zwischen den Entwicklungsphasen zeigen in beide Richtungen, was die möglichen iterativen Rückschritte darstellt. Das V-Modell integriert die Qualitätssicherung in das Wasserfall-Modell nach Boehm [Boeh81, Boeh84]. Durch die V-förmige Anordnung der Entwicklungsphasen mit ihren jeweiligen Teilprodukten stehen sich die korrespondierenden Phasen vor und nach der Implementierung gegenüber. Auf diese Weise kann aus der Abbildung abgelesen werden, anhand welcher Dokumente die Teilprodukte im rechten Teil des V-Modells zu prüfen sind.

Die Validierung und Verifikation der Teilprodukte der einzelnen Entwicklungsphasen sind Bestandteile des V-Modells. Mit Validierung und Verifikation werden qualitätssichernde,

prüfende Aktivitäten während der Produktentwicklung bezeichnet. Sie unterscheiden sich vor allem durch das Ziel der Prüfung. Im Zusammenhang mit der Entwicklung des Spiralmodells bringt Boehm [Boeh84] den Unterschied zwischen den beiden Aktivitäten auf folgenden einfachen Nenner:

- Validierung: „Am I building the right product?“
- Verifikation: „Am I building the product right?“

Für die vorliegende Arbeit werden die Definitionen aus dem V-Modell übernommen [BMI92].

Validierung:

Die Validierung ist die Nachweisführung, bei der gezeigt wird, dass ein Produkt die Erwartungen des Anwenders erfüllt. Eine Validierung kann durchgeführt werden

- in Form einer Prüfung des Endprodukts gegen die Erwartung des Anwenders,
- in Form einer Simulation auf der Basis von Zwischenergebnissen, bei der der Anwender einbezogen wird.

Verifikation:

Die Verifikation ist die Nachweisführung, bei der gezeigt wird, dass ein Produkt die Anforderungen erfüllt, die während vorhergehender Entwicklungsphasen erstellt wurden.

Für Validierung und Verifikation können zur Nachweisführung prinzipiell alle Prüfverfahren verwendet werden, die geeignet sind, die Ziele der qualitätssichernden Aktivitäten zu erreichen.

In diesem Kapitel wurden die für das Verständnis der vorliegenden Arbeit relevanten Grundlagen und Begriffe definiert. Dazu wurde der prinzipielle Aufbau von Prozessautomatisierungssystemen sowie die Echtzeiteigenschaften von Automatisierungssystemen erläutert. Weiterhin wurde eine Übersicht über die Software-Qualitätssicherung und die dazu etablierten Maßnahmen und Prüfverfahren vermittelt. Abschließend wurden die Begriffe Validierung und Verifikation im Zusammenhang mit dem V-Modell der Softwareentwicklung definiert. Ausgehend von diesen Grundlagen und der im Abschnitt 1.3 formulierten Zielsetzung der Arbeit wird im nächsten Kapitel der Stand der Technik bei der komponentenbasierten Entwicklung und Validierung von Software für Echtzeitsysteme analysiert und bewertet.

3 Stand der Technik

Das Ziel der vorliegenden Arbeit ist die Entwicklung eines Prüfverfahrens zur Validierung komponentenbasierter Software für Echtzeitsysteme. Deshalb werden in diesem Kapitel verfügbare Lösungsansätze im Bereich der komponentenbasierten Softwareentwicklung für Echtzeitsysteme und der Wissensstand bei der Validierung von Software für Echtzeitsysteme untersucht. Für beide Bereiche werden jeweils zunächst Anforderungskriterien aufgestellt, mit denen anschließend die in der Literatur verfügbaren Lösungsansätze bewertet werden.

3.1 Stand der Technik bei der komponentenbasierten Softwareentwicklung für Echtzeitsysteme

3.1.1 Anforderungen an die komponentenbasierte Softwareentwicklung für Echtzeitsysteme

Die Motivation für die Entwicklung von Software aus Komponenten ist dieselbe wie in anderen Ingenieurdisziplinen. Durch die Verwendung vorgefertigter, gezielt für die Mehrfachverwendung entwickelter Komponenten sollen nach dem Baukastenprinzip neue Anwendungen schneller, kostengünstiger und qualitativ besser entwickelt werden. Die dazu eingesetzten, mehrfach verwendbaren Komponenten werden bei der Komponentenentwicklung bereits umfassend geprüft und zeichnen sich wegen der Mehrfachverwendung durch eine große Betriebsbewährtheit aus. Bei der Implementierung von Software aus Komponenten können daher nicht mehr in jeder Programmzeile beliebig viele Implementierungsfehler gemacht werden, was die Fehlermöglichkeiten stark reduziert. Die Anforderungen an eine Software-Komponente werden von [Szyp98] wie folgt beschrieben: Eine Software-Komponente muss eine in sich abgeschlossene funktionale Einheit mit vollständig spezifizierten Schnittstellen bilden, sie muss gut dokumentiert, qualitativ hochwertig und unverändert mehrfach verwendbar sein. Zusätzlich soll eine Software-Komponente unabhängig von anderen Software-Komponenten sein, an eine gegebene Aufgabenstellung anpassbar und bezüglich bestimmter Eigenschaften konfigurierbar sein. Unter einer Software-Komponente versteht man nicht zwangsläufig ein Stück ausführbarer Code. Nach [Göhn98] kann eine Software-Komponente auch eine Beschreibung sein, aus der Code generiert werden kann, bzw. jedes wesentliche Ergebnis des Softwareentwicklungsprozesses, wie z.B. eine Entwurfsspezifikation oder Testfälle. Zur Vereinfachung wird im weiteren Verlauf der vorliegenden Arbeit eine Software-Komponente nur noch als Komponente bezeichnet.

Bei Echtzeitsystemen sind im Gegensatz zu den transaktionsorientierten Informationssystemen zusätzliche Anforderungen einzuhalten, wie z.B. vorhersehbares, zuverlässiges zeitliches Verhalten und geringer Ressourcenbedarf. Auf Grund dieser zusätzlichen Anforderungen sind die bekannten, teilweise standardisierten Komponententechnologien aus der transaktionsorientierten Datenverarbeitung, wie z.B. die „Common Object Request Broker Architecture“ (CORBA) [OMG01], „Microsoft - Component Object Model“ (MS-COM) [Micro95] oder JavaBeans [GJSB00, Sun00], in der Praxis nicht unmittelbar für die Entwicklung komponentenbasierter Software für Echtzeitsysteme anwendbar. Mehrfach verwendbare Komponenten werden bei diesen Komponententechnologien auf der Implementierungsebene verwendet. Komponenten in Codeform haben aber den Nachteil, dass eine umfangreiche Funktionalität gleichzeitig auch entsprechende Ressourcen zur Laufzeit benötigt.

Aus diesem Grund werden zur komponentenbasierten Softwareentwicklung für Echtzeitsysteme spezialisierte Softwareentwicklungsmethoden benötigt, die den besonderen Anforderungen von Echtzeitsystemen Rechnung tragen. Leistungsfähige komponentenbasierte Softwareentwicklungsmethoden sollten deshalb Vorgehensweisen unterstützen, bei denen mehrfach verwendbare Komponenten bereits auf der Entwurfsebene eingesetzt werden. Die Komponenten sollten durch Konfigurierung von Parametern an die gegebene Aufgabenstellung angepasst und bei der anschließenden Implementierung noch optimiert werden können. Zur effizienten Optimierung bietet sich der Einsatz von automatischen Codegenerierungswerkzeugen an.

3.1.2 Anforderungen an Beschreibungsmittel zur Entwurfsspezifikation komponentenbasierter Software für Echtzeitsysteme

Damit der Einsatz mehrfach verwendbarer Komponenten bereits in der Entwurfsphase stattfinden kann, müssen zur Erstellung der Entwurfsspezifikation einer komponentenbasierten Software geeignete Beschreibungsmittel zur Verfügung stehen. Mit den Beschreibungsmitteln müssen grundsätzlich die folgenden wesentlichen Bestandteile einer komponentenbasierten Software spezifiziert werden können [Flei99]:

- Komponentenverhalten
- Strukturmodell
- Verteilungsmodell
- Kommunikationsmodell
- Ausführungsmodell

Die Anforderungen an die einzelnen Bestandteile werden im Folgenden näher beschrieben.

Komponentenverhalten

Eine Komponente darf nur über die Eingänge und Ausgänge ihrer Schnittstelle mit anderen Komponenten verbunden werden und darüber Informationen austauschen. Sowohl das Komponentenverhalten als auch die Struktur der Komponenten-Schnittstelle können über Parameter konfigurierbar sein. Bei der Instanziierung werden alle Parameter auf einen konkreten Wert eingestellt. Ein Beispiel für die von außen sichtbaren, konfigurierbaren Parameter und Attribute einer Komponente ist in Tabelle 3.1 dargestellt.

Tabelle 3.1: Beispiel für die konfigurierbaren Parameter und Attribute einer Komponente

<Komponententyp>			
Parameter	Typ	Bedeutung / Codierung	Konfigurierter Wert
Instanzname	-	-	<Name>
<Parameter 1>	<Typ 1>	<Beschreibung 1, z.B. zulässiger Wertebereich>	<Wert 1>
...
Ereignis-Eingänge		Bedeutung	
<Ereignis EE.1>		<Beschreibung EE.1>	
...		...	
Ereignis-Ausgänge		Bedeutung	
<Ereignis-EA.1>		<Beschreibung EA.1>	
...		...	
Daten-Eingänge		Typ	Bedeutung / Codierung
<Datum DE.1>		<Typ DE.1>	<Beschreibung DE.1>
...	
Daten-Ausgänge		Typ	Bedeutung / Codierung
<Datum DA.1>		<Typ DA.1>	<Beschreibung DA.1>
...	

Jede Komponente besitzt mindestens den elementaren Parameter Instanznamen, der bei der Instanziierung vergeben werden muss. Die Eingänge und Ausgänge der Schnittstelle werden in diesem Beispiel in die Bereiche Ereignisse und Daten unterschieden.

Das gekapselte innere Komponentenverhalten kann beliebig spezifiziert werden. In der Praxis wird das Komponentenverhalten beispielsweise oftmals mit Hilfe von Zustandsautomaten spezifiziert.

Strukturmodell

Die Komposition einer Menge von Komponenten zu einem Softwaresystem wird mit Hilfe eines Strukturmodells spezifiziert. Bei der Komposition werden die Eingänge und Ausgänge der Komponenten miteinander verbunden. Für eine übersichtlichere Darstellung lassen sich eine Menge von Komponenten zu einer Baugruppe zusammenfassen. Dazu werden alle Eingänge und Ausgänge, die für Komponenten außerhalb der Baugruppe sichtbar sein sollen, als Schnittstelle einer Baugruppe zusammengefasst. Auf diese Weise kann eine hierarchische Darstellung des Strukturmodells realisiert werden. Abbildung 3.1 zeigt ein Beispiel für ein Strukturmodell einer komponentenbasierten Software.

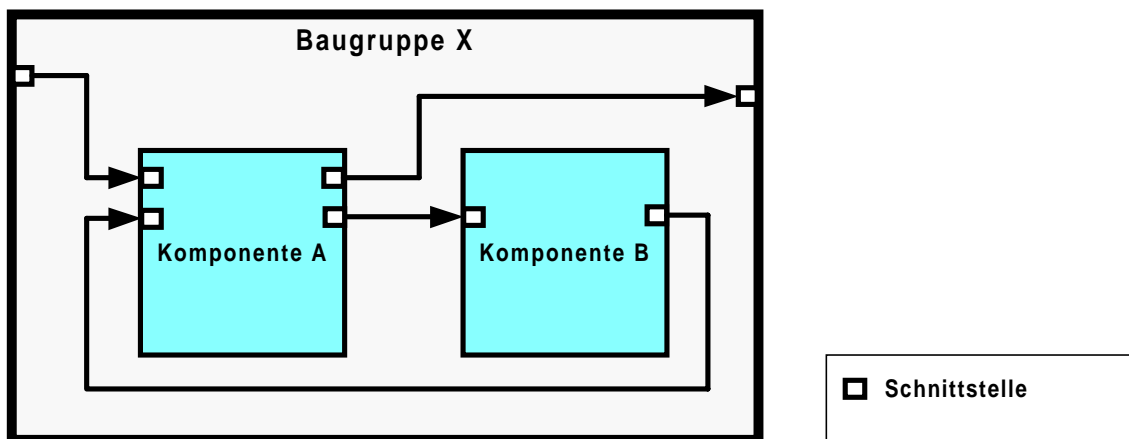


Abbildung 3.1: Beispiel für ein Strukturmodell einer komponentenbasierten Software

An dieser Stelle zeigt sich auch der wesentliche Unterschied zu objektorientierter Software. Abbildung 3.2 zeigt das Strukturmodell einer objektorientierten Software.

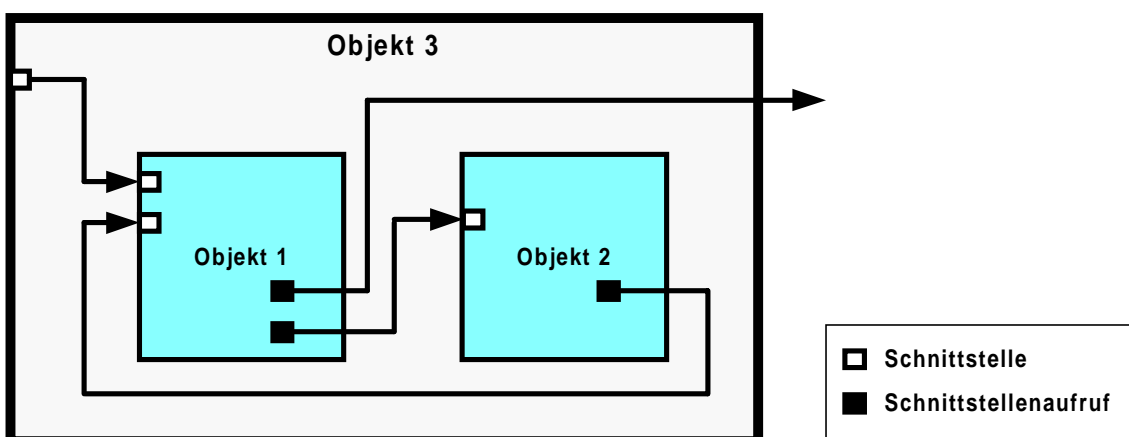


Abbildung 3.2: Beispiel für ein Strukturmodell einer objektorientierten Software

Bei der objektorientierten Struktur greift ein Objekt auf ein zweites Objekt zu, indem es von seinem inneren Kontrollfluss direkt auf eine Operation der Schnittstelle eines anderen Objekts zugreift. Soll beispielsweise Objekt1 aus Abbildung 3.2 in einer anderen Anwendung wiederverwendet werden, so müssen alle Zugriffe auf die Schnittstellen anderer Objekte innerhalb des Kontrollflusses von Objekt1 verändert werden. Die notwendigen Änderungen im Innern der Verhaltensbeschreibung von Objekt1 zeigen, dass Objekte bzw. Klassen prinzipiell schlechter mehrfach verwendet werden können als Komponenten.

Bei einer komponentenbasierten Struktur (siehe Abbildung 3.1) hingegen ist der innere Kontrollfluss direkt nur mit der eigenen Schnittstelle verbunden, wodurch wirkliche Black-Box-Komponenten zur Mehrfachverwendung bereit stehen. Die innere Verhaltensbeschreibung von Komponenten ist im Gegensatz zu Objekten völlig unabhängig vom Strukturmodell der Software. Die Verbindung zwischen Komponenten wird im Strukturmodell ausschließlich über die Verbindungen zwischen den Schnittstellen der Komponenten realisiert.

Verteilungsmodell

Werden die Komponenten auf mehrere Automatisierungsrechner verteilt, so ist zur Spezifikation ein Verteilungsmodell notwendig. Abbildung 3.3 zeigt ein Beispiel für ein Verteilungsmodell.

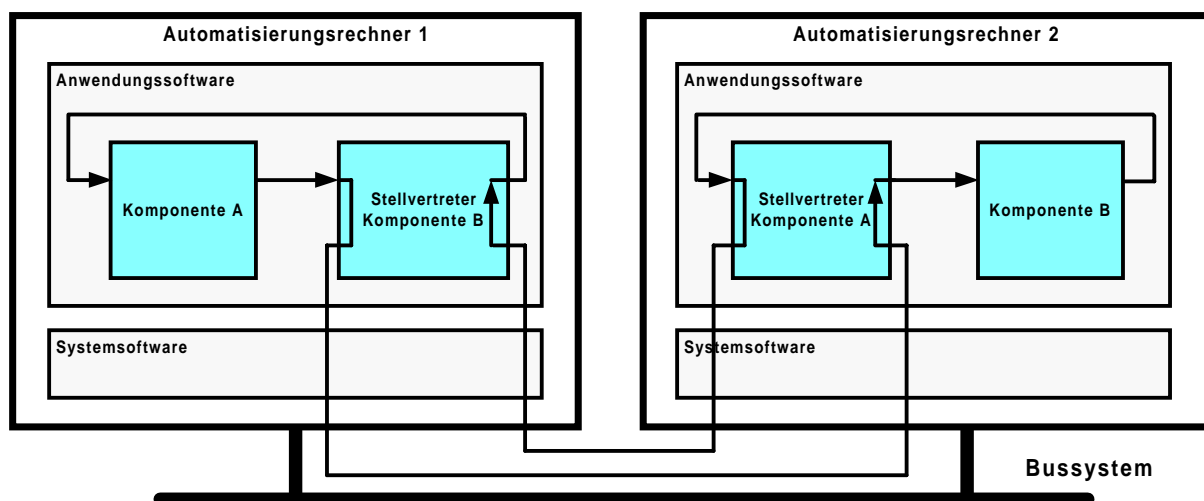


Abbildung 3.3: Beispiel für ein Verteilungsmodell

Im Verteilungsmodell wird die physische Struktur der verteilten Automatisierungsrechner und das bestehende Kommunikationssystem modelliert. Die Komponenten aus dem Strukturmodell werden im Verteilungsmodell je einem Automatisierungsrechner zugeordnet. Die komponentenbasierte Software eines Automatisierungsrechners kann dabei als eine Baugruppe aufgefasst werden. In dem Beispiel von Abbildung 3.3 wird für die Kommunikation zwischen verteilten Komponenten das in der Praxis sehr häufig verwendete Stellvertreter-Muster (engl. proxy pattern) [GHJV95] verwendet. Die dabei zusätzlich eingefügten Stellvertreter-Komponenten exportieren die Schnittstelle einer Komponente auf andere Automatisierungsrechner. Die

Stellvertreter-Komponenten realisieren das Senden und Empfangen von Nachrichten zwischen Komponenten auf verteilten Automatisierungsrechnern über ein Bussystem unter Verwendung zentraler Systemsoftware, wie z.B. Bussystem-Treiber-Komponenten.

Kommunikationsmodell

Das Kommunikationsmodell legt fest, wie auf den Verbindungen zwischen den Schnittstellen von Komponenten Informationen ausgetauscht werden. Prinzipiell wird die Information von einem Ausgang einer Komponente zu einem Eingang einer verbundenen Komponente in Form von Nachrichten synchron oder asynchron übertragen. Die empfangenen Nachrichten am Eingang einer Komponente lösen an der Schnittstelle Ereignisse aus, die das innere Verhalten einer Komponente beeinflussen. Ein Ereignis kann prinzipiell durch einen Operationsaufruf, eine Zustandsänderung einer Variablen, ein Ablaufen eines Zeitgebers (engl. timer) oder ein Signal ausgelöst werden. Das innere Verhalten einer Komponente erzeugt bei der Ausführung wiederum Nachrichten, die über die Ausgänge der Schnittstelle an andere Komponenten gesendet werden können.

Da speziell bei Echtzeitsystemen die Ausführung der Komponenten auf verschiedene nebenläufige Programme verteilt sein kann, muss das Kommunikationsmodell neben dem Informationsaustausch auch die Datenkonsistenz für versendete Nachrichten im Echtzeitbetrieb gewährleisten. Dies kann prinzipiell durch Warteschlangen an den Eingängen der Komponenten oder durch die Verwendung von Zustandsnachrichten mit lokalen Kopien erreicht werden.

Ausführungsmodell

Das Ausführungsmodell ist bei Echtzeitsystemen besonders wichtig, da es die Einhaltung der Echtzeitanforderungen sicherstellen muss. Im Ausführungsmodell wird spezifiziert, in welcher Reihenfolge die Komponenten ausgeführt werden. Prinzipiell kann zwischen zeitgesteuerten und ereignisgesteuerten Ausführungsmodellen unterschieden werden. Bei zeitgesteuerten Ausführungsmodellen wird die Ausführungsreihenfolge bei der Spezifikation statisch festgelegt. Ein ereignisgesteuertes Ausführungsmodell bestimmt die Ausführungsreihenfolge dynamisch zur Ausführungszeit anhand eines zuvor festgelegten Scheduling-Verfahrens. Die ereignisgesteuerte Ausführung wird in der Implementierung z.B. durch den Einsatz eines Echtzeitbetriebssystems oder eines Thread-Handlers realisiert.

3.1.3 Ansätze zur komponentenbasierten Softwareentwicklung für Echtzeitsysteme

3.1.3.1 Übersicht

In diesem Abschnitt werden die aus Sicht dieser Arbeit interessantesten komponentenbasierten Softwareentwicklungsmethoden für Echtzeitsysteme vorgestellt. Dazu werden die verfügbaren Beschreibungsmittel zur Entwurfsspezifikation, die zugehörigen Vorgehensweisen und unterstützende CASE-Werkzeuge untersucht. Tabelle 3.2 gibt eine Übersicht über die Beschreibungsmittel der drei ausgewählten komponentenbasierten Softwareentwicklungsmethoden für Echtzeitsysteme IEC 61499 Standard [IEC61499], ROOM [SGW94] und ASCET-SD [ETAS00]. Diese drei wurden ausgewählt, da ihre Beschreibungsmittel und ihre unterstützenden CASE-Werkzeuge bereits erfolgreich in Industrieprojekten eingesetzt werden. Die weitaus größere Anzahl vergleichbarer Ansätze, wie z.B. AUTOFOCUS [BGG+96], Ptolomey [LiLe00] oder Rapide [Rapi97], befinden sich hingegen noch im Forschungsstadium.

Tabelle 3.2: Beschreibungsmittel komponentenbasierter Softwareentwicklungsmethoden für Echtzeitsysteme

	IEC 61499	ROOM	ASCET-SD
Komponentenverhalten	Zustandsdiagramm	Zustandsdiagramm	Zustandsdiagramm, Blockdiagramm, ESDL, C-Code
Strukturmodell	Funktionsblockdiagramm	ROOM-Chart	Blockdiagramm
Verteilungsmodell	Systemmodell	(ROOM-Chart)	(Blockdiagramm)
Kommunikationsmodell	Getrennte Nachrichten für Ereignisse und Daten	Kombinierte Nachrichten für Ereignisse plus optional Daten	Getrennte Nachrichten für Ereignisse und Daten
Ausführungsmodell	Ressourcenmodell	Implizite Zuordnung der Ereignis-Nachrichten auf die „ROOM Virtual Machine“	Explizite Zuordnung der Ereignis-Nachrichten auf die Tasks eines OSEK/VDX Echtzeitbetriebssystems

Wie in Tabelle 3.2 zu sehen ist, erfüllen die drei ausgewählten Beschreibungsmittel im Prinzip alle im Abschnitt 3.1.2 aufgestellten Anforderungen an die komponentenbasierte Softwareentwicklung für Echtzeitsysteme. Aus diesem Grund werden nachfolgend vor allem ihre Besonderheiten herausgestellt.

3.1.3.2 IEC 61499

Der für den Anwendungsbereich der Anlagenautomatisierung entwickelte Standard IEC 61499 [IEC61499] bietet als Weiterentwicklung des SPS-Standards IEC 61131-3 [IEC61131-3] Beschreibungsmittel für die komponentenbasierte Entwurfsspezifikation verteilter Automatisierungssoftware. Die Komponenten werden als Funktionsbausteine (engl. function blocks) bezeichnet und in Funktionsblockdiagrammen grafisch zu komponentenbasierter Software verbunden. Zur Spezifikation des inneren Verhaltens von Komponenten stehen Zustandsdiagramme zur Verfügung. Die Ein- und Ausgänge der Komponenten-Schnittstelle sind in die Bereiche Ereignisfluss und Datenfluss aufgeteilt. Die Verteilung der Komponenten auf mehrere Automatisierungsrechner wird mit einem Systemmodell spezifiziert. Die Ausführung der Komponenten wird gemäß dem Standard über Ereignis-Nachrichten angestoßen. Auf jedem Automatisierungsrechner wird dies über eine zentrale Ausführungssteuerung, z.B. einem Echtzeitbetriebssystem, realisiert. Die Zuordnung auf die zentrale Ausführungssteuerung erfolgt im Ressourcenmodell, für das im Standard keine formale Syntax festgelegt ist.

Werkzeug ASPECT

Die Entwicklung von komponentenbasierter Software mit IEC 61499 Komponenten wird beispielsweise von dem CASE-Werkzeug ASPECT [ISG01] der ISG GmbH unterstützt. Die für ASPECT vorgeschlagene Vorgehensweise [SLSW99] beginnt mit der Erstellung eines objektorientierten Analysemodells mit Hilfe von Klassen- und Objektdiagrammen. Beim anschließenden komponentenbasierten Entwurf der Steuerungssoftware werden die Objekte des Analysemodells durch Instanzen von Funktionsblöcken (IEC 61499-Komponenten) ersetzt. Die Komponenten werden von ASPECT in der Komponenten-Bibliothek des Werkzeugs zur Mehrfachverwendung bereitgestellt. Das Verhalten neu entwickelter Komponenten kann mit Hilfe von Zustandsdiagrammen spezifiziert werden. ASPECT stellt für alle Diagrammartentypen grafische Editoren zur Verfügung. Aus dem komponentenbasierten Entwurfsmodell kann wahlweise C-Code oder Anweisungsliste (AWL) nach IEC 61131-3 generiert werden. Der generierte Code kann dann manuell in die Ausführungssteuerung eines Automatisierungsrechners, z.B. einer SPS oder eines Industrie-PC, eingebunden werden.

Zum Testen der generierten komponentenbasierten Software steht in ASPECT eine interaktive Testumgebung zur Verfügung. Der interne Zustand einzelner Komponenten wird dabei durch die Animation des hinterlegten Zustandsautomaten visualisiert und der Zustand mehrerer Komponenten kann tabellarisch dargestellt werden.

3.1.3.3 ROOM

Die „Real-Time Object-Oriented Modeling“ (ROOM) Methode [SGW94] hat ihren Ursprung in der Erstellung von Echtzeit-Software für Telekommunikationssysteme. Die Komponenten werden in sogenannten ROOM-Charts grafisch spezifiziert und als „Actors“ bezeichnet. Zur Spezifikation des inneren Verhaltens der Komponenten stehen hierarchische Zustandsautomaten nach [Hare87] zur Verfügung. Über die verbundenen Ein- und Ausgänge der Komponenten-Schnittstellen werden asynchron Nachrichten ausgetauscht. Jede Komponente besitzt für ihre Eingänge FIFO-Warteschlangen zur Pufferung der empfangenen Nachrichten. Eine Nachricht ist immer ein Ereignis und kann optional auch Daten enthalten. Alle Nachrichten werden mit einer Priorität versehen. Als zentrale Ausführungssteuerung gibt es die „ROOM Virtual Machine“, welche für einen Automatisierungsrechner die FIFO-Warteschlangen aller Komponenten zentral verwaltet und aus den Prioritäten der Nachrichten die Ausführungsreihenfolge der Komponenten zur Ausführungszeit bestimmt.

Die Vorgehensweise der ROOM-Methode basiert auf der schrittweisen Top-Down-Verfeinerung der Komponenten in der Analyse- und Entwurfsphase. Das bedeutet, dass eine Komponente in mehrere andere Komponenten verfeinert wird. Das Verhalten der Komponenten auf den untersten Hierarchieebenen wird immer durch einen Zustandsautomaten spezifiziert. Auch Komponenten, die sich aus mehreren Komponenten zusammensetzen, können ebenfalls noch zusätzlich einen Zustandsautomaten enthalten.

Werkzeug ObjecTime Developer / Rational Rose RealTime

Zur ROOM-Methode wurde das CASE-Werkzeug ObjecTime Developer [Obj97] von ObjecTime Limited entwickelt. Seit kurzem wird das CASE-Werkzeug auch unter der neuen Bezeichnung Rational Rose RealTime [Rati01] von Rational vertrieben. Das Werkzeug unterstützt die komponentenbasierte Softwareentwicklung nach der ROOM-Methode. Es stellt Editoren für die Spezifikation des Strukturmodells mit ROOM-Charts und für die Verhaltensspezifikation mit Zustandsdiagrammen bereit. Aus einer ROOM Entwurfsspezifikation kann mit ObjecTime Developer C- oder C++-Code für eine vorgefertigte „ROOM Virtual Machine“ generiert werden. Die „ROOM Virtual Machine“ realisiert die zentrale Nachrichten-Verwaltung und arbeitet die in den Warteschlangen gepufferten Eingangs-Nachrichten der Komponenten prioritätsgesteuert ab. Bei Nachrichten auf derselben Prioritätsebene wird nach dem FIFO-Prinzip verfahren.

Die generierte Software kann zusammen mit der „ROOM Virtual Machine“ direkt auf einem Automatisierungsrechner oder innerhalb einer Task eines Echtzeitbetriebssystems ausgeführt werden. Das Zusatz-Werkzeug TestScope ermöglicht für das Testen von einem Entwicklungsrechner aus die Beobachtung von versendeten Nachrichten zwischen Komponenten bei der Ausführung im Automatisierungsrechner. Die beobachteten Nachrichten-Sequenzen können als „Message Sequence Charts“ (MSC) [ITU120] visualisiert werden.

ObjecTime Developer bietet außerdem die Simulation der Entwurfsspezifikation auf dem Entwicklungsrechner an. Dabei existieren dieselben Beobachtungsmöglichkeiten für Nachrichten, wie beim Einsatz von TestScope. Die verwendete Simulationszeit ist jedoch nur die reale Uhrzeit des verwendeten Simulationsrechners, so dass keine realitätsnahen Auswertungen bezüglich des Echtzeitverhaltens möglich sind.

3.1.3.4 ASCET-SD Komponentenmodell

Das zum CASE-Werkzeug ASCET-SD [ETAS00] gehörende Komponentenmodell wurde speziell zur Entwicklung komponentenbasierter Software für Steuergeräte im Kfz-Bereich entwickelt. Die Komponenten werden grafisch spezifiziert und als „Module“ bezeichnet. Die Spezifikation des inneren Verhaltens der Komponenten kann mit regelungstechnischen Blockdiagrammen [Föll78], hierarchischen Zustandsautomaten [Hare87], in der Programmiersprache C oder in der Werkzeug-spezifischen Spezifikationssprache „Embedded Software Description Language“ (ESDL) erfolgen. Das Strukturmodell wird als Blockdiagramm spezifiziert. Im Strukturmodell können Komponenten mit unterschiedlicher innerer Verhaltensbeschreibung über ihre Schnittstellen miteinander verbunden werden. Die Ein- und Ausgänge von Komponenten-Schnittstellen werden nach Ereignisfluss und Datenfluss unterschieden. Als Ausführungsmodell ist ein OSEK/VDX-basiertes Echtzeitbetriebssystem [OSEK00] vorgegeben. Die Ereignis-Nachrichten werden beim Entwurf der Ausführungssteuerung den Tasks des Echtzeitbetriebssystems zugeordnet. Alle Ereignis-Nachrichten zwischen den Komponenten eines Steuergeräts werden zentral durch das Echtzeitbetriebssystem verwaltet. Das OSEK/VDX-basierte Echtzeitbetriebssystem bestimmt die Ausführungsreihenfolge der Komponenten zur Ausführungszeit nach prioritätsgesteuertem präemptivem Scheduling.

Werkzeug ASCET-SD

Das CASE-Werkzeug ASCET-SD unterstützt die Entwurfsspezifikation komponentenbasierter Software für Steuergeräte. Es besitzt eine Komponenten-Bibliotheksverwaltung und liefert bereits vorgefertigte Komponenten der standardisierten MSR-MEGMA-Komponenten-Bibliothek [MSR99] mit. Die Editoren unterstützen den grafischen Entwurf von komponentenbasierter Software, die Konfigurierung der Echtzeitbetriebssystem-Komponente sowie die Spezifikation neuer Komponenten. Der Codegenerator erzeugt Laufzeit- und Speicherplatz-optimierten C-Code, welcher automatisch mit dem konfigurierten OSEK/VDX-basierten Echtzeitbetriebssystem ERCOS^{EK} [ETAS01] verbunden wird. Die generierte Software kann auf allen Mikrocontroller-basierten Steuergeräten betrieben werden, für die ERCOS^{EK} verfügbar ist.

ASCET-SD besitzt eine Testumgebung, mit der von einem Entwicklungsrechner aus die Nachrichten-Kommunikation zwischen den Komponenten bei der Ausführung im Steuergerät beobachtbar ist.

Für den Entwicklungsrechner oder für ein spezielles VME-Bus-basiertes Echtzeitsimulationssystem kann ein Simulationsmodell generiert werden. Die Simulationsumgebung besitzt


echtzeitfähige Beobachtungs- und Aufzeichnungsmöglichkeiten für die kommunizierten Nachrichten zwischen den Komponenten, die auch grafisch visualisiert werden können. Außerdem kann der interne Zustand einzelner Komponenten, deren Verhalten mit Zustandsautomaten spezifiziert wurde, bei der Simulation animiert dargestellt werden. Bei Simulation auf dem Echtzeitsimulationssystem wird das Echtzeitverhalten des Echtzeitbetriebssystems realitätsnah nachgebildet. Dadurch kann bei der Simulation auch das Echtzeitverhalten geprüft werden.

3.1.3.5 Bewertung der unterschiedlichen Werkzeugunterstützungen

Die Unterstützung für Entwurf und Prüfung von komponentenbasierter Software durch die verfügbaren Werkzeuge der ausgewählten komponentenbasierten Softwareentwicklungsmethoden ist in Tabelle 3.3 zusammenfassend bewertet.

Tabelle 3.3: Bewertung der Werkzeugunterstützungen

		ASPECT (IEC 61499)	ObjecTime Developer (ROOM)	ASCET-SD
Entwurf	Komponenten-Bibliotheksverwaltung	●	◐	●
	Spezifikation neuer Komponenten	●	●	●
	Entwurf komponentenbasierter Anwendungssoftware	●	●	●
	Entwurf verteilter komponentenbasierter Anwendungssoftware	●	◐	◐
	Konfigurierung der Ausführungssteuerung	○	◐	●
	Automatische Codegenerierung für Automatisierungsrechner	●♣	●♦	●♦
Prüfung	Simulationsumgebung	○	●	●
	Simulation von Echtzeitverhalten	○	○	●
	Testumgebung	◐	◐	◐



* C-Code oder IEC 61131-3 Anweisungsliste (AWL)

♦ C-Code

Die Bewertung der Werkzeugunterstützungen in Tabelle 3.3 zeigt, dass der Entwurf komponentenbasierter Software von allen drei Werkzeugen ausreichend gut unterstützt wird. ObjecTime Developer und ASCET-SD bieten zusätzlich noch Simulationsumgebungen zur Prüfung der Software an. ASCET-SD bietet als einziges Werkzeug eine realitätsnahe Echtzeitsimulation, das jedoch die Verwendung eines speziellen VME-Bus-basierten Echtzeitsimulationssystems voraussetzt.

3.2 Stand der Technik bei der Validierung von Software für Echtzeitsysteme

3.2.1 Anforderungen an Prüfverfahren zur Validierung von Software für Echtzeitsysteme

Wie bereits in den Grundlagen in Abschnitt 2.2.6 festgestellt, können zur Validierung von Software prinzipiell alle Prüfverfahren verwendet werden, die geeignet sind die Prüfziele zu erreichen. Die Prüfziele bei der Validierung sind das Auffinden von Fehlern in der komponentenbasierten Software und der Nachweis, dass die Anforderungen aus Anwendersicht erfüllt sind. Die Prüffallentwicklung bei einem Prüfverfahren zur Validierung sollte daher eine systematische Herleitung von Prüffällen aus diesen Anforderungen unterstützen. Um die Anforderungen als Referenz bzw. Referenz-Verhalten für eine Prüfung verwenden zu können, sollten sie mit prüfbar beschreibenden Mitteln spezifiziert sein. Da die funktionale Gültigkeit von Software für Echtzeitsysteme nicht nur vom Resultat der Berechnung bzw. der Ausgabe von Steuerungsinformationen abhängt, sondern auch davon, ob das Resultat bzw. die Ausgabe innerhalb vorgegebener und vorhersagbarer Zeitschranken produziert und ausgegeben wird, müssen bei der Prüfung auch Echtzeitanforderungen explizit spezifiziert und überprüft werden können.

3.2.2 Ansätze zur Prüfung von Software für Echtzeitsysteme

3.2.2.1 Übersicht

In der Literatur finden sich viele Ansätze zur Prüfung von Software für Echtzeitsysteme, die prinzipiell auf den bekannten Prüfverfahren [Bind99, FLS95, Ligg00, Myer91] basieren. Stellvertretend für die vielen Ansätze werden nachfolgend einige charakteristische Ansätze vorgestellt und bewertet, welche eine Prüfung von Echtzeitverhalten unterstützen und als Prüfverfahren Testen bzw. Simulation verwenden. Tabelle 3.4 gibt eine Übersicht über die untersuchten Ansätze. Als Unterscheidungskriterien für die Ansätze sind darin das verwendete Prüfverfahren, der Ausgangspunkt für die Herleitung von Prüffällen, die Beschreibungsmittel für das Referenz-Verhalten zu den Prüffällen, die Berücksichtigung von Echtzeitverhalten bei der Prüfung und die Entwicklungsphase, in der die Prüfung stattfindet, aufgeführt.

Tabelle 3.4: Ansätze zur Prüfung von Software für Echtzeitsysteme

	Bergmann	Chung et al.	Strassacker	Belschner
Prüfverfahren	Testen	Testen	Testen + Simulation	Simulation
Ausgangspunkt für die Herleitung von Prüffällen	Entwurfsspezifikation + Progr.struktur	MSC Entwurfsspezifikation	Keine	Keine
Beschreibungsmittel für das Referenz-Verhalten zu den Prüffällen	Eigene Sprache	MSC	Keine	Eigene Sprache (TC-SL)
Prüfung von Echtzeitverhalten	Ja	Bedingt	Bedingt	Ja
Entwicklungsphase in der die Prüfung stattfindet	Implementierung + Integration	Implementierung + Integration	Integration + Abnahme	Entwurf

3.2.2.2 Bergmann

Jochen Bergmann [Berg98, BeBe99] entwickelte ein Testverfahren zur Funktions- und Echtzeitprüfung von Automatisierungssoftware für eingebettete Systeme. Zur Spezifikation des Referenz-Verhaltens konzipierte er eine eigene textuelle Verhaltensbeschreibungssprache, welche Sprachelemente für verschiedene funktionale Verhaltensaspekte bietet. Die Verhaltensbeschreibungssprache besitzt Sprachelemente zur Beschreibung von Datentransformationen, zustandsabhängigem Verhalten und dem Ablaufverhalten von Algorithmen.

Die Prüffälle für die in der Verhaltensbeschreibung spezifizierten Datentransformationen werden nach dem von Myers [Myer91] beschriebenen Verfahren der Äquivalenzklassenbildung für die Eingabedaten hergeleitet. Für die spezifizierten zustandsabhängigen Bestandteile der Verhaltensbeschreibung werden die Prüffälle nach der von Binder [Bind99] sowie Liggesmeyer und Rüpl [LiRü96] empfohlenen Strategie entwickelt. Bei dieser Strategie werden Prüffälle systematisch aus einer Zustandsautomaten-Spezifikation hergeleitet. Die Prüffälle müssen dabei die Ausführung aller Zustände und Zustandsübergänge im Programm abdecken. Für die als Ablaufpfade in der Verhaltensbeschreibung spezifizierten Bestandteile werden Prüffälle entwickelt, welche die Ausführung der geforderten Ablaufpfade im Programm sicherstellen.

Auf Basis des kommerziellen Testwerkzeugs CANTATA [IPL01] von IPL Corp. entwickelte Bergmann eine unterstützende Echtzeitprüfumgebung, welche die Prüfinstrumentierung der

implementierten Automatisierungssoftware und die Prüfablaufsteuerung beim Testen auf einem Steuergerät automatisiert. Die Bewertung des beobachteten Verhaltens wird ebenfalls automatisch durchgeführt.

Bewertung

Die Prüffallentwicklung erfolgt bei diesem Testverfahren sehr systematisch. Die proprietäre textuelle Verhaltensbeschreibungssprache enthält umfangreiche Sprachmittel zur Spezifikation des Referenz-Verhaltens. Mit den Sprachmitteln können prinzipiell alle Arten von Anforderungen – speziell auch Echtzeitanforderungen – für die Prüfung formuliert werden. Da die Syntax der Verhaltensbeschreibungssprache aber in etwa den Umfang einer Programmiersprache aufweist, ist der Einsatz in der Praxis sicherlich mit großem Lernaufwand und Akzeptanzproblemen verbunden. Die Verhaltensbeschreibung setzt bei vielen Bestandteilen bereits eine exakte Kenntnis der Programmstruktur voraus, so dass das Testen nur mit White-Box-Wissen sinnvoll durchgeführt werden kann. Aus diesem Grund kann das Testverfahren erst am Ende der Implementierungs- bzw. Integrationsphase zur Validierung von Anforderungen eingesetzt werden.

3.2.2.3 Chung et. al.

Chung, Kim, Bae, Kwon und Lee [CKB+99] entwickelten ein Testverfahren für das spezifikationsbasierte Testen von Modulen nebenläufiger CHILL-Programme für Telekommunikations-Vermittlungsknoten (engl. switching system). Zur Spezifikation des Referenz-Verhaltens werden die in der Telekommunikation stark verbreiteten Sequenzdiagramme „Message Sequence Charts“ (MSC) [ITU120] verwendet. Die MSCs sind Bestandteil der Spezifikationsprache „Specification and Description Language“ (SDL) [ITU100]. Mit den MSCs werden sequenzielle Abläufe von Nachrichten zwischen SDL-Modulen grafisch spezifiziert.

Bei der Prüffallentwicklung werden die einzelnen Programm-Module als Black-Box-Komponenten betrachtet. Aus diesem Grund wird für die Prüfung einzelner Programm-Module aus jedem MSC ein reduziertes MSC erstellt, indem die spezifizierten Nachrichten, die innerhalb des Programm-Moduls ablaufen sollen, eliminiert werden. Das reduzierte MSC beschreibt nur noch die Interaktion des Moduls mit seiner Umgebung. Bei der Testausführung auf dem Telekommunikations-Vermittlungsknoten wird das Programm-Modul dann explizit für die Ausführung des reduzierten MSC stimuliert, und die Antwort-Nachrichten beobachtet und mit der spezifizierten Referenz im reduzierten MSC verglichen. Zur Beobachtung der stimulierenden Nachrichten und der Antwort-Nachrichten wird das CHILL-Programm-Modul manuell instrumentiert.

Bewertung

Das Testverfahren ermöglicht eine systematische Herleitung von Prüffällen aus einer grafischen MSC-Spezifikation. Mit MSCs können Anforderungen als geforderte Reihenfolge von Nachrichten zwischen Programm-Modulen – Software-Komponenten auf Implementierungsebene –

spezifiziert werden. Die Spezifikation von Echtzeitanforderungen, wie z.B. Zeitbedingungen zwischen Nachrichten, ist in dem vorgestellten Ansatz jedoch nicht vorgesehen. Bei der Testausführung auf dem Telekommunikations-Vermittlungsknoten in Echtzeit wird nur die Einhaltung der kausalen Reihenfolge der Nachrichten verglichen, wodurch nur bedingt eine Bewertung des Echtzeitverhaltens erfolgt. Das Testverfahren kann während und nach der Implementierungsphase zur Validierung von Anforderungen eingesetzt werden.

3.2.2.4 Strassacker

Dirk Strassacker [Stra97] entwickelte eine Testumgebung für die Implementierung und Inbetriebnahme von adaptierbaren Prozessleitsystemen. Die Testumgebung unterstützt sowohl den Modultest als auch den Systemtest der Software eines Prozessleitsystems. Für das Testen wird mit Hilfe eines Simulationsmodells das Antwortverhalten eines flexiblen Produktionssystems auf Auftrags-Nachrichten des Prozessleitsystems nachbildet. Die Testumgebung integriert ein Simulationswerkzeug zur Erstellung von Simulationsmodellen für flexible Produktionssysteme. Das Simulationswerkzeug enthält eine Bibliothek mit Standard-Modellkomponenten, aus denen die Anlagenkomponenten des technischen Prozesses und zugehörige Steuerung modelliert werden können. Außerdem können auf Basis von Zustandsautomaten oder durch Programmierung in der Programmiersprache SMALLTALK eigene Simulationsmodell-Komponenten entwickelt werden.

Die Simulation erfolgt im Zeitrafferbetrieb. Die Testumgebung ermöglicht dabei die Beobachtung der in der Produktionsanlage ablaufenden Vorgänge und der Nachrichten-Kommunikation zwischen dem Prozessleitsystem und dem flexiblen Produktionssystem.

Bewertung

Die Testumgebung unterstützt die Nachbildung eines flexiblen Produktionssystems mit Hilfe eines Simulationsmodells. Dadurch kann die Software eines Prozessleitsystems bereits ohne verfügbare Produktionsanlagen nach der Implementierung frühzeitig getestet werden. Das Testen kann mit der Testumgebung interaktiv oder anhand zuvor festgelegter Testszenarios erfolgen. Die Simulation des flexiblen Produktionssystems im Zeitrafferbetrieb ermöglicht die Prüfung der Software eines Prozessleitsystems auch über größere Zeitspannen hinweg. Durch den zusätzlichen Rechenaufwand für die Simulation und die Nachrichten-Kommunikation zwischen dem Simulationsmodell und der Software des Prozessleitsystems ist eine Bewertung des Echtzeitverhaltens jedoch nur bedingt möglich. Eine Unterstützung zur systematischen Prüffallentwicklung wird von der Testumgebung nicht angeboten.

3.2.2.5 Belschner

Ralf Belschner [Bels97] entwickelte ein Verfahren zur simulationsbasierten Analyse von verteilten Prozessautomatisierungssystemen. Zur Spezifikation von Referenz-Anforderungen entwickelte er die textuelle Spezifikationsprache „Timing Constraint – Specification

Language“ (TC-SL). TC-SL unterstützt nur die Spezifikation von Echtzeitanforderungen zwischen Ereignis-Zeitpunkten. Mit TC-SL können Anforderungen an die kausale Reihenfolge von mehreren Ereignissen und Zeitbedingungen für einmalig oder periodisch auftretende Ereignisse spezifiziert werden. Ein Ereignis-Zeitpunkt kann beispielsweise als Versenden einer Nachricht zwischen Software-Komponenten definiert sein.

Geprüft wird damit das simulierte Verhalten von Entwurfsspezifikationen von Software in einem verteilten Prozessautomatisierungssystem. Zur Entwurfsspezifikation der Software verwendet er die ebenfalls selbst entwickelte Spezifikationsprache „Extended Specification Language for Portable Operating System Interface“ (EPOSIX), die neben der Spezifikation von Kontroll- und Datenstrukturen auch die Verwendung von Echtzeitsprachmitteln von Echtzeit-Unix-Betriebssystemen, basierend auf dem POSIX 1003.4 Standard [FGG+91], unterstützt. Zur Evaluierung des Prüfverfahrens entwickelte er ein Simulationsmodell, welches die verschiedenen Bestandteile eines verteilten Prozessautomatisierungssystems nachbildet. Das Simulationsmodell enthält konfigurierbare Simulationsmodell-Komponenten zur Nachbildung des Verhaltens des Bussystems und der verteilten Automatisierungsrechner, einschließlich eines darauf ablaufenden POSIX 1003.4 konformen Echtzeit-UNIX-Betriebssystems. Ebenso wurden Simulationsmodell-Komponenten zur Nachbildung des Verhaltens des technischen Prozesses integriert. Die EPOSIX Software-Entwurfsspezifikationen können in der Simulation auf den Instanzen der Echtzeit-UNIX-Betriebssysteme ausgeführt werden. Zur Beobachtung des simulierten Verhaltens wird das Simulationsmodell manuell instrumentiert. Die Simulation ausführung erfolgt im Zeitrafferbetrieb.

Mit den von ihm entwickelten Analyse- und Parameteroptimierungs-Werkzeugen wird das beobachtete Verhalten des Simulationsmodells gegenüber den in TC-SL spezifizierten Echtzeitanforderungen mit Hilfe von Gütekriterien automatisch bewertet und Systemparameter mit Hilfe genetischer Algorithmen automatisch optimiert. Außerdem steht eine grafische Visualisierung der Simulationsaufzeichnungen in Form von Taskablaufdiagrammen zur Verfügung.

Bewertung

Das von Belschner entwickelte Analyseverfahren demonstriert, wie mit Hilfe von Simulation die frühzeitige Prüfung von Entwurfsspezifikationen von Automatisierungssoftware möglich ist. Die selbst entwickelte proprietäre Spezifikationsprache EPOSIX bietet Sprachmittel zur Spezifikation von Task-internem Verhalten und der Kommunikation zwischen Tasks. Damit ist prinzipiell auch eine Entwurfsspezifikation von Software-Komponenten möglich, sofern pro Software-Komponente eine Task spezifiziert wird. Dies deckt sich jedoch nicht mit der ursprünglichen Intention von EPOSIX.

Das von ihm entwickelte Analyseverfahren legt den Schwerpunkt auf die Optimierung von Systemparametern. Eine Systematik zur Entwicklung von Prüffällen, welche auf die Aufdeckung von Entwurfsfehlern abzielt, wurde nicht entwickelt. Die sehr einfache Spezifikations-

sprache TC-SL unterstützt sehr gut die Spezifikation von Echtzeitanforderungen. Die Sprachmittel reichen jedoch nicht aus, um Anforderungen an die Funktionalität aus Anwendersicht verständlich spezifizieren zu können.

3.3 Zusammenfassende Bewertung und Festlegung der Anforderungen

3.3.1 Bewertung der Ansätze zur komponentenbasierten Softwareentwicklung für Echtzeitsysteme

Bei der Untersuchung der komponentenbasierten Softwareentwicklung für Echtzeitsysteme wurde festgestellt, dass Komponenten bei Echtzeitsystemen aus Effizienzgründen bereits auf der Entwurfsebene eingesetzt werden müssen. Die untersuchten komponentenbasierten Softwareentwicklungsmethoden bieten ausgereifte Beschreibungsmittel zur Entwurfsspezifikation komponentenbasierter Software für Echtzeitsysteme. Die zugehörigen Werkzeuge unterstützen relativ gut die Konstruktion von komponentenbasierter Software und bieten außerdem Simulations- und Testumgebungen zur Prüfung an.

Defizite bestehen allerdings bei den propagierten Vorgehensweisen zur komponentenbasierten Softwareentwicklung. Diese sind entweder nicht vorhanden oder zielen, wie bei der ROOM-Methode, auf eine schrittweise Top-Down-Verfeinerung beim Entwurf ab. Lediglich zum Werkzeug ASPECT existiert eine ausgereifte Vorgehensweise, die eine gezielte Mehrfachverwendung von Komponenten unterstützt.

3.3.2 Bewertung der Ansätze zur Prüfung von Software für Echtzeitsysteme

Die meisten Ansätze zur Prüfung von Software für Echtzeitsysteme basieren auf dem Testen der Software. Da für das Testen die Implementierung der Software und der Automatisierungsrechner erforderlich ist, kann es erst in den späten Integrations- und Abnahmephasen durchgeführt werden. Damit ist die Validierung der Software zu einem relativ späten Zeitpunkt durchführbar. Der Ansatz von Belschner zeigt hingegen, dass mit Hilfe von Simulation bereits eine frühzeitige Prüfung von Software-Entwurfsspezifikationen machbar ist. Bei der Simulation muss jedoch das Echtzeitverhalten des simulierten Gesamtsystems möglichst realitätsnah nachgebildet werden, da sonst keine sinnvolle Bewertung des zeitlichen Verhaltens der entworfenen Software möglich ist.

Defizite finden sich insbesondere bei der Systematik zur Prüffallentwicklung. Nur ein Teil der Ansätze unterstützt eine systematische Prüffallentwicklung, was dann jedoch mit großem

zusätzlichen Aufwand verbunden ist. Eine systematische Prüffallentwicklung ist jedoch ganz entscheidend für die erfolgreiche Aufdeckung von Fehlern und dem Nachweis von Anforderungen bei der Validierung. Die Herleitung des Referenz-Verhaltens und der Stimulations-Daten für die Prüffälle wird beispielsweise in den Ansätzen von Strassacker und Belschner überhaupt nicht betrachtet. Im Ansatz von Bergmann werden dazu Kenntnisse über die Implementierung der Programmstruktur vorausgesetzt. Zusätzlich kommen in diesem Ansatz noch sehr komplexe, nicht standardisierte Beschreibungsmittel zum Einsatz, was die Akzeptanz für einen praktischen Einsatz stark beeinträchtigt. Lediglich im Ansatz von Chung et. al. erfolgt die Spezifikation des Referenz-Verhaltens in der standardisierten MSC Notation. Der Ansatz sieht jedoch keine speziellen Möglichkeiten zur Formulierung von Echtzeitanforderungen vor. Bei der Validierung komponentenbasierter Software für Echtzeitsysteme muss aber auch das zeitliche Verhalten der Software explizit überprüft werden. Das Referenz-Verhalten für die Prüffälle wird bei Bergmann aus einer zustandsautomaten-basierten Entwurfsspezifikation sowie der Programmstruktur und bei Chung et. al. aus einer MSC Entwurfsspezifikation hergeleitet. Eine direkte Herleitung aus den Anforderungen aus Anwendersicht wird jedoch von keinem der untersuchten Ansätze unterstützt.

Vergleichbare Defizite finden sich auch in der industriellen Praxis wieder [Ligg93, StWi00]. Dort wird bis heute wegen fehlender Kenntnisse und hohem Zeitdruck oftmals das intuitive, nicht reproduzierbare Entwickeln von Prüffällen und deren Durchführung ohne nachvollziehbare Prüf-Dokumentation in den Integrations- und Abnahmephasen praktiziert.

Bei der Werkzeugunterstützung wird dies noch deutlicher. Die Werkzeuge bieten allesamt Simulations- und Testumgebungen mit guter Unterstützung zur Prüfinstrumentierung, Prüf-ablaufsteuerung und zur Aufzeichnung des beobachteten Verhaltens. Ebenso werden teilweise komfortable Visualisierungen und Animationen des simulierten oder ausgeführten Verhaltens der Software angeboten. Die Werkzeuge bieten jedoch keine Unterstützung zur systematischen Entwicklung von Prüffällen.

3.3.3 Anforderungen an ein Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme

Aus den erkannten Defiziten bei der komponentenbasierten Softwareentwicklung für Echtzeitsysteme ergibt sich für die vorliegende Arbeit folgende Anforderung:

- Da die vorgestellten komponentenbasierten Softwareentwicklungsmethoden für Echtzeitsysteme keine einheitliche Vorgehensweise propagieren, soll für diese Arbeit ein verbindlicher Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme festgelegt werden.

Aus der Zielsetzung der Arbeit und den erkannten Defiziten bei den betrachteten Ansätzen zur Validierung von Software für Echtzeitsysteme ergeben sich folgende Anforderungen:

- Es soll ein Prüfverfahren zur Validierung der Funktionalität komponentenbasierter Software entwickelt werden. Dabei soll speziell das dynamische Verhalten eines Gesamtsystems beim Zusammenspiel der Komponenten geprüft werden.
- Bei der Prüfung des dynamischen Verhaltens müssen die besonderen Eigenschaften komponentenbasierter Software berücksichtigt werden. Durch die Verwendung von Black-Box-Komponenten reduzieren sich beispielsweise die möglichen Beobachtungspunkte für die Prüfinstrumentierung auf die Schnittstellen der Komponenten und deren Verbindungen.
- Die Validierung der Software soll bereits frühzeitig, auch ohne real vorhandene Bestandteile des technischen Prozesses und des Automatisierungssystems, möglich sein.
- Bei der Validierung sollen die Ergebnisse der Entwurfsphase gegenüber den Anforderungen aus Sicht des Auftraggebers bzw. der zukünftigen Anwender geprüft werden.
- Das Prüfverfahren soll eine systematische Entwicklung von Prüffällen unterstützen. Die in einer Anforderungsspezifikation enthaltenen Anforderungen an das dynamische Verhalten aus Anwendersicht sollen dabei vollständig berücksichtigt werden können. Außerdem sollen im Referenz-Verhalten neben funktionalen Abläufen auch Echtzeitanforderungen spezifiziert werden können.
- Die systematische Entwicklung von Prüffällen soll durch Werkzeuge unterstützt werden.
- Bei Echtzeitsystemen ist die Einhaltung von Echtzeitanforderungen ein wesentlicher Bestandteil der Funktionalität. Deshalb soll das Prüfverfahren auch verletzte Zeitbedingungen im dynamischen Verhalten der komponentenbasierten Software aufdecken können.
- Das Prüfverfahren soll auch die Validierung von verteilter Software bei verteilten Automatisierungssystemen unterstützen.
- Das Prüfverfahren zur Validierung soll einen möglichst hohen Automatisierungsgrad aufweisen. Automatisierbare Bestandteile sollen dabei durch Werkzeuge unterstützt werden, da nur dadurch eine große Akzeptanz für den praktischen Einsatz erreicht werden kann.

Ausgehend von den in diesem Kapitel ermittelten Anforderungen wird im nachfolgenden Kapitel eine Konzeption für ein Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme entwickelt. Da die im Stand der Technik untersuchten Ansätze zur komponentenbasierten Softwareentwicklung für Echtzeitsysteme keine einheitliche Vorgehensweise propagieren, wird zunächst ein für diese Arbeit verbindlicher Entwicklungsprozess festgelegt. Darauf aufbauend wird dann ein geeignetes Prüfverfahren zur Validierung komponentenbasierter Software für Echtzeitsysteme konzipiert und in den anschließenden Kapiteln 5 bis 7 verfeinert.

4 Konzeption eines Prüfverfahrens zur Validierung komponentenbasierter Software für Echtzeitsysteme

In diesem Kapitel wird die Konzeption eines Prüfverfahrens zur Validierung komponentenbasierter Software für Echtzeitsysteme vorgestellt. Als Ausgangsbasis dafür wird zunächst ein für diese Arbeit verbindlicher Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme definiert. Nach der Betrachtung der charakteristischen Fehlermöglichkeiten bei der komponentenbasierten Softwareentwicklung wird ein Prüfverfahren zur frühzeitigen Validierung von komponentenbasierter Software für Echtzeitsysteme konzipiert, welches insbesondere die Aufdeckung von Entwurfsfehlern unterstützt.

4.1 Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme

4.1.1 Übersicht über den Entwicklungsprozess

Bei der Bewertung der komponentenbasierten Softwareentwicklungsmethoden für Echtzeitsysteme in Abschnitt 3.3.1 wurde festgestellt, dass ausgereifte Beschreibungsmittel zur Entwurfsspezifikation und unterstützende Werkzeuge bereits im industriellen Einsatz sind. Die untersuchten Ansätze lassen jedoch keine einheitliche Vorgehensweise erkennen und die meisten Ansätze unterstützen keine gezielte Mehrfachverwendung von vorgefertigten Komponenten. Aus diesem Grund wird als Voraussetzung für die Konzeption eines Prüfverfahrens nachfolgend ein für diese Arbeit verbindlicher komponentenbasierter Softwareentwicklungsprozess für Echtzeitsysteme definiert, welcher speziell die Mehrfachverwendung von vorgefertigten Komponenten beim Entwurf berücksichtigt. Der Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme wird auf Basis des V-Modells definiert, da das V-Modell (siehe auch Abschnitt 2.2.6) in der industriellen Praxis eine weite Verbreitung und große Akzeptanz besitzt. Abbildung 4.1 zeigt die Entwicklungsphasen und Teilprodukte des Entwicklungsprozesses.

Die wesentlichen Neuerungen im Entwicklungsprozess finden sich vor allem beim komponentenbasierten Entwurf, der im Gegensatz zu bisherigen Entwurfsmethoden auf die Mehrfachverwendung von Komponenten abzielt. Anstelle der herkömmlich praktizierten schrittweisen Verfeinerung des Entwurfmodells wird das komponentenbasierte Entwurfsmodell unter Verwendung vorgefertigter Komponenten aus einer Komponenten-Bibliothek zusammengesetzt.

Aus diesem Grund wurden die beiden Entwicklungsphasen Grobentwurf und Feinentwurf des ursprünglichen V-Modells zu einer Entwicklungsphase „komponentenbasierter Entwurf“ zusammengefasst.

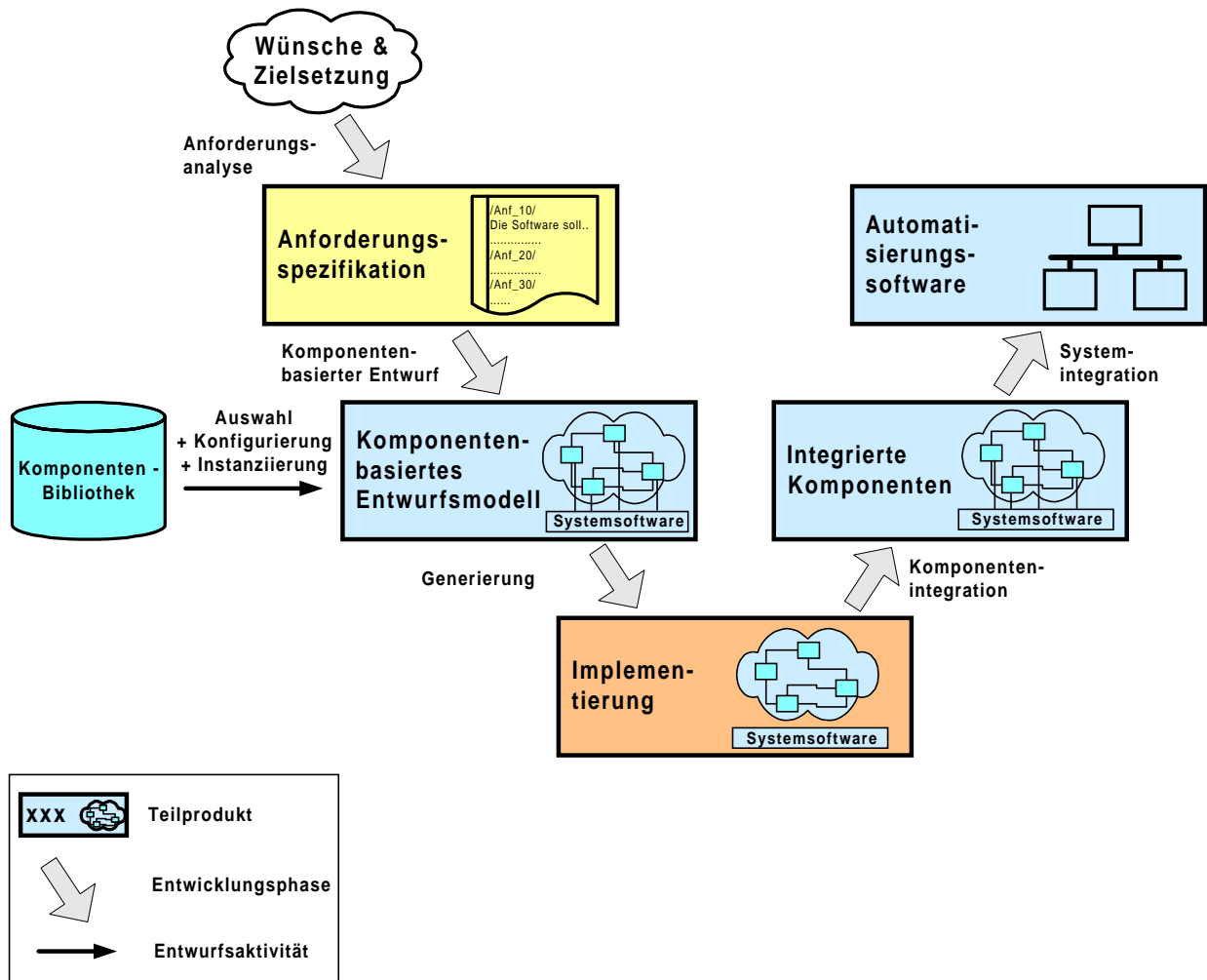


Abbildung 4.1: Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme

Zwischen den Entwicklungsphasen sind gemäß dem V-Modell auch Rückschritte möglich, so dass der Entwicklungsprozess insgesamt iterativ durchlaufen wird. Die Darstellung der Iterationen und der validierenden und verifizierenden Prüfmaßnahmen sind in Abbildung 4.1 aus Gründen der Übersichtlichkeit bewusst nicht abgebildet. Nachfolgend werden die einzelnen Entwicklungsphasen kurz beschrieben.

4.1.2 Anforderungsanalyse

Ausgehend von den Wünschen und Zielsetzungen des Auftraggebers werden bei der Anforderungsanalyse die Anforderungen an die zu entwickelnde komponentenbasierte Software erfasst. Die Ergebnisse der Anforderungsanalyse werden in einer Anforderungsspezifikation festgehalten, die häufig auch als Pflichtenheft bezeichnet wird. Als Analysemethode wird dazu

die bewährte objektorientierte Analyse (OOA) festgelegt. Die OOA wurde auch in dem IEC61499-basierten Ansatz von Storr et. al. [SLSW99] als geeignete Methode für die Anforderungsanalyse bei der komponentenbasierten Softwareentwicklung ausgewählt.

Echtzeitsysteme in der Automatisierungstechnik bestehen in der Regel sowohl aus Software- als auch aus Hardwareanteilen. Meistens sind im technischen Prozess auch mechanische, elektrische oder elektronische Komponenten an der Verarbeitung beteiligt. Der Mensch ist zudem nicht nur Auslöser, sondern teilweise auch an der Durchführung der Verarbeitungsprozesse aktiv beteiligt. Aus diesem Grund ist ein ganzheitlicher Ansatz für die Anforderungsanalyse des Echtzeitsystems notwendig, der sich nicht sofort auf die Einzelteile des Systems konzentriert, sondern das Echtzeitsystem von außen – aus der Nutzersicht – betrachtet. Die als Einstiegspunkt für die OOA vorgesehenen Anwendungsfälle (engl. use cases) [JBR98, JCJÖ92] sind aus diesem Grund besonders gut für die Anforderungsanalyse bei Echtzeitsystemen geeignet [HrRu01]. In Anwendungsfällen werden Funktionsanforderungen in natürlicher Sprache als charakteristische Abläufe für ein zu entwickelndes System beschrieben [JBR98, JCJÖ92]. Sie ermöglichen damit eine Spezifikation der funktionalen Anforderungen in einer Form, die auch für den Auftraggeber verständlich ist [Moor01]. Anwendungsfälle bieten dem Softwareentwickler die Möglichkeit, mit den zukünftigen Anwendern eines Systems und Fachleuten für den Anwendungsbereich zu einem gemeinsamen Verständnis zu kommen. Man verwendet Anwendungsfälle, um das gewünschte Verhalten eines zu entwickelnden Systems oder eines Teils eines Systems festzuhalten, ohne spezifizieren zu müssen, wie dieses Verhalten implementiert wird [BJR99]. Die Spezifikation von Anwendungsfällen wird deshalb als kleinster gemeinsamer Nenner für die Anforderungsanalyse festgelegt. Optional können noch weitere Beschreibungsmittel der OOA für die Erstellung der Anforderungsspezifikation eingesetzt werden.

4.1.3 Komponentenbasierter Entwurf

Beim komponentenbasierten Entwurf werden die Anforderungen aus der Anforderungsspezifikation in ein komponentenbasiertes Entwurfsmodell umgesetzt. Die Entwurfsspezifikation in Form eines Modells hat den Vorteil, dass durch die formale Syntax und Semantik eines Modells die vollständige Umsetzbarkeit bei der Implementierung bzw. Generierung in der nachfolgenden Entwicklungsphase gewährleistet wird. Das komponentenbasierte Entwurfsmodell besteht aus den Teilmodellen Strukturmodell, Verteilungsmodell, Kommunikationsmodell und Ausführungsmodell, um die im Abschnitt 3.1.2 aufgestellten Anforderungen an Beschreibungsmittel für komponentenbasierte Software für Echtzeitsysteme zu erfüllen. Die Durchführung des komponentenbasierten Entwurfs wird dabei durch eine geeignete Werkzeugunterstützung, wie bei den in Abschnitt 3.1.3 vorgestellten komponentenbasierten Softwareentwicklungsmethoden für Echtzeitsysteme, unterstützt. Die Vorgehensweise beim komponentenbasierten Entwurf unterscheidet sich stark gegenüber Vorgehensweisen bei herkömmlichen Entwurfsmethoden. Das komponentenbasierte Entwurfsmodell wird nicht durch schrittweise Verfeinerung (Top-

Down-Vorgehensweise) von Grund auf neu entwickelt, sondern aus vorgefertigten Komponenten aus einer Komponenten-Bibliothek zusammengesetzt (Bottom-Up-Vorgehensweise). Folgende wesentliche Entwurfsaktivitäten werden bei der Erstellung des komponentenbasierten Entwurfsmodells durchgeführt:

1. **Auswahl** von geeigneten Komponenten aus einer Komponenten-Bibliothek
2. **Konfigurierung** des Komponentenverhaltens der ausgewählten Komponenten über die vorgegebenen Komponenten-Parameter
3. **Instanziierung** der konfigurierten Komponenten im Strukturmodell
4. **Verbindung** der Eingänge und Ausgänge der Komponenten im Strukturmodell

Diese Vorgehensweise setzt eine Komponenten-Bibliothek mit Anwendungsbereichspezifischen Komponenten voraus, die speziell für die Mehrfachverwendung entwickelt und bereits bei der Komponentenentwicklung systematisch gegenüber ihrem spezifizierten Verhalten geprüft wurden. In der Praxis existieren jedoch nicht für alle Anforderungen passende Komponenten in der Komponenten-Bibliothek. Deshalb müssen in der Regel bei der Entwicklung neuer Anwendungen auch einige Komponenten neu entwickelt werden, welche die bereits vorhandenen Komponenten in der Komponenten-Bibliothek ergänzen. Insgesamt zeigt sich aber der Vorteil, dass durch den Einsatz vorgefertigter Komponenten neue Anwendungen wesentlich schneller und kostengünstiger entwickelt werden können.

4.1.4 Implementierung auf Basis von Generierung

Die Implementierung der Komponenten erfolgt durch Generierung aus dem komponentenbasierten Entwurfsmodell. Die Implementierung einer Komponente wird dabei entweder aus ihrer Verhaltensspezifikation auf der Entwurfsebene vollständig neu generiert, oder es ist bereits eine Implementierung hinterlegt, die über die konfigurierten Parameter bei der Generierung ausgewählt wird. Geeigneter Weise wird die Generierung mit Hilfe von Codegenerierungswerkzeugen automatisch durchgeführt. Bei der Generierung ist dabei eine Codeoptimierung bezüglich Laufzeit und Speicherplatzverbrauch möglich. Dies ist besonders bei der Entwicklung von Echtzeitsystemen für Massenprodukte, wie z.B. in der Produktautomatisierung, von großer Bedeutung. Der größte Vorteil im Vergleich zur herkömmlichen Softwareentwicklung liegt jedoch darin, dass die Implementierung der Software nicht mehr Zeile für Zeile neu programmiert werden muss und dadurch eine Vielzahl möglicher Fehler vermieden wird.

4.1.5 Komponenten-Integration und System-Integration

Bei der Komponenten-Integration werden die generierten Komponenten der Anwendungssoftware zusammen mit den Komponenten der Systemsoftware auf dem Automatisierungsrechner integriert. Dieser Entwicklungsschritt wird ebenfalls am besten mit der Unterstützung von Generierungswerkzeugen durchgeführt. Eine System-Integration der Software-Teilsysteme

ist erforderlich, wenn bei komplexeren Softwareentwicklungsprojekten mehrere Teilsysteme unabhängig voneinander entwickelt wurden.

4.2 Fehlermöglichkeiten bei der komponentenbasierten Softwareentwicklung für Echtzeitsysteme

Bei der Validierung soll insbesondere geprüft werden, ob die komponentenbasierte Software die Anforderungen aus Sicht des Auftraggebers bzw. der Anwender erfüllt.

Setzt man einen zuverlässigen Generierungsprozess bei der Implementierung und Integration der Komponenten durch eine ausgereifte Werkzeugunterstützung voraus, dann reduzieren sich die Möglichkeiten Fehler zu produzieren auf die Tätigkeiten während der Anforderungsanalyse und des komponentenbasierten Entwurfs. Die im Rahmen der Arbeit gesammelten praktischen Erfahrungen haben gezeigt, dass die Fehler hauptsächlich beim komponentenbasierten Entwurf der Software entstehen. Gleichwohl reduzieren sich durch den Einsatz von Komponenten die möglichen Fehlerursachen auf folgende Entwurfstätigkeiten:

- Auswahl von Komponenten mit ungeeignetem Verhalten
- Falsche Konfigurierung des Komponentenverhaltens über Komponenten-Parameter
- Fehlende Verbindungen im Strukturmodell
- Falsche Verbindungen im Strukturmodell

Entwurfsfehler zeigen sich durch fehlerhaftes Verhalten von Komponenten im Zusammenspiel mit anderen Komponenten. Folgende charakteristische Fehlerarten wurden dabei identifiziert:

- Versenden inhaltlich falscher Nachrichten
- Verspätetes Versenden von Nachrichten (Verletzung von Echtzeitanforderungen)
- Kein Versenden von Nachrichten (unterbrochener Kontrollfluss)
- Deadlock-Situationen zwischen zwei oder mehreren Komponenten

Die Konstruktionswerkzeuge zu den Abschnitt 3.1.3 vorgestellten komponentenbasierten Softwareentwicklungsmethoden unterstützen bereits beim Zusammensetzen der Komponenten während des komponentenbasierten Entwurfs die Prüfung statischer Eigenschaften, wie z.B. den Schnittstellen-Typ bei der Verbindung von Eingängen mit Ausgängen oder die Einhaltung von Komponenten-spezifischen Wertebereichen bei der Konfigurierung der Parameter. Die Prüfung des dynamischen Verhaltens der zusammengesetzten Komponenten kann hingegen nicht anhand statischer Einstellungen durchgeführt werden.

Aus diesem Grund ist es wichtig, speziell das dynamische Verhalten als funktionales Qualitätsmerkmal einer komponentenbasierten Software im Kontext des gesamten Echtzeitsystems mit Hilfe eines geeigneten Prüfverfahrens gegenüber den Anforderungen zu validieren.

4.3 Grundkonzept des Prüfverfahrens

Die komponentenbasierte Softwareentwicklung vermeidet durch die Verwendung bereits geprüfter Komponenten viele Implementierungsfehler. Bei der Entwicklung komponentenbasierter Software für Echtzeitsysteme entsteht daher der größte Prozentsatz der Fehler bereits während den frühen Entwicklungsphasen. Aus diesem Grund besteht großes Interesse diese Fehler möglichst frühzeitig aufzudecken und zu beseitigen, um dadurch die Fehlerbeseitigungskosten zu reduzieren. Wie im vorigen Abschnitt bereits herausgearbeitet wurde, entsteht der Großteil der Fehler, wie z.B. die Verletzung von Echtzeitanforderungen, durch nicht berücksichtigte oder verletzte Anforderungen beim komponentenbasierten Entwurf, welche erst im Zusammenspiel der Komponenten erkennbar sind. Ein Prüfverfahren muss deshalb speziell die frühzeitige Aufdeckung von Entwurfsfehlern unterstützen.

Bei der komponentenbasierten Softwareentwicklung für Echtzeitsysteme werden beim Entwurf Black-Box-Komponenten eingesetzt, die funktional und zeitlich gekapselt sind und deren Verhalten bereits vollständig spezifiziert ist. Dadurch ist auch das Verhalten eines aus Komponenten zusammengesetzten komponentenbasierten Entwurfsmodells bereits ausreichend spezifiziert, um es gegenüber den Anforderungen validieren zu können.

Die Grundidee des Prüfverfahrens zur Validierung komponentenbasierter Software besteht darin, bereits das dynamische Verhalten des komponentenbasierten Entwurfsmodells gegenüber den in der Anforderungsspezifikation definierten Anforderungen frühzeitig zu prüfen.

Da in der Entwurfsphase der komponentenbasierten Software das Vorhandensein der Automatisierungsrechner und auch der Bestandteile des technischen Prozesses nicht vorausgesetzt werden kann, soll als Hilfsmittel zur Prüfung die Simulation eingesetzt werden. Das Verhalten des komponentenbasierten Entwurfsmodells der Software soll deshalb mit Hilfe eines ausführbaren Simulationsmodells gegenüber den in der Anforderungsspezifikation enthaltenen Anforderungen geprüft werden. Abbildung 4.2 zeigt das Grundkonzept für ein Prüfverfahren zur Validierung des dynamischen Verhaltens von komponentenbasierter Software für Echtzeitsysteme.

Das Grundkonzept des Prüfverfahrens sieht vor, dass aus dem komponentenbasierten Entwurfsmodell der Software ausführbare Simulationsmodell-Komponenten generiert werden. Mit diesen Simulationsmodell-Komponenten der komponentenbasierten Software und zusätzlichen Simulationsmodell-Komponenten zur Nachbildung des Verhaltens der Umgebung (z.B. Benutzer, Automatisierungsrechner und technischer Prozess) wird ein Simulationsmodell des gesamten Echtzeitsystems erstellt, welches eine realitätsnahe Verhaltenssimulation ermöglicht. Realitätsnah bedeutet, dass das Simulationsmodell auch das Echtzeitverhalten der komponentenbasierten Software möglichst gut nachbildet, damit die Einhaltung von Echtzeitanforderungen bei der Prüfung ebenfalls validiert werden kann.

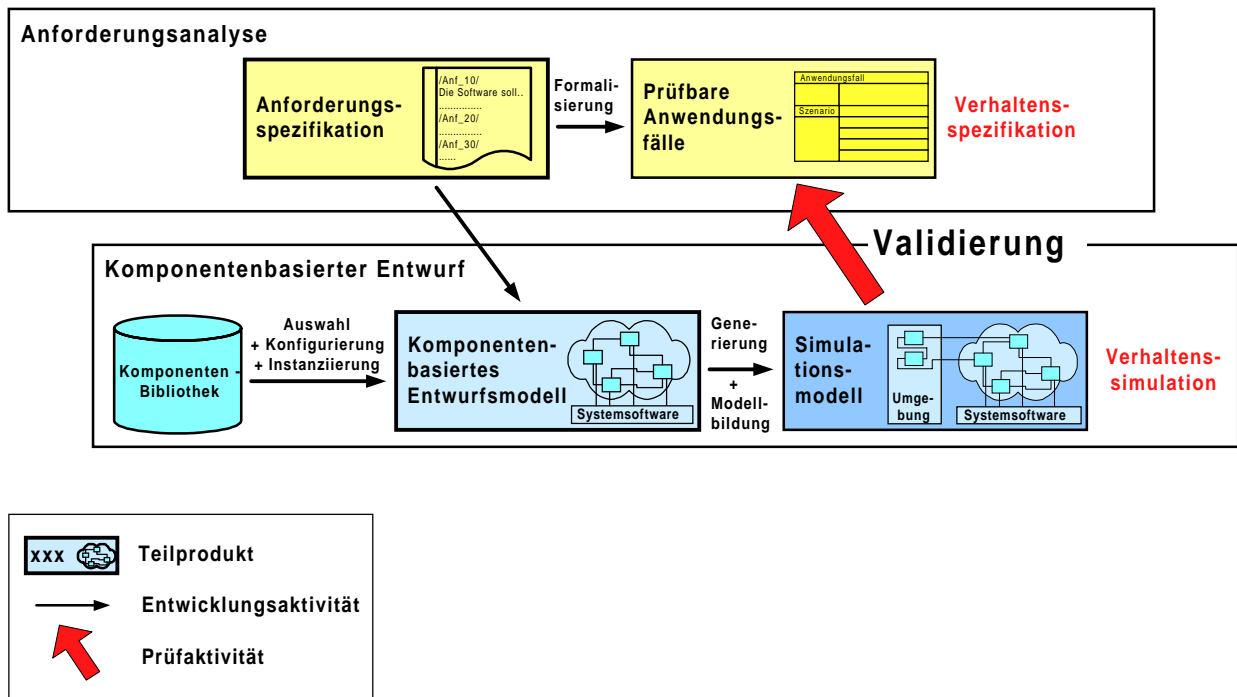


Abbildung 4.2: Grundkonzept für ein Prüfverfahren zur Validierung des dynamischen Verhaltens von komponentenbasierter Software für Echtzeitsysteme

Als Referenz für die Prüfung des simulierten Verhaltens der komponentenbasierten Software werden die Anforderungen aus Sicht des Anwenders zugrunde gelegt, welche bereits in der Anforderungsanalyse mit Hilfe von Anwendungsfällen als Teil der Anforderungsspezifikation erfasst wurden. Um eine systematische Prüfung des simulierten Verhaltens vorzubereiten, werden die Anwendungsfälle soweit formalisiert, dass sie in einer rechnerunterstützten, prüfbaren Notation vorliegen. Da die Anwendungsfälle bereits als Teilprodukt des konstruktiven Entwicklungsprozesses entstehen, kann im Vergleich zu bekannten Prüfverfahren, wie z.B. dem im Stand der Technik erläuterten Ansatz von Bergmann [Berg98], das Referenz-Verhalten mit relativ geringem Aufwand systematisch spezifiziert werden. Ausgehend von diesem Referenz-Verhalten können dann ebenfalls mit relativ geringem Aufwand systematisch Prüffälle abgeleitet werden.

Auf die einzelnen Bestandteile des Prüfverfahrens zur Validierung komponentenbasierter Software für Echtzeitsysteme wird in den nachfolgenden Kapiteln detaillierter eingegangen. In Kapitel 5 wird dazu die Spezifikation prüfbarer Anwendungsfälle erläutert. Kapitel 6 zeigt die Erstellung von Simulationsmodellen zur Verhaltenssimulation eines gesamten Echtzeitsystems. Die Details zur systematischen Prüffallentwicklung auf Basis der spezifizierten Anwendungsfälle sowie die Stimulation, Beobachtung und Bewertung des simulierten Verhaltens werden in Kapitel 7 betrachtet.

5 Spezifikation prüfbarer Anwendungsfälle

Als Referenz-Verhalten für die Validierung des simulierten Verhaltens von komponentenbasierter Software für Echtzeitsysteme sollen die in der Anforderungsanalyse erfassten Anwendungsfälle verwendet werden. In diesem Kapitel werden daher zunächst die Eigenschaften von Anwendungsfällen detaillierter erläutert und anschließend eine formale Notation zur prüfbaren Verhaltensspezifikation von Anwendungsfällen ausgewählt. Die ausgewählte Notation stellt geeignete Beschreibungsmittel zur Spezifikation des funktionalen und zeitlichen Verhaltens von Anwendungsfällen zur Verfügung und unterstützt gleichermaßen eine anschließende systematische Herleitung von Prüffällen daraus.

5.1 Anwendungsfälle zur Verhaltensspezifikation

5.1.1 Eigenschaften von Anwendungsfällen

Bei der komponentenbasierten Softwareentwicklung werden in der Anforderungsanalyse die Anforderungen an das Verhalten des zu entwickelnden Echtzeitsystems aus der Sicht des Anwenders mit Hilfe von Anwendungsfällen spezifiziert. In den Anwendungsfällen werden dazu charakteristische Abläufe für ein zu entwickelndes Echtzeitsystem beschrieben.

Anwendungsfälle sind ein Basiskonzept der objektorientierten Analyse (OOA) [JBR98, JCJÖ92] und eignen sich sehr gut zur frühen Spezifikation von Anforderungen an das Systemverhalten, da sie ohne tiefere Vorkenntnisse der objektorientierten Analyse oder einer speziellen formalen Syntax einfach mit informellem Text formuliert werden können. Durch ihre Systemsicht und ihren informellen Charakter verleiten sie auch nicht zu voreiligen Entwurfsentscheidungen während der Anforderungsanalyse. Außerdem bilden Anwendungsfälle nach der Klassifizierung von Balzert [Balz96] eine bewährte Vorstufe für die Erstellung formalisierter Verhaltensspezifikationen, wie z.B. bei der Anwendung von Petri-Netzen, Zustandsautomaten oder Interaktionsdiagrammen.

Ein Anwendungsfall ist die Beschreibung einer Menge von sequenziellen Aktionen (Szenarios³), einschließlich von Alternativen und Varianten, die ein System ausführt, um ein erkennbares, für einen externen Akteur nützliches Ziel zu erreichen. Die in den Anwendungsfällen beschriebenen Abläufe werden dabei von externen Akteuren ausgelöst. Bei der Spezifikation von Anwendungsfällen wird eine Außenbetrachtung des zunächst unbekanntes Systems zugrunde gelegt [JCJÖ92]. Das bedeutet, dass man zunächst nach Akteuren außerhalb des zu entwickelnden

³ Der Plural von Szenario ist laut Duden Szenarios

Systems sucht und von diesen wissen will, was sie von dem System erwarten. In einem Anwendungsfall wird in verschiedenen Szenarios gezeigt, wie das Ziel eines ausgewählten externen Akteurs erreicht wird. Ein externer Akteur ist dabei eine Rolle, die eine Person oder ein technisches System in Bezug auf eine bestimmte Angelegenheit spielt. Bei Echtzeitsystemen eignet sich nicht nur der Mensch als externer Akteur, sondern auch Sensoren, Nachbarsysteme und Zeitgeberdienste [HrRu01]. In fortgeschrittenen Ansätzen zur Anforderungsanalyse mit Anwendungsfällen, wie z.B. bei Cockburn [Cock97] oder bei Mattingly und Rao [MaRa98], werden neben externen Akteuren auch interne Akteure eines zu entwickelnden Systems verwendet. Diese Anwendungsfall-Sichtweise beinhaltet, dass ein zu entwickelndes System – insbesondere ein Softwaresystem – und seine Umgebung aus Akteuren bestehen, die miteinander interagieren. Bei der Interaktion kommunizieren die beteiligten externen und internen Akteure über Nachrichten miteinander. Als interne Akteure werden dabei das zu entwickelnde System selbst oder verwendete Teilsysteme auf unterster Abstraktionsebene verwendet. In Echtzeitsystemen bieten sich als interne Akteure beispielsweise die Aktoren von Automatisierungsrechnern bzw. einzelne Bestandteile des technischen Prozesses an.

5.1.2 UML Anwendungsfall-Diagramm

Für die Übersichtsdarstellung einer Menge von Anwendungsfällen bietet die „Unified Modeling Language“ (UML Standard) [UML99] das Anwendungsfall-Diagramm als grafische Notation an. Im Anwendungsfall-Diagramm können Akteure als Strichmännchen und die einzelnen Anwendungsfälle als Ovale dargestellt werden. Das zu entwickelnde System oder Teilsystem wird mit einem rechteckigen Kasten abgegrenzt. Abbildung 5.1 zeigt die wesentlichen Elemente eines Anwendungsfall-Diagramms.

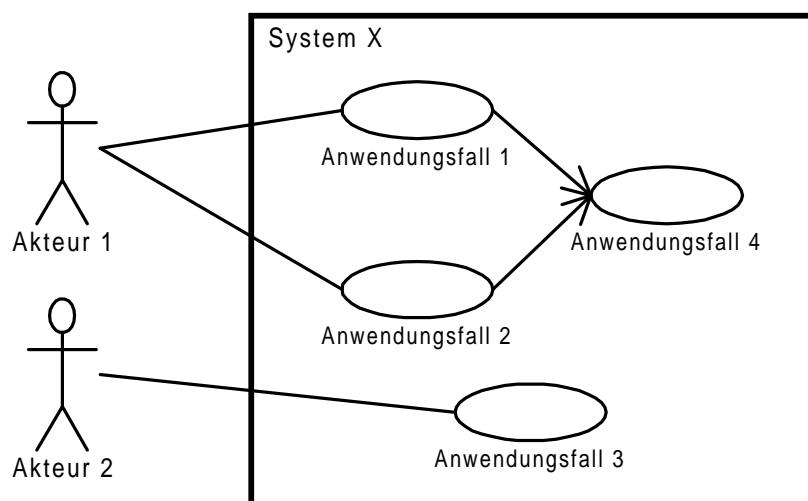


Abbildung 5.1: Aufbau eines Anwendungsfall-Diagramms

5.1.3 Spezifikationsschablone für Anwendungsfälle

Obwohl die Anwendungsfälle Teil des UML Standards [UML99] sind, sieht der Standard zur Spezifikation von Anwendungsfällen in der ersten Phase der Anforderungsanalyse nur informellen, umgangssprachlichen Text vor. Informeller Text bietet jedoch zu viele Freiheitsgrade, so dass eine stärkere Formalisierung der Anwendungsfälle sinnvoll ist. Zur Formalisierung der Anwendungsfälle schlagen deshalb verschiedene Autoren [Balz99, Cock97, MaRa98] die Verwendung einer Spezifikationsschablone und die Vorgabe von Heuristiken zum Ausfüllen der Spezifikationsschablone vor. Die einheitliche Struktur der Spezifikationsschablone vereinfacht sowohl die Erstellung von Anwendungsfällen als auch die anschließende systematische Herleitung von Prüffällen daraus.

Die in Tabelle 5.1 gezeigte tabellarische Spezifikationsschablone für Anwendungsfälle wurde in Anlehnung an die Vorlagen von Cockburn [Cock97] im Rahmen dieser Arbeit entwickelt [Flei00a, Flei99].

Tabelle 5.1: Spezifikationsschablone für Anwendungsfälle

Anwendungsfall	①	
Ziel	②	
Vorbedingungen	③	
Nachbedingungen	④	
Akteure	⑤	
Hauptszenario ⑥	Schritt	Aktion
	1.1	⑧
	1.2	⑨
	...	⑩
Szenario 2 ⑦	Schritt	Aktion
	2.1	⑪
	2.2	⑫
	...	⑬
Variationen	⑭	
Ausnahmen	⑮	
Anmerkungen	⑯	

Die Spezifikationsschablone unterstützt die Beschreibung einer oder mehrerer Folgen von sequenziell ablaufenden Aktionen, die als Szenarios bezeichnet werden. Jedes Szenario beschreibt dabei einen Ablauf, der versucht, das formulierte Ziel des Anwendungsfalls zu errei-

chen. Das Hauptszenario beschreibt den Ablauf, den ein Anwender bei der Anforderungsanalyse intuitiv als am wichtigsten einstuft. Die alternativen Szenarios sind optional und beschreiben alternative Abläufe zur Erreichung des Ziels. Mit den Vorbedingungen werden die Umstände beschrieben, die erfüllt sein müssen, damit ein Szenario ablaufen kann. Die Nachbedingungen müssen nach Ablauf eines Szenarios erfüllt sein. Mit ihnen können Bedingungen formuliert werden, wann ein Szenario das Ziel des Anwendungsfalls erreicht hat. Bei Ausnahmen werden in der Regel die Nachbedingungen nicht erfüllt. Variationen können verwendet werden, wenn ein Szenario in nur wenigen Bestandteilen von einem bereits spezifizierten Szenario abweicht. Dadurch werden redundante Beschreibungen vermieden.

Die Anwendungsfall-Spezifikationsschablone besitzt folgende Felder, die der Entwickler bei der Spezifikation eines Anwendungsfalls mit Inhalten ausfüllt:

- ① Anwendungsfallnummer und -name kennzeichnen einen Anwendungsfall durch einen eindeutigen Bezeichner und prägnanten Namen.
- ② Ziel, das durch den Anwendungsfall erreicht werden soll.
- ③ Vorbedingungen (als logischer Ausdruck von Systemvariablen bzw. Systemgrößen) kennzeichnen die Umstände, die erfüllt sein müssen, damit die im Anwendungsfall beschriebenen Szenarios ablaufen können (Muss-Zustand vorher). Bei Bedarf können hier auch mehrere Vorbedingungsalternativen beschrieben werden, die dann durch Nummerierung mit verschiedenen Buchstaben (a, b, c, ...) gekennzeichnet werden.
- ④ Nachbedingungen (als logischer Ausdruck von Systemvariablen bzw. Systemgrößen) kennzeichnen den Zustand (Muss-Zustand nachher), der erfüllt sein muss, wenn das Hauptszenario oder die alternativen Szenarios erfolgreich das Ziel des Anwendungsfalls erreicht haben.
- ⑤ Liste mit Akteuren (z.B. Benutzer oder technische Systeme), die während des Szenarios auf das System einwirken oder auf die das System einwirkt.
- ⑥ Das Hauptszenario (für jeden Anwendungsfall gibt es genau eines) soll den Ablauf beschreiben, den ein Anwender intuitiv als am wichtigsten einstuft, um das formulierte Ziel zu erreichen.
- ⑦ Weitere alternative Szenarios sind optional und beschreiben alternative Abläufe, die zum selben Ziel führen. Diese werden in der Regel durch eine andere Vorbedingung ausgelöst.
- ⑧ Der erste Schritt in einem Szenario findet sich nach folgender Regel: Unter welcher Vorbedingung wird durch welchen Akteur welche Nachricht an welchen anderen Akteur gesendet? Anmerkung: Bei der Entwicklung von Prüffällen werden später aus diesem Schritt meistens ein oder mehrere Nachrichten für die Stimulation eines Prüffalls hergeleitet. Im Automobilbereich wird hier z. B. oft „Fahrer (=externer Akteur) betätigt Schalter oder Hebel“ angegeben. Genauso gut kann aber auch ein System-interner Akteur die

stimulierende Nachricht auslösen.

- ⑨ Weitere Schritte eines Szenarios werden wie folgt beschrieben: Welcher Akteur sendet welche Nachricht an welchen anderen Akteur. Hier werden alle für den Ablauf des Szenarios benötigten Interaktionen angegeben. Optional können Zeitbedingungen zwischen den einzelnen Schritten angegeben werden. Werden Zeitbedingungen formuliert, so beziehen sie sich automatisch auf den vorherigen Schritt, ansonsten ist der Referenz-Schritt explizit anzugeben.
- ⑩ Variationen sind Szenarios, die sich nur geringfügig in einem Schritt von anderen Szenarios unterscheiden oder in denen nur die Akteure variiert werden.
- ⑪ Ausnahmen (z.B. Abbruchbedingungen) beschreiben Szenarios von Schritten, die die Akteure ausführen, wenn der Ablauf ungewöhnlich unterbrochen wird. Die Nachbedingung muss dann nicht unbedingt erfüllt sein.
- ⑫ Allgemeine Bemerkungen

Die tabellarische Anwendungsfall-Spezifikationsschablone ist sehr gut geeignet für eine informelle Spezifikation der Anforderungen an eine zu entwickelnde komponentenbasierte Software für ein Echtzeitsystem. Sie ist jedoch noch nicht formalisiert genug, um eine systematische, rechnerunterstützte Prüfung zu ermöglichen. Aus diesem Grund wird im folgenden Abschnitt 5.2 eine formale Notation ermittelt, mit der die Anwendungsfälle in ein prüfbares Format überführt werden können.

5.2 Ermittlung einer geeigneten Notation zur formalisierten Verhaltensspezifikation von Anwendungsfällen

Das in den Anwendungsfällen spezifizierte Verhalten soll als Referenz-Verhalten für die Validierung des simulierten Verhaltens von komponentenbasierter Software verwendet werden. Um eine automatisierte, rechnerunterstützte Prüfung zu ermöglichen, wird im folgenden eine geeignete Notation ermittelt, mit der das in den Anwendungsfällen als Text spezifizierte Verhalten in eine formalisierte Verhaltensspezifikation einfach und systematisch überführt werden kann.

Für die Spezifikation von Anforderungen bei der Softwareentwicklung haben sich nach Balzert [Balz96] mittlerweile eine überschaubare Anzahl (≈ 10) von Notationen und zugehörigen Basis-konzepten, unabhängig von den verwendeten Software-Entwicklungsmethoden, etabliert. Um die Akzeptanz des Prüfverfahrens zu gewährleisten, kommen nur diejenigen Notationen in die engere Wahl, die sich prinzipiell für die Spezifikation von zustands- und ereignisorientiertem Verhalten bei der Anforderungsanalyse eignen, weil dies den wesentlichen Bestandteil der Beschreibungen in den Anwendungsfällen ausmacht. Die Aktionsfolgen (Szenarios) der Anwendungsfälle sind ereignisorientiert, die Vor- und Nachbedingungen der Szenarios sind

zustandsorientiert. Weiterhin werden nur die grafischen Repräsentationen der Notationen berücksichtigt. In Tabelle 5.2 werden die vier in die Auswahl kommenden grafischen Notationen Zustandsdiagramm, Petri-Netz-Diagramm, Sequenzdiagramm und Kollaborationsdiagramm bezüglich ihrer Eignung zur Überführung des in den Anwendungsfällen spezifizierten Verhaltens und bezüglich der verfügbaren Beschreibungsmittel von Verhaltenseigenschaften von komponentenbasierter Software für Echtzeitsysteme bewertet.

Tabelle 5.2: Bewertung unterschiedlicher Diagrammartentypen zur formalisierten Verhaltensspezifikation von Anwendungsfällen

		Zustandsdiagramm	Petri-Netz-Diagramm	Sequenzdiagramm	Kollaborationsdiagramm
Beschreibungsmittel zur Verhaltensspezifikation	Nachrichtenorientierte Sicht (Schnittstellen-Sicht)	◐	◐	●	●
	Zustandsorientierte Sicht	●	●	◐	◐
	Sequenzielle Aktionen	●	●	●	●
	Nebenläufige Aktionen	◐	●	◐	◐
	Komplexe Aktionsfolgen	●	●	●	◐
	Quantitative Zeitbedingungen	●	◐	●	◐
	Qualitative Zeitbedingungen (Reihenfolge)	◐	◐	●	●
	Rechnerunterstützt, automatisierte Prüfbarkeit	●	●	●	●
Überführbarkeit aus Anwendungsfällen	Szenarios	◐	◐	●	●
	Bedingungen (Vor- und Nachbedingungen)	●	●	●	●
	Varianten	◐	◐	●	●
	Ausnahmen	◐	◐	●	●
	Überführungsaufwand	○	○	●	●

● gut geeignet ◐ teilweise geeignet ○ ungeeignet

Die Syntax der ausgewählten Diagrammart ist, mit Ausnahme der Petri-Netz-Diagramme, im UML Standard [UML99] festgelegt. Die Sequenzdiagramme finden sich auch in dem bereits älteren MSC Standard ITU-TS Z.120-1994 [ITU120] im Telekommunikationsbereich, einer Erweiterung des SDL Standards ITU-TS Z.100-1988 [ITU100].

Die Bewertung zeigt, dass alle vier Diagrammart zum wenigsten teilweise zur Verhaltensspezifikation von Anwendungsfällen für Echtzeitsysteme geeignet sind.

Bezüglich der Überführbarkeit aus der Anwendungsfall-Spezifikationsschablone sind die Sequenz- und Kollaborationsdiagramme deutlich besser geeignet als Zustandsdiagramme oder Petri-Netz-Diagramme, da sie direkt die Beschreibung der Interaktionen und Kommunikation zwischen den Akteuren mit einer nachrichtenorientierten Sichtweise unterstützen. Beide Diagrammart werden zusammen auch als Interaktionsdiagramme bezeichnet. Bei einer Überführung in Sequenz- oder Kollaborationsdiagramme muss prinzipiell für jedes einzelne Szenario eines Anwendungsfalls, sowie Varianten oder Ausnahmen davon, ein eigenständiges Diagramm erstellt werden. Mit einem Zustandsdiagramm oder Petri-Netz-Diagramm kann prinzipiell ein kompletter Anwendungsfall mit alternativen Szenarios sowie Varianten und Ausnahmen spezifiziert werden. Die Überführung ist aber wesentlich aufwändiger und fehleranfälliger, da bei der Spezifikation mit diesen Diagrammart mehr die Systemzustände und nicht die Interaktionen und Kommunikation zwischen Akteuren im Mittelpunkt stehen.

Die Sequenzdiagramme sind im Vergleich zu den Kollaborationsdiagrammen wesentlich besser geeignet, weil mit ihnen auch komplexere Aktionsfolgen ohne Einschränkung der Übersichtlichkeit spezifiziert werden können. Außerdem ist die in den Kollaborationsdiagrammen zusätzlich enthaltene Strukturinformation bei einer Prüfung, die auf das dynamische Verhalten von komponentenbasierter Software abzielt, eher einschränkend als nützlich. Auf Basis der erfolgten Bewertung werden die UML Sequenzdiagramme zur Verhaltensspezifikation von formalisierten, prüfaren Anwendungsfällen ausgewählt.

5.3 Sequenzdiagramme zur formalisierten Verhaltensspezifikation von Anwendungsfällen

5.3.1 UML Sequenzdiagramme

Mit Hilfe der Sequenzdiagramme werden die spezifizierten Anwendungsfälle in eine formale, prüfbare Notation überführt. Jedes Szenario eines Anwendungsfalls, sowie Varianten oder Ausnahmen davon, werden dabei in ein einzelnes Sequenzdiagramm überführt. Deshalb werden an dieser Stelle die zur Verhaltensspezifikation von Anwendungsfällen notwendigen Eigenschaften der Sequenzdiagramme aus dem UML Standard [UML99] kurz vorgestellt.

Die grafische Notation eines Sequenzdiagramms hebt die zeitliche Reihenfolge von Nachrichten

zwischen Akteuren hervor. Wie in Abbildung 5.2 zu sehen, werden im Sequenzdiagramm die an dem Szenario beteiligten Akteure auf der x-Achse angeordnet. An Stelle von Akteuren können auch Komponenten oder Objekte gesetzt werden. Die gestrichelten vertikalen Linien zeigen die Lebenslinien der einzelnen Akteure an. Die zwischen den Akteuren versendeten und empfangenen Nachrichten werden als gerichtete Pfeile in aufsteigender zeitlicher Reihenfolge von oben nach unten entlang der y-Achse aufgetragen. Die Nachrichten können optional mit einer Bedingung in „[“ Klammern als Präfix vor dem Nachrichten-Bezeichner versehen werden, von deren Erfüllung dann das Versenden der Nachricht abhängt. Nachrichten können an der angegebenen Stelle in der Sequenz auch mehrfach versendet werden. Dies wird mit dem Suffix „*“ nach dem Nachrichten-Bezeichner dargestellt.

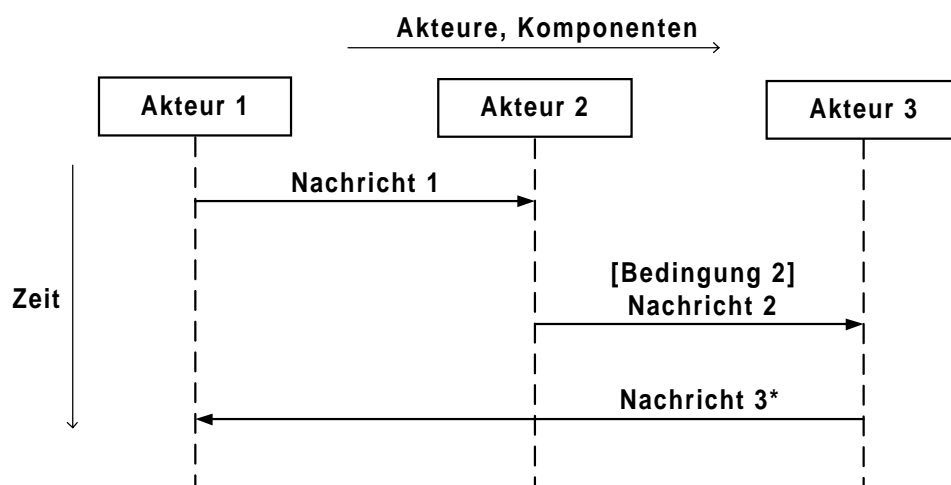


Abbildung 5.2: Aufbau eines UML Sequenzdiagramms

Die Nachrichten können Ereignisse oder Daten übermitteln, mit denen Aufträge und Informationen an andere Akteure versendet werden. Die Nachrichten können beim Versenden und beim Empfangen Aktionen auslösen, die wiederum das Versenden weiterer Nachrichten veranlassen. Weiterhin sind im UML Standard auch Beschreibungsmittel für Zeitbedingungen zwischen Nachrichten vorgesehen. Es handelt sich dabei aber um eine relativ oberflächliche Festlegung, die durch den Benutzer bzw. durch die Hersteller von UML Modellierungswerkzeugen mit einer präziseren Semantik versehen werden kann [Doug98, UML99]. Die notwendige Festlegung für die vorliegende Arbeit wird neben anderen Erweiterungen im nachfolgenden Abschnitt entwickelt.

5.3.2 Erweiterung der Eigenschaften von UML Sequenzdiagrammen

Mit den standardisierten Eigenschaften der UML Sequenzdiagramme lassen sich die sequenziellen Schritte von Aktionen eines Anwendungsfall-Szenarios als Sequenz von Nachrichten in ein Sequenzdiagramm überführen. Für eine komfortable und vollständige Überführung der anderen Informationen eines Anwendungsfalls sind Erweiterungen in der Notation der

UML Sequenzdiagramme notwendig. So müssen zusätzlich auch Vor- und Nachbedingungen sowie Echtzeitanforderungen spezifiziert werden können. Die dazu notwendigen Erweiterungen werden im folgenden vorgestellt.

Vorbedingungen

Eine Vorbedingung gibt einen Zustand an, der erfüllt sein muss, damit ein Szenario ablaufen kann (Zustand vorher). Die Bedingung zur Feststellung dieses Zustands wird durch einen logischen Ausdruck mit Systemvariablen bzw. Systemgrößen spezifiziert. Es bietet sich an, diese Bedingung mit der ersten Nachricht im Sequenzdiagramm zu formulieren. Zur Spezifikation einer Vorbedingung wird folgende Syntax definiert:

Vorbedingung: [PRECONDITION : <condition>]

<condition> ist ein logischer Ausdruck mit Systemvariablen bzw. Systemgrößen, wie z.B. „SchalterX == Ein“. Die Spezifikation einer Vorbedingung ist wie eine Bedingung optional.

Nachbedingungen

Eine Nachbedingung gibt einen Zustand an, der nach Ablauf eines Szenarios erfüllt sein muss (Zustand nachher). Die Bedingung zur Feststellung dieses Zustands wird ebenfalls durch den logischen Ausdruck <condition> mit Systemvariablen bzw. Systemgrößen spezifiziert. Da sich diese Bedingung keiner Nachricht zuordnen lässt, wird eine zusätzliche Pseudo-Nachricht in der Sequenz eingeführt. Die Pseudo-Nachricht wird als letzte Nachricht im Sequenzdiagramm zwischen zwei beliebigen Akteuren ausgetauscht und besteht nur aus einer Bedingung, jedoch keinem Nachrichten-Bezeichner. Zur Spezifikation einer Nachbedingung mit Hilfe einer Pseudo-Nachricht wird folgende Syntax definiert:

Nachbedingung: [POSTCONDITION : <condition>]

Die Spezifikation einer Nachbedingung mit einer Pseudo-Nachricht ist ebenfalls optional. Ein Beispiel für ein Sequenzdiagramm mit Vor- und Nachbedingung zeigt Abbildung 5.3.

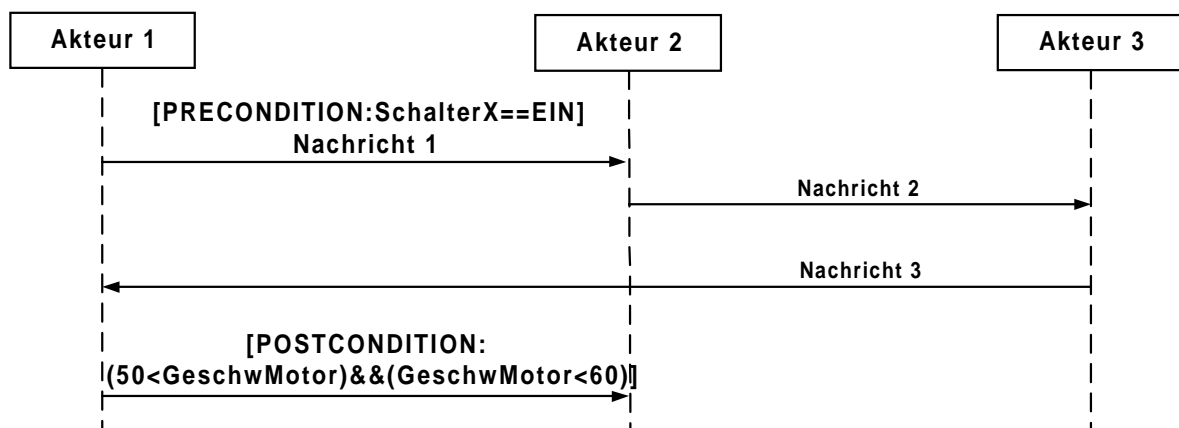


Abbildung 5.3: Sequenzdiagramm mit Vor- und Nachbedingung

Zeitbedingungen

Die Prüfung des Echtzeitverhaltens ist ein wichtiger Gegenstand bei der Validierung von Software für Echtzeitsysteme. Bei der Prüfung muss insbesondere die Einhaltung der Merkmale Rechtzeitigkeit, Gleichzeitigkeit und Vorhersagbarkeit berücksichtigt werden. Eine grundlegende gemeinsame Eigenschaft dieser Merkmale ist, dass in Echtzeitsystemen Reaktionen innerhalb vorgegebener und vorhersagbarer Zeitschranken erfolgen müssen [LaGö99]. Zeitanforderungen bei Interaktionen an den Schnittstellen eines Echtzeitsystems müssen durch technisch bedingte Toleranzen als Zeitintervalle definiert werden [Berg98, Dasa85, StRa92]. Grundlage hierfür bildet das Verhalten bezüglich Stimulus und Antwort aus schnittstellenorientierter Sicht. Die Definition einer Zeitanforderung erfolgt dabei relativ zum Zeitpunkt des Entstehens der Anforderung durch die Stimulus-Nachricht. Auf diese Anforderung muss der Akteur⁴ bzw. mehrere Akteure im Verbund eine Bearbeitung starten und eine Reaktion ausführen. Die Bearbeitung der Anforderung kann mit einem Zeitverzug gestartet werden. Für die Reaktion wird als Zeitanforderung der Zeitraum definiert, in dem die Reaktion zu erfolgen hat. Für den Zeitpunkt, an dem die Zeitanforderung durch die abschließende Antwort-Nachricht erfüllt sein muss, kann sowohl eine minimale Frist t_{\min} als auch eine maximale Frist t_{\max} oder beides definiert sein. Die Zeitanforderung für eine Reaktion zwischen zwei Nachrichten kann formal in Form von Intervallen gefasst werden. Folgende Zeitintervalle sind dabei möglich:

$$\begin{array}{ll} (0 ; t_{\max}] & : \quad 0 < t_{\max} < \infty \\ [t_{\min} ; t_{\max}] & : \quad 0 < t_{\min} < t_{\max} < \infty \\ [t_{\min} ; \infty) & : \quad 0 < t_{\min} < \infty \end{array}$$

Bereits im vorigen Abschnitt wurde festgestellt, dass im UML Standard für die Spezifikation von quantitativen Zeitbedingungen in Sequenzdiagrammen noch Freiheitsgrade bestehen. Um auch Echtzeitanforderungen in Form von Zeitintervallen zwischen zwei Nachrichten spezifizieren zu können, wird dazu die grafische Notation des Sequenzdiagramms für quantitative Zeitbedingungen zwischen zwei beliebigen Nachrichten erweitert, wie in Abbildung 5.4 dargestellt.

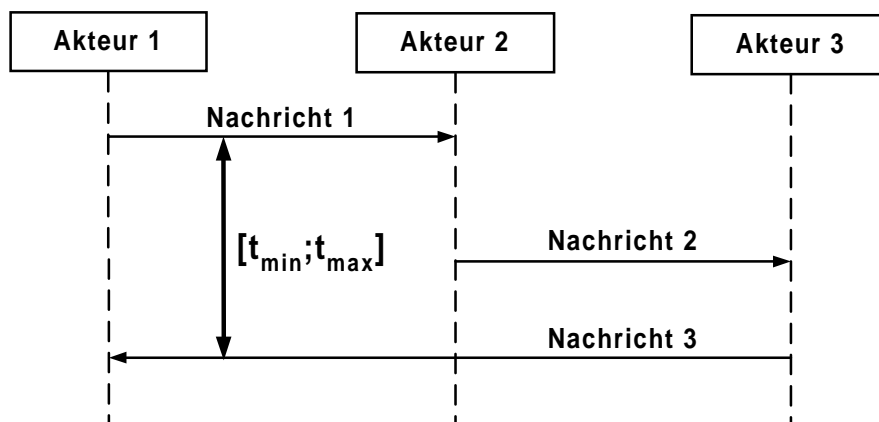


Abbildung 5.4: Sequenzdiagramm mit Zeitbedingung

⁴ Dies gilt auch für die Abläufe zwischen den Komponenten im komponentenbasierten Entwurfsmodell

Die grafische Notation ermöglicht eine übersichtliche Darstellung der für Echtzeitsysteme wichtigen Zeitbedingungen. Zeitbedingungen zwischen zwei Nachrichten werden durch einen Pfeil mit Doppelspitze in y-Richtung und der Zeitanforderung als Zeitintervall grafisch spezifiziert. Die zwei Nachrichten müssen in der Sequenz nicht direkt aufeinander folgen.

Periodische Sequenzen

Neben den quantitativen Zeitbedingungen müssen zur Prüfung des Echtzeitverhaltens auch qualitative Zeitanforderungen spezifiziert werden können. Belschner [Bels97] und Pereira [Pere95] klassifizieren diese als zeitpunktbasierte Anforderungen. Bei den zeitpunktbasierten Anforderungen wird in einmaliges Auftreten, periodisches Auftreten und sequenzielles Auftreten unterschieden. Ein Zeitpunkt ist eine zeitliche Markierung eines für die Verhaltensspezifikation wichtigen Geschehens. Es wird angenommen, dass ein Zeitpunkt keine zeitliche Ausdehnung hat.

In einem Sequenzdiagramm werden Zeitpunkte durch das Versenden oder Empfangen einer Nachricht definiert. Im standardisierten UML Sequenzdiagramm kann das einmalige Auftreten einer Nachricht und das sequenzielle Auftreten mehrerer Nachrichten spezifiziert werden. Die Spezifikation von periodischem Auftreten einer Nachricht ist nicht vorgesehen. Aus diesem Grund wird die periodische Wiederholung von Teil-Sequenzen innerhalb eines Sequenzdiagramms zusätzlich eingeführt. Eine periodische Teil-Sequenz wird in der Bedingung der letzten Nachricht der Teil-Sequenz als Hinweis auf die zum Rücksprung bestimmte erste Nachricht der Teil-Sequenz definiert. Die Syntax zur Spezifikation periodischer Teil-Sequenzen ist wie folgt:

[REPEAT <Nachrichten-Bezeichner erste Nachricht> UNTIL <condition>]

Die Teil-Sequenz muss dann solange periodisch wiederholt werden, bis der logische Ausdruck <condition> mit den Systemvariablen bzw. Systemgrößen wahr ist. Bei mehrfachem Auftreten des selben Nachrichten-Bezeichners innerhalb eines Sequenzdiagramms wird auf das zeitlich erstmalige Auftreten des Nachrichten-Bezeichners zurückgesprungen. Abbildung 5.5 zeigt ein Beispiel für ein Sequenzdiagramm mit zwei periodischen Teil-Sequenzen.

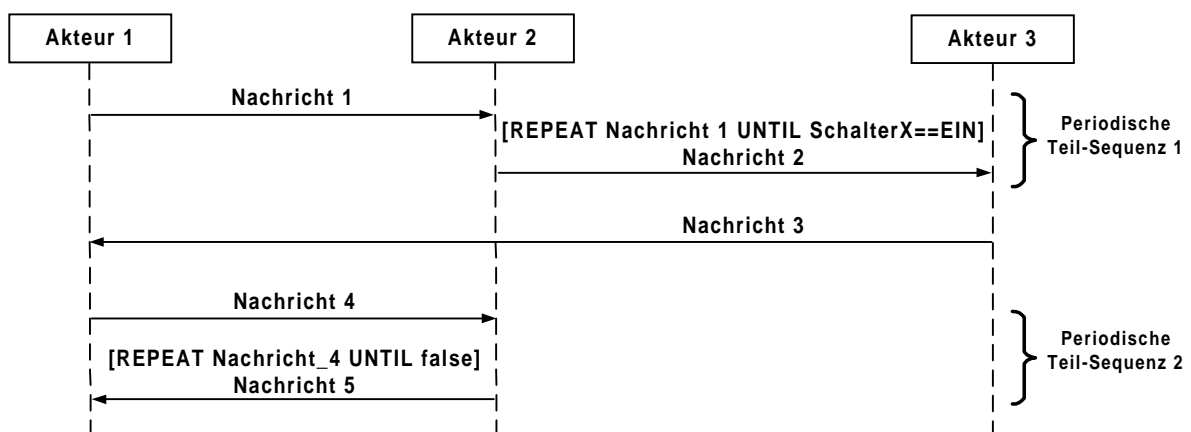


Abbildung 5.5: Sequenzdiagramm mit periodischen Teil-Sequenzen

Werden mehrere periodische Teil-Sequenzen in einem Sequenzdiagramm spezifiziert, darf keine Überlappung der Teil-Sequenzen entstehen. In der Praxis tritt oft der Spezialfall auf, dass die gesamte Sequenz periodisch wiederholt wird. Als weiterer Spezialfall kann die periodische Teil-Sequenz auch aus nur einer Nachricht bestehen. In der „REPEAT“-Anweisung wird dann die Nachricht selbst als Rücksprung-Nachricht spezifiziert. Die optionale zeitliche Anforderung an die Periodendauer wird dabei über die Zeitbedingung zur vorhergehenden Bezugs-Nachricht spezifiziert.

Sequenztyp

Wenn Anwendungsfälle während der Anforderungsanalyse spezifiziert werden, ist häufig deren Geltungsbereich und Verbindlichkeit noch nicht exakt festgelegt. Mit den Sequenzdiagrammen sollen die einzelnen Szenarios eines Anwendungsfalls jedoch in verbindliche, prüfbare Anwendungsfälle überführt werden. Es stellt sich also die Frage, ob ein Szenario bei erfüllter Vorbedingung immer als vollständige Sequenz ablaufen muss, oder ob es nur bei expliziter Stimulation als vollständige Sequenz ablaufen muss. Damm und Harel [DaHa98] unterscheiden beispielsweise in ihrer Erweiterung der UML Sequenzdiagramme zu „Live Sequence Charts“ (LSC) zwischen zwingenden und beispielhaften Szenarios.

Zur präzisen Festlegung des Geltungsbereichs bei der Prüfung jedes Anwendungsfall-Szenarios wird deshalb für Sequenzdiagramme das zusätzliche Attribut Sequenztyp eingeführt [Flei00b]. Dem Attribut können drei unterschiedliche Werte zugeordnet werden. Die Bedeutung der Werte bezüglich der Prüfung ist folgendermaßen definiert:

- **Normal:** Mindestens ein Ablauf in der entwickelten komponentenbasierten Software erfüllt das Szenario vollständig, wenn die Vorbedingung erfüllt ist.
- **Zwingend:** Alle Abläufe in der entwickelten komponentenbasierten Software erfüllen das Szenario vollständig, wenn die Vorbedingung erfüllt ist.
- **Verboten:** Kein Ablauf in der entwickelten komponentenbasierten Software darf das Szenario vollständig erfüllen, wenn die Vorbedingung erfüllt ist.

„Normal“ eignet sich allgemein zur Spezifikation von Szenarios, bei denen man noch nicht sicher ist, ob sie nicht auch als Teil-Sequenz einer anderen Sequenz mit beispielsweise veränderten Zeitbedingungen auftreten können. „Zwingend“ bedeutet, dass bei Eintreten der Vorbedingung die Sequenz von Nachrichten auf jeden Fall so ablaufen muss. Der Sequenztyp „Verboten“ eröffnet die Möglichkeit unerwünschte Abläufe explizit zu spezifizieren. Szenarios von diesem Sequenztyp werden in der Regel aus den Ausnahme-Beschreibungen der Anwendungsfälle hergeleitet. Außerdem eignen sie sich zur Spezifikation von zusätzlichen ablauforientierten Anforderungen, die nicht mit Hilfe der Anwendungsfälle analysiert wurden. Diese ergänzenden Szenarios stammen beispielsweise aus Sicherheitsanalysen [Bert00], die gefährliche und unerwünschte Abläufe identifizieren.

5.4 Überführung der Anwendungsfälle in formalisierte Anwendungsszenarios

Mit der im vorangegangenen Abschnitt entwickelten Notation der erweiterten Sequenzdiagramme können die tabellarisch spezifizierten Anwendungsfälle nun in eine formalisierte Verhaltensspezifikation überführt werden, welche das Referenz-Verhalten für die spätere Prüfung des simulierten Verhaltens der komponentenbasierten Software in einem rechnerunterstützt prüfbareren Format bildet.

Abbildung 5.6 zeigt exemplarisch die Herleitung von drei Anwendungsszenarios aus einem Anwendungsfall. Das im Vordergrund dargestellte Anwendungsszenario W1_1 ist aus Platzgründen nicht vollständig abgebildet. Für die Überführung der einzelnen Felder eines tabellarischen Anwendungsfalls in die formalisierte Notation eines erweiterten Sequenzdiagramms können nachfolgende Heuristiken angewandt werden:

- Aus jedem Szenario (Hauptszenario und alternative Szenarios) wird mindestens ein Anwendungsszenario hergeleitet.
- Für jede spezifizierte Variation bzw. Ausnahme im Anwendungsfall wird ebenfalls ein eigenes Anwendungsszenario erstellt.
- Die im Szenario eines Anwendungsfalls auftretenden Akteure werden direkt als Akteure in das Anwendungsszenario übernommen.
- Jeder einzelne Schritt eines Szenarios im Anwendungsfall ergibt eine Nachricht im Anwendungsszenario.
- Zeitangaben im Text eines Schrittes in einem Szenario werden als Zeitbedingung einer Nachricht im Anwendungsszenario spezifiziert.
- Bei Referenzen auf andere Anwendungsfälle wird der Bezeichner des referenzierten Anwendungsfalls als Akteur in das Anwendungsszenario aufgenommen.
- Ist in einem Schritt eines Szenarios im Anwendungsfall ein periodisches Auftreten eines oder mehrerer Schritte spezifiziert, wird dies bei der zugeordneten Nachricht im Anwendungsszenario als „REPEAT ... UNTIL“ Bedingung umgesetzt.
- Ist im Anwendungsfall eine Vorbedingung spezifiziert, so wird diese als „PRECONDITION“ in der ersten Nachricht des Anwendungsszenarios formuliert.
- Ist im Anwendungsfall eine Nachbedingung spezifiziert, wird am Ende des Anwendungsszenarios als letzte Nachricht eine Pseudo-Nachricht mit einer „POSTCONDITION“ ergänzt.

Anhand der aufgeführten Heuristiken ist leicht zu erkennen, dass die ursprünglich als tabellarischer Text spezifizierten Anwendungsfälle mit geringem Aufwand in das rechnerunterstützt prüfbare Format der Anwendungsszenarios überführt werden können.

Anwendungsfall	W1 Langsam Wischen	
Ziel	Scheibe von Regenwasser befreien	
Vorbedingung	Zündung ein (Klemme 15 „Ein“)	
Nachbedingung	Wischerblätter in Parkposition	
Akteure	Fahrer, Lenkstockhebel, Steuergerät, Scheibenwischermotor, Nullposition-Sensor	
Hauptzenario	Schritt	Aktion
	1.0	Fahrer stellt Lenkstockhebel auf Position „Langsam Wischen“
	1.1	Lenkstockhebel liefert dem Steuergerät den Betriebsmodus „Langsam Wischen“
	1.2	Steuergerät startet Scheibenwischermotor mit Wischgeschwindigkeit „Langsam“ nach max. 100ms
	1.3	Steuergerät aktiviert die kontinuierliche Überlast-Diagnose (→ Anwendungsfall „W5 Überlast Diagnostizieren“)
	1.4	Fahrer stellt Lenkstockhebel auf Position „Aus“
	1.5	Lenkstockhebel liefert dem Steuergerät den Betriebsmodus „Aus“
	1.6	Nullposition-Sensor liefert dem Steuergerät die Position „Ruhelage“
1.7	Steuergerät stoppt den Scheibenwischermotor nach max. 20ms	
Variationen	1.1a	Fahrer stellt Lenkstockhebel auf Position „Schnell Wischen“. Fortfahren bei (→ Anwendungsfall „W2 Schnell Wischen“).
	1.1b	Fahrer stellt Lenkstockhebel auf Position „Automatisch Wischen“. Fortfahren bei (→ Anwendungsfall „W3 Automatisch Wischen“).
Ausnahmen	- keine -	
Anmerkungen	- keine -	

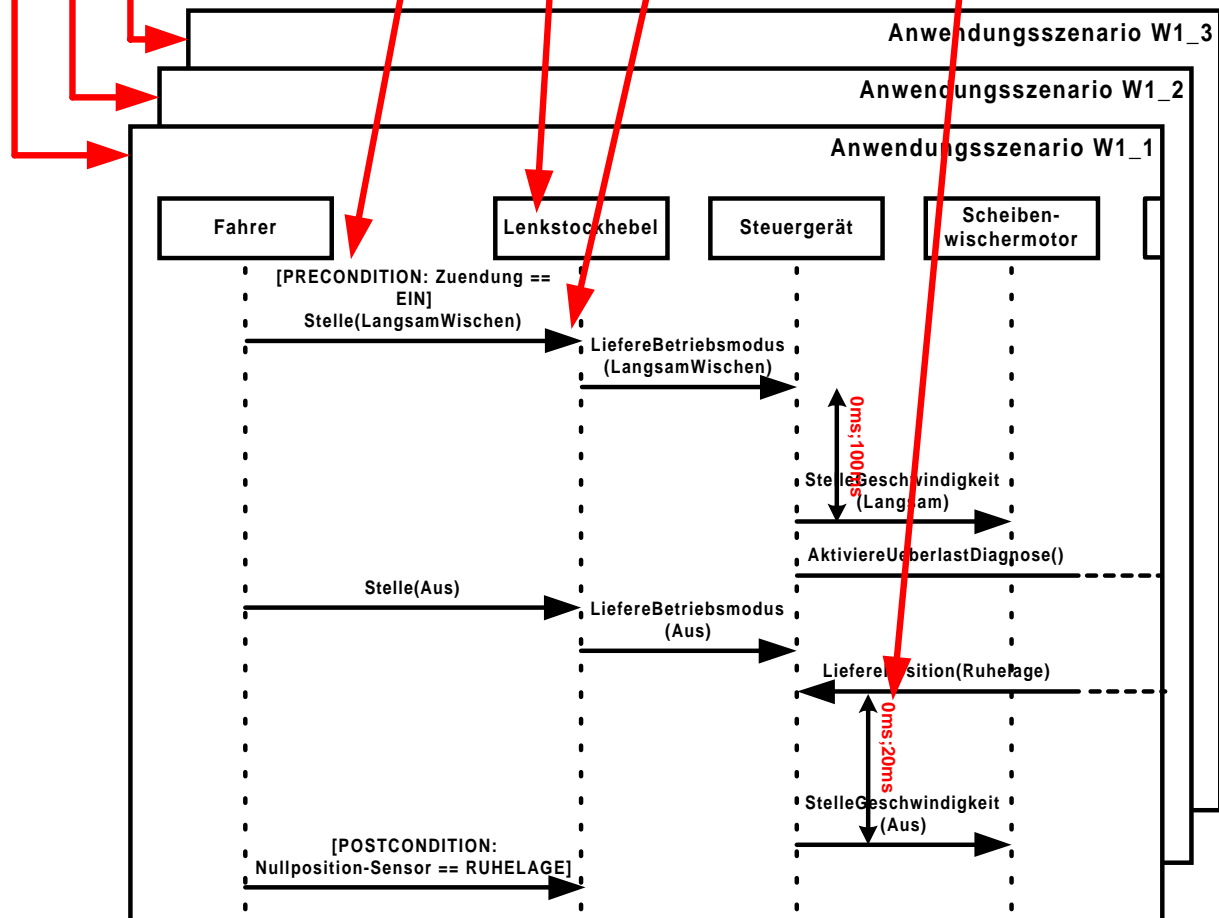


Abbildung 5.6: Beispiel für die Herleitung von Anwendungsszenarios aus einem Anwendungsfall

Mit den in diesem Kapitel eingeführten semantischen und syntaktischen Erweiterungen für die UML Sequenzdiagramme steht eine formale Notation zur Spezifikation prüfbarer Anwendungsfälle zur Verfügung. Die während der Anforderungsanalyse in tabellarischen Anwendungsfällen erfassten Anforderungen an das dynamische Verhalten einer komponentenbasierten Software können mit Hilfe der erweiterten UML Sequenzdiagramme mit geringem Aufwand in eine formalisierte, prüfbare Verhaltensspezifikation überführt werden. Die formalisierte Verhaltensspezifikation der Anwendungsfälle besteht dann aus einer Menge von Sequenzdiagrammen und beschreibt das Referenz-Verhalten für die Prüfung gegenüber dem simulierten Verhalten eines komponentenbasierten Entwurfsmodells. Auf die systematische Entwicklung von Prüffällen aus einer Verhaltensspezifikation und auf die verwendete Prüfstrategie wird in Kapitel 7 näher eingegangen. Zunächst wird jedoch im nachfolgenden Kapitel 6 die Erstellung des Simulationsmodells zur Verhaltenssimulation detaillierter behandelt.

6 Erstellung des Simulationsmodells

Da beim Entwurf komponentenbasierter Software für Echtzeitsysteme bereits vollständig spezifizierte Komponenten verwendet werden, kann das Verhalten der komponentenbasierten Software schon in dieser frühen Entwicklungsphase mit Hilfe von Simulation validiert werden. Dazu ist jedoch ein Simulationsmodell erforderlich, das neben der komponentenbasierten Software auch ausführbare Simulationsmodell-Komponenten für die Umgebung des Echtzeitsystems enthält. Auf die Erstellung des Simulationsmodells wird in diesem Kapitel detaillierter eingegangen. Zunächst werden die Anforderungen an den Aufbau des Simulationsmodells formuliert. Anschließend wird ein geeignetes Simulationswerkzeug ausgewählt, welches die Generierung der Simulationsmodell-Komponenten für die komponentenbasierte Software ermöglicht und die Modellierung der Umgebung des Echtzeitsystems unterstützt. Für die Simulation von Echtzeitsystemen mit einzelnen und mit verteilten Steuergeräten werden zwei unterschiedliche Lösungskonzepte vorgestellt.

6.1 Anforderungen an den Aufbau des Simulationsmodells

6.1.1 Realitätsnahes simuliertes Verhalten

Die technischen Möglichkeiten zur Erstellung eines Simulationsmodells sind sehr vielfältig. Simulationsmodelle können prinzipiell programmiert oder aus Simulationsbibliotheks-Komponenten zusammengesetzt werden [MaMe88]. Im günstigsten Fall wird die Erstellung durch ein Simulationswerkzeug unterstützt. Zur Auswahl eines geeigneten Simulationswerkzeugs werden in den nächsten Abschnitten die Anforderungen an das Simulationsmodell festgelegt.

Das allgemeine Ziel der Modellerstellung ist die Realität möglichst exakt nachzubilden. Neben der reinen Funktionalität ist es wichtig, auch das zeitliche Verhalten möglichst exakt nachzubilden. Werden aus dem komponentenbasierten Entwurfsmodell der Software ausführbare Simulationsmodell-Komponenten generiert und in das Simulationsmodell des Echtzeitsystems eingebunden, sind die Ausführungszeiten auf dem Simulationsrechner im Allgemeinen unterschiedlich im Vergleich zu den Ausführungszeiten auf dem realen Automatisierungsrechner. Für eine realitätsnahe Simulation der komponentenbasierten Software müssen deshalb Verzögerungszeiten und von Zeitgeberdiensten ausgelöste Ereignisse in der Simulationszeit möglichst vergleichbar zur realen Uhrzeit auftreten.

Um das Verhalten einer komponentenbasierten Software mit Hilfe eines Simulationsmodells prüfen zu können, ist es notwendig, neben der komponentenbasierten Software auch alle

anderen wesentlichen Bestandteile des Echtzeitsystems im Simulationsmodell zu berücksichtigen [FRB97]. Die anderen wesentlichen Bestandteile des Echtzeitsystems werden im Simulationsmodell in einem Modell der Umgebung nachgebildet. Zur gezielten Simulation der zu prüfenden Anwendungsfälle müssen im Simulationsmodell außerdem Stimulations-Daten zeitgesteuert eingespeist werden können. Für diesen Zweck enthält das Simulationsmodell eine Stimulations-Daten-Generator-Komponente. Die wesentlichen Bestandteile des Simulationsmodells sind in Abbildung 6.1 dargestellt.

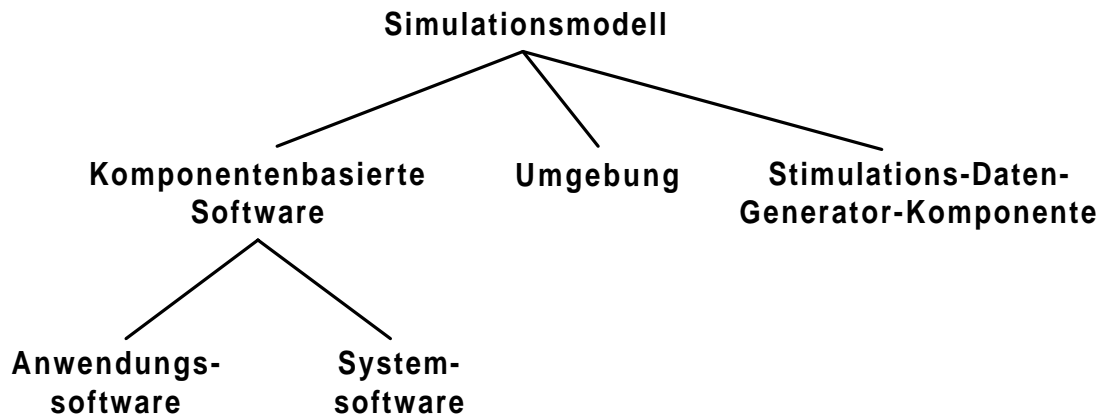


Abbildung 6.1: Wesentliche Bestandteile des Simulationsmodells

Auf die Anforderungen an die einzelnen Bestandteile des Simulationsmodells wird in den folgenden Abschnitten eingegangen.

6.1.2 Modell der komponentenbasierten Software

Die komponentenbasierte Software wird bereits beim komponentenbasierten Entwurf modelliert. Für die Erstellung des Simulationsmodells müssen die Komponenten des komponentenbasierten Entwurfsmodell deshalb in ausführbare Simulationsmodell-Komponenten überführt werden. Am einfachsten ist die Überführung durch automatische Generierung mit Hilfe eines kombinierten Entwurfs- und Simulationswerkzeugs. Das Simulationsmodell muss dazu alle wesentlichen Bestandteile der Anwendungssoftware und der Systemsoftware berücksichtigen. Abbildung 6.2 zeigt ein Beispiel für ein Modell der komponentenbasierten Software.

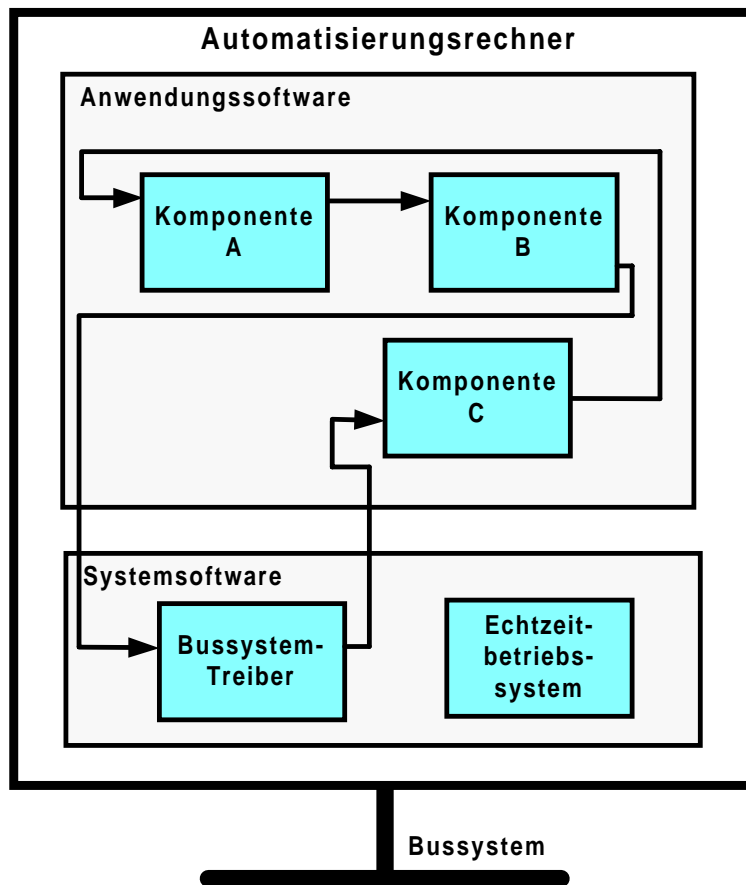


Abbildung 6.2: Modell der komponentenbasierten Software im Simulationsmodell

Für komplexere Komponenten der Systemsoftware, wie z.B. eine Echtzeitbetriebssystem-Komponente oder eine Bussystem-Treiber-Komponente, entstehen bei einer Generierung rechenzeitintensive Simulationsmodell-Komponenten. Aus diesem Grund ist es sinnvoll an deren Stelle idealisierte Simulationsmodell-Komponenten zu verwenden. Einige Simulationswerkzeuge besitzen dafür bereits vorgefertigte Simulationsmodell-Komponenten.

6.1.3 Modell der Umgebung

Die anderen Bestandteile des Simulationsmodells müssen deduktiv modelliert werden. Das bedeutet, dass sie durch die Analyse des technischen Prozesses und der Schnittstellen des Automatisierungsrechners durch Abstraktion bzw. Idealisierung von dessen zukünftiger Realität gewonnen werden. Die Bestandteile des technischen Prozesses können durch ihren materiellen Charakter sehr leicht als Komponenten identifiziert und dadurch ebenfalls komponentenbasiert modelliert werden. Die identifizierten Komponenten werden im Simulationsmodell als Modell der Umgebung (siehe Abbildung 6.3) zusammengefasst.

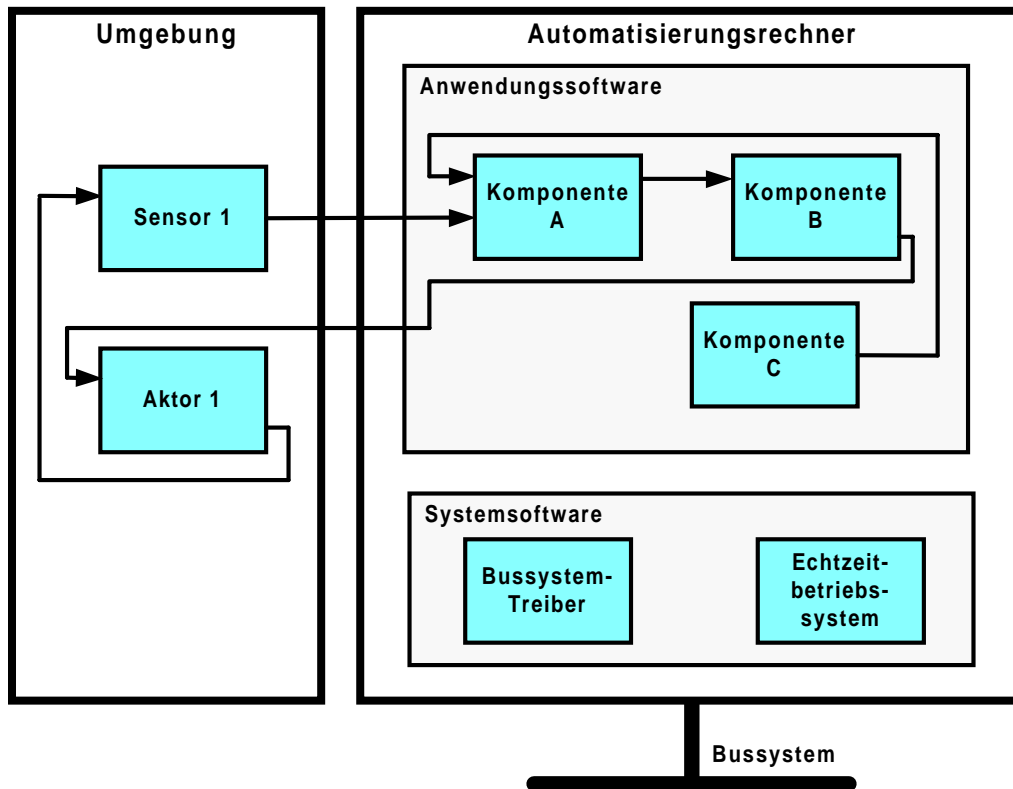


Abbildung 6.3: Modell der Umgebung im Simulationsmodell

Zum Modell der Umgebung gehören:

- Benutzer, z.B. ein Fahrer eines Kraftfahrzeugs
- Einfache Komponenten ohne Rückkopplung, z.B. Sensoren und Aktoren
- Rückkopplungsbehaftete Komponenten, z.B. mechanische Bauelemente

Besitzt der zu simulierende technische Prozess geschlossene Regelkreise, so muss das Modell der Umgebung ein Streckenmodell zur Nachbildung des Verhaltens der Regelstrecke enthalten. Bei einem Elektromotor befinden sich die rückkopplungsbehafteten Bestandteile beispielsweise in der Mechanik. Inkrementalgeber-Sensoren können beispielsweise die Position der Motormechanik abfragen. Für die Simulation der komponentenbasierten Software genügt im Normalfall ein einfaches diskretes Modell der Regelstrecke. Eine exaktere Nachbildung lässt sich mit kontinuierlichen Modellen erreichen, die aber im Allgemeinen eine größere Rechenleistung bei der Simulation erfordern. Bietet die verwendete Simulationstechnik die Möglichkeit, hybride Simulationsmodelle, das heißt sowohl diskrete als auch kontinuierliche Modellteile, gleichzeitig zu simulieren, erhöht dies die Exaktheit des simulierten Verhaltens.

Die Komponenten im Modell der Umgebung bieten außerdem die Möglichkeit Ausfallsituationen und Störungen im technischen Prozess gezielt nachzubilden. Ausfallsituationen und Störungen sind in der Regel sogar einfacher und ungefährlicher zu simulieren als sie bei der Ausführung im realen Echtzeitsystem kontrolliert zu testen.

Die Simulationsmodell-Komponenten der Umgebung sollten dieselben Schnittstelleneigenschaften wie Software-Komponenten aufweisen. Im Simulationsmodell werden sie an Stelle der realen Schnittstellen des Automatisierungsrechners mit den Simulationsmodell-Komponenten der Software verbunden. Die Komponenten der Umgebung werden bei der Simulation von der Simulationsablaufsteuerung (in Abbildung 6.3 nicht dargestellt) periodisch oder ereignisgesteuert zur Ausführung gebracht. Die erforderlichen Ausführungshäufigkeiten ergeben sich aus den dynamischen Eigenschaften des technischen Prozesses bzw. des Benutzers.

6.1.4 Simulationsmodell eines verteilten Echtzeitsystems

Besteht das Echtzeitsystem aus mehreren, verteilten Automatisierungsrechnern, so muss das Simulationsmodell die verteilte Struktur der komponentenbasierten Software und das verbindende Bussystem nachbilden. Damit das Verhalten realitätsnah nachgebildet wird, muss im Simulationsmodell für jeden Automatisierungsrechner sowohl die Anwendungssoftware als auch die Systemsoftware simuliert werden. Abbildung 6.4 zeigt das Modell eines verteilten Echtzeitsystems mit zwei verteilten Automatisierungsrechnern und dem verbindenden Bussystem.

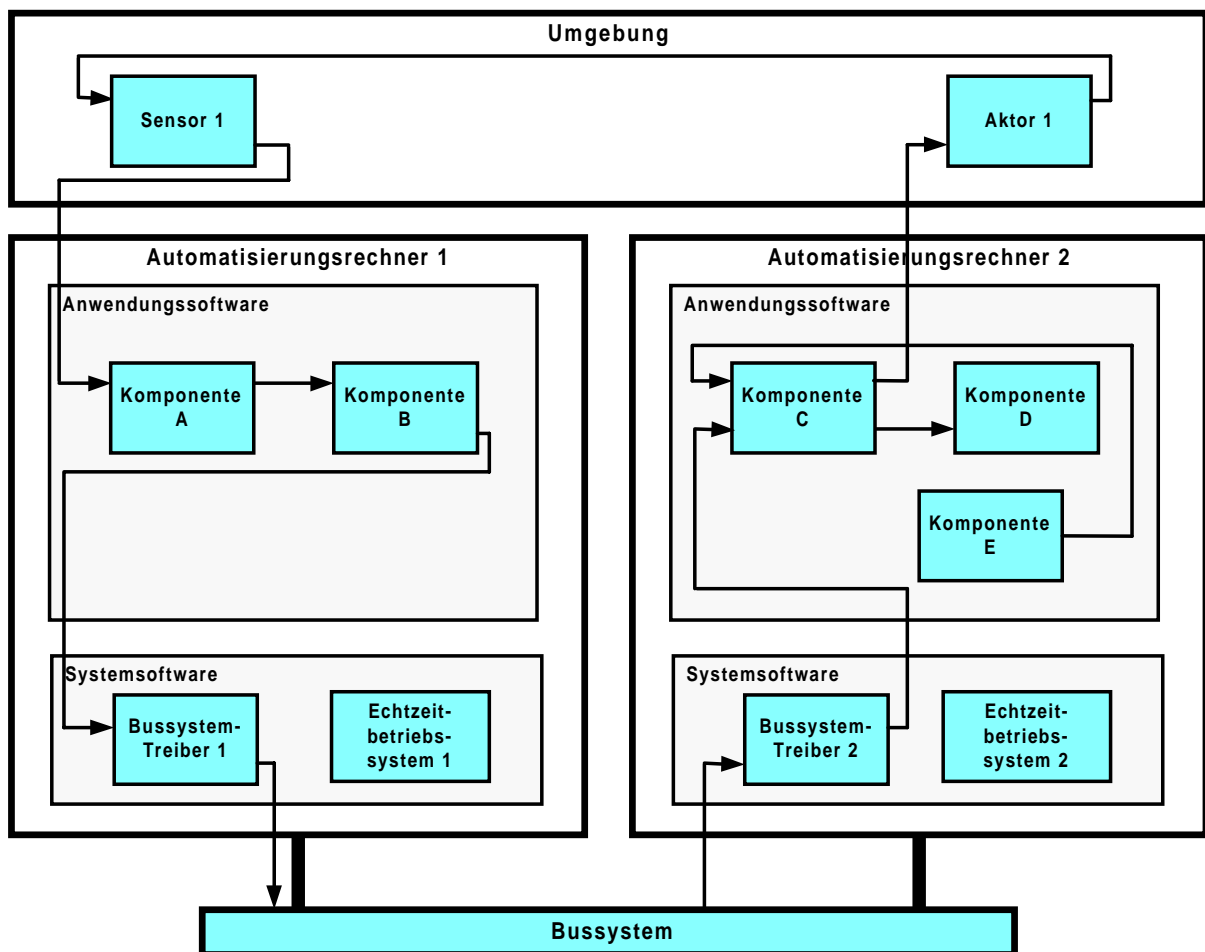


Abbildung 6.4: Simulationsmodell eines verteilten Echtzeitsystems

6.1.5 Simulationsablaufsteuerung

Die Simulationsablaufsteuerung steuert die Simulationsausführung. Um ein möglichst realitätsnahes Echtzeitverhalten zu simulieren, sollte die Simulationsausführung entweder als Echtzeitsimulation oder im echtzeitproportionalen Zeitrafferbetrieb erfolgen. Echtzeitsimulation bedeutet, dass die Simulationszeit im Simulationsmodell synchron mit der realen Uhrzeit voranschreitet. Bei echtzeitproportionalem Zeitrafferbetrieb läuft die Simulationszeit und damit auch alle Aktionen im Simulationsmodell um einen Proportionalitätsfaktor schneller oder langsamer als die reale Uhrzeit.

6.1.6 Generierung von Stimulations-Daten

Während der Simulationsausführung müssen zur gezielten Simulation ausgewählter, prüfbarer Anwendungsfälle Stimulations-Daten zeitgesteuert generiert und an den Eingängen von Komponenten angelegt werden. Die Stimulations-Daten werden zu einem relativ festgelegten Simulationszeitpunkt in Form von Nachrichten an die Eingänge von Schnittstellen gesendet. Zur Simulation verwendete Simulationswerkzeuge müssen dazu eine zeitgesteuerte Generierung von Stimulations-Daten unterstützen.

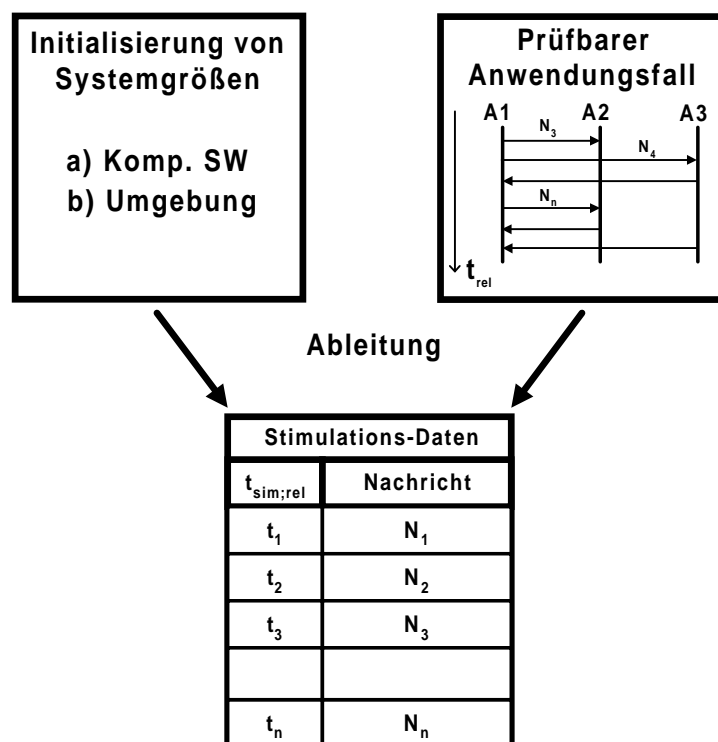


Abbildung 6.5: Ableitung von Stimulations-Daten

Die Stimations-Daten (siehe auch Abschnitt 7.2.2: Aufbau einer Stimuli-Sequenz) bringen das Simulationsmodell in der Initialisierungsphase in einen definierten Ausgangszustand, indem zu Beginn der Simulation die Systemgrößen des technischen Prozesses im Modell der Umgebung eingestellt werden. Zur Validierung eines spezifizierten Anwendungsfalls müssen außerdem die anstoßenden Nachrichten aus dem Sequenzdiagramm des prüfbar Anwendungsfalls in die Sequenz der Stimations-Daten übernommen werden. Anstoßende Nachrichten sind z.B. ein simulierter Bedieneingriff oder eine Ausfallsituation eines Sensors im technischen Prozess. Abbildung 6.5 zeigt die Ableitung von Stimations-Daten. Die tabellarischen Stimations-Daten können prinzipiell auch als Stimuli-Sequenz in Form eines Sequenzdiagramms dargestellt werden.

Die Systemgrößen in Komponenten, die bei der Simulation durch Stimations-Daten verändert werden sollen, müssen bei der Modellierung als Eingänge an die Komponenten-Schnittstelle herausgeführt werden. Abbildung 6.6 zeigt die Generierung von Stimations-Daten im Simulationsmodell.

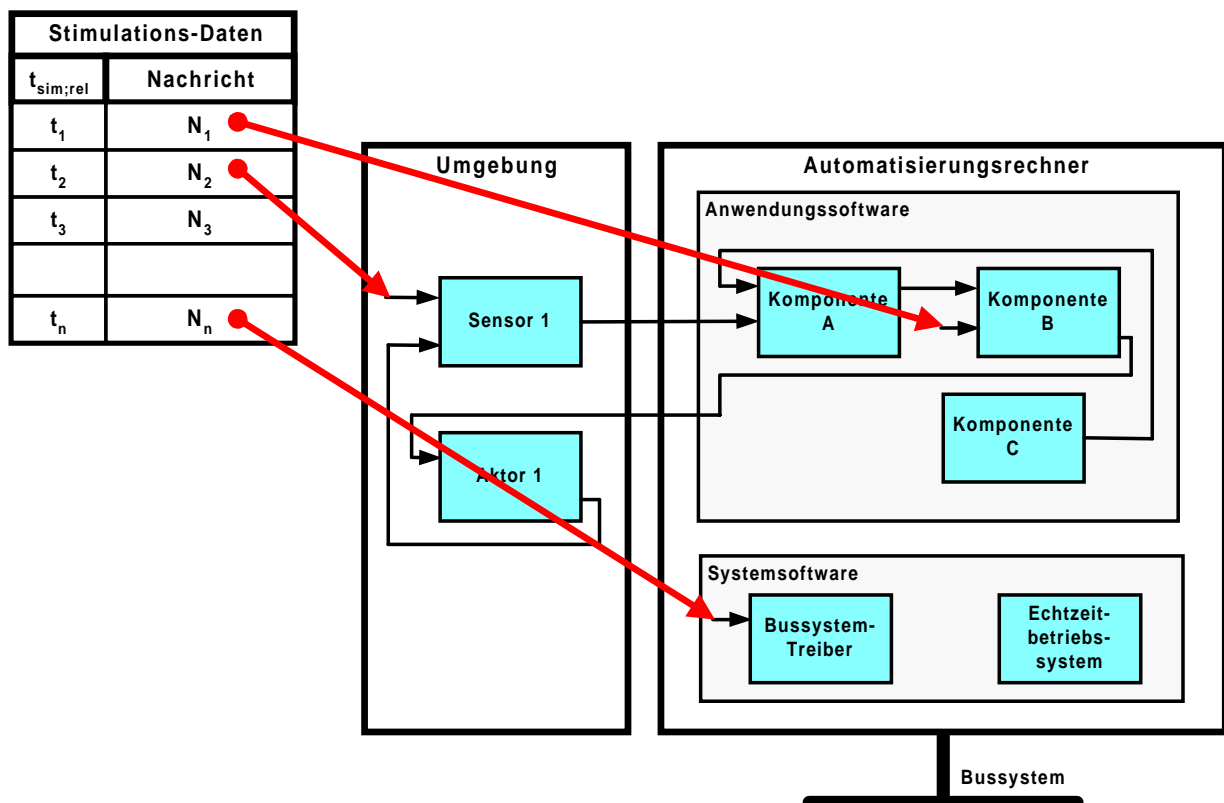


Abbildung 6.6: Generierung von Stimations-Daten

6.1.7 Beobachtung und Aufzeichnung von Nachrichten im Simulationsmodell

Neben der komponentenbasierten Software werden auch die übrigen Bestandteile des Simulationsmodells mit Komponenten modelliert. Deshalb kann bei der Simulation das Black-Box-Verhalten aller Simulationsmodell-Komponenten anhand der an den Schnittstellen der Simulationsmodell-Komponenten versendeten Nachrichten beobachtet und aufgezeichnet werden. Die Beobachtbarkeit der Abläufe im Innern von Simulationsmodell-Komponenten kann insbesondere für die Komponenten der Software nicht vorausgesetzt werden. Da jedoch die zu prüfende Verhaltensspezifikation in den Szenarios der Anwendungsfälle ausschließlich Interaktionen auf Systemebene beschreibt, ist die Beobachtung der versendeten Nachrichten an den Ein- und Ausgängen der Komponenten für die Prüfung des dynamischen Verhaltens völlig ausreichend. Ein Simulationswerkzeug muss daher möglichst komfortabel die Instrumentierung des Simulationsmodells an den Ein- und Ausgängen von Komponenten zur Beobachtung und Aufzeichnung von Nachrichten unterstützen.

Bei der Aufzeichnung werden bei jedem Versenden einer Nachricht mit Hilfe der Instrumentierung die aktuelle Simulationszeit und der Nachrichten-Bezeichner als Zeitstempel protokolliert. Abbildung 6.7 zeigt die Aufzeichnung von Nachrichten im Simulationsmodell mit Hilfe von Zeitstempeln.

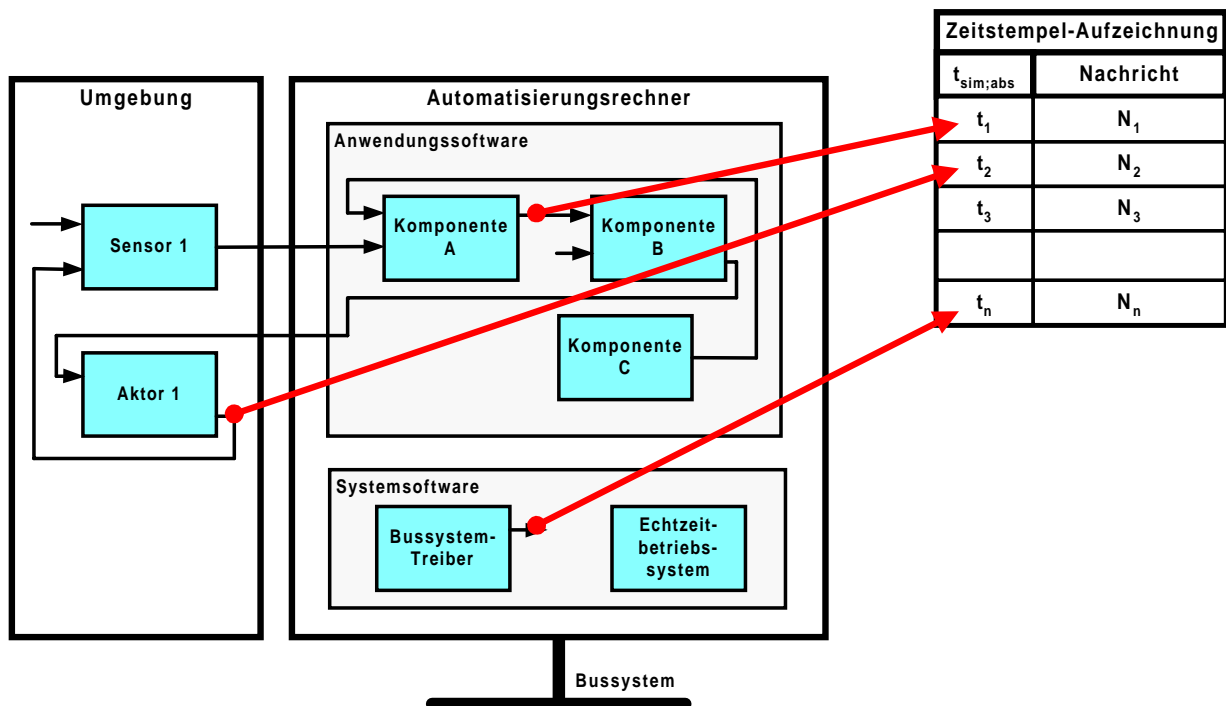


Abbildung 6.7: Zeitstempel-Aufzeichnung von Nachrichten im Simulationsmodell

Die Aufzeichnung der Nachrichten zwischen den Komponenten hat den Vorteil, dass die eigentliche Prüfung des beobachteten Verhaltens erst im Anschluss an die Simulation „offline“ erfolgen kann, da der Entwicklungsrechner im Normalfall durch die rechenzeitintensive Simulationsausführung bereits genügend belastet ist. Der durch die Protokollierung während der Simulationsausführung entstehende Mehraufwand an Rechenzeit ist zwar ebenfalls nicht zu vernachlässigen, aber im Vergleich zu der alternativen Lösung mit Beobachter-Komponenten (engl. observer) wesentlich geringer. Bei der Verwendung von Beobachter-Komponenten wird „online“ nicht nur beobachtet, sondern auch die Bewertung des beobachteten Verhaltens durchgeführt.

6.2 Auswahl eines geeigneten Simulationswerkzeugs

Oberstes Ziel bei der Auswahl eines geeigneten Simulationswerkzeugs ist es, den Erstellungsaufwand und Änderungsaufwand für das Simulationsmodell zu minimieren, bei gleichzeitig realitätsnaher Nachbildung des Verhaltens. Um dieses Ziel zu erreichen, wird deshalb die Auswahl auf kombinierte Entwurfs- und Simulationswerkzeuge zur komponentenbasierten Softwareentwicklung für Echtzeitsysteme eingeschränkt, welche eine automatische Generierung der Simulationsmodell-Komponenten für die komponentenbasierte Software unterstützen. Da für die verschiedenen Anwendungsbereiche von Echtzeitsystemen eine Vielzahl von Entwurfs- und Simulationswerkzeugen existiert, wird die Auswahl auf die Softwareentwicklung für Mikrocontroller-basierte Steuergeräte eingeschränkt. Aus diesem Grund werden nur Werkzeuge betrachtet, die neben dem Entwurf und der Simulation auch eine automatische C-Codegenerierung für Mikrocontroller-basierte Steuergeräte unterstützen. Unter diesen Voraussetzungen kamen nur die zwei kommerziellen Werkzeuge ASCET-SD und ObjecTime Developer in die engere Auswahl. Diese wurden anhand von Beispielprojekten evaluiert [Erns98, Lind00]. Das Ergebnis der Evaluierungen ist in Tabelle 6.1 zu sehen.

Man erkennt, dass beide Werkzeuge die Anforderungen bezüglich der Modellierung überwiegend erfüllen. Beide Werkzeuge unterstützen auch eine Simulation auf dem Entwicklungsrechner, jedoch lediglich ASCET-SD verfügt über eine realitätsnahe Echtzeitsimulation. Dafür ist aber ein spezielles, relativ kostenintensives VME-Bus-basiertes Simulationssystem erforderlich. ObjecTime Developer hingegen bietet lediglich eine Unterstützung zur Instrumentierung des generierten C-Codes zum Testen der Implementierung auf dem Steuergerät. Auf Grund der besseren Simulationseigenschaften wurde für die Erstellung des Simulationsmodells ASCET-SD ausgewählt.

Ein Schwachpunkt beider Werkzeuge ist die mangelnde Unterstützung zur Simulation des Echtzeitverhaltens verteilter Steuergeräte im Verbund. Die Werkzeuge können dabei nicht mehrere Instanzen der Systemsoftware gleichzeitig simulieren. Speziell die

Ausführungssteuerungs-Komponente⁵ ist jedoch maßgeblich für das zeitliche Verhalten der simulierten komponentenbasierten Software verantwortlich. Ein Lösungskonzept, wie mit ASCET-SD entworfene komponentenbasierte Software auch für verteilte Echtzeitsysteme realitätsnah simuliert werden kann, wird in Abschnitt 6.4 gezeigt.

Tabelle 6.1: Bewertung der Entwurfs-, Simulations- und Codegenerierungswerkzeuge ASCET-SD und ObjecTime Developer

		ObjecTime Developer	ASCET-SD
Modellierung und Generierung	Entwurf komponentenbasierter Software	●	●
	Automatische Generierung der Simulationsmodell-Komponenten für komponentenbasierte Software	●	●
	Automatisierte Generierung von C-Code für Steuergeräte	●	●
	Vorgefertigte Simulationsmodell-Komponenten für Systemsoftware	●	●
	Modellierung verteilter Steuergeräte	◐	◐
	Modellierung von Simulationsmodell-Komponenten für Umgebung	◐	●
Simulation	Echtzeitsimulation	○	●
	Echtzeitproportionaler Zeitrafferbetrieb	○	○
	Verteilte Steuergeräte	○	○
	Komfortable Generierung von Simulations-Daten	◐	◐
	Komfortable Beobachtung und Aufzeichnung von Nachrichten an Komponenten-Schnittstellen	●	●

● erfüllt	◐ teilweise erfüllt	○ nicht erfüllt
-----------	---------------------	-----------------

⁵ bei ASCET-SD das OSEK/VDX-basierte Echtzeitbetriebssystem und bei ObjecTime Developer die „ROOM Virtual Machine“

6.3 Erstellung eines Simulationsmodells für einzelne Steuergeräte mit dem Werkzeug ASCET-SD

Das Werkzeug ASCET-SD unterstützt die automatische Generierung von ausführbaren Simulationsmodell-Komponenten aus einem komponentenbasierten Entwurfsmodell. Beim komponentenbasierten Entwurf wird zunächst nur die komponentenbasierte Software mit ASCET-SD modelliert. Zur Vervollständigung des Simulationsmodells müssen das Modell der Umgebung und die Stimulations-Daten ergänzt werden. Da der im Werkzeug enthaltene ASCET-SD Daten-Generator das Anlegen von Stimulations-Daten für mehrere sequenziell auszuführende Szenarios von Anwendungsfällen nicht komfortabel unterstützt, wurde eine Generator-Komponente für Stimulations-Daten entwickelt. Die Stimulations-Daten-Generator-Komponente kann für die Simulation mit mehreren Stimuli-Sequenzen und der gewünschten Ausführungsreihenfolge instanziiert werden. Dadurch kann in einem Simulationslauf das Verhalten mehrerer prüfbarer Anwendungsfälle, von denen die Stimuli-Sequenzen abgeleitet wurden, simuliert und anschließend geprüft werden. Abbildung 6.8 zeigt den Aufbau des Simulationsmodells für ein einzelnes Steuergerät mit ASCET-SD.

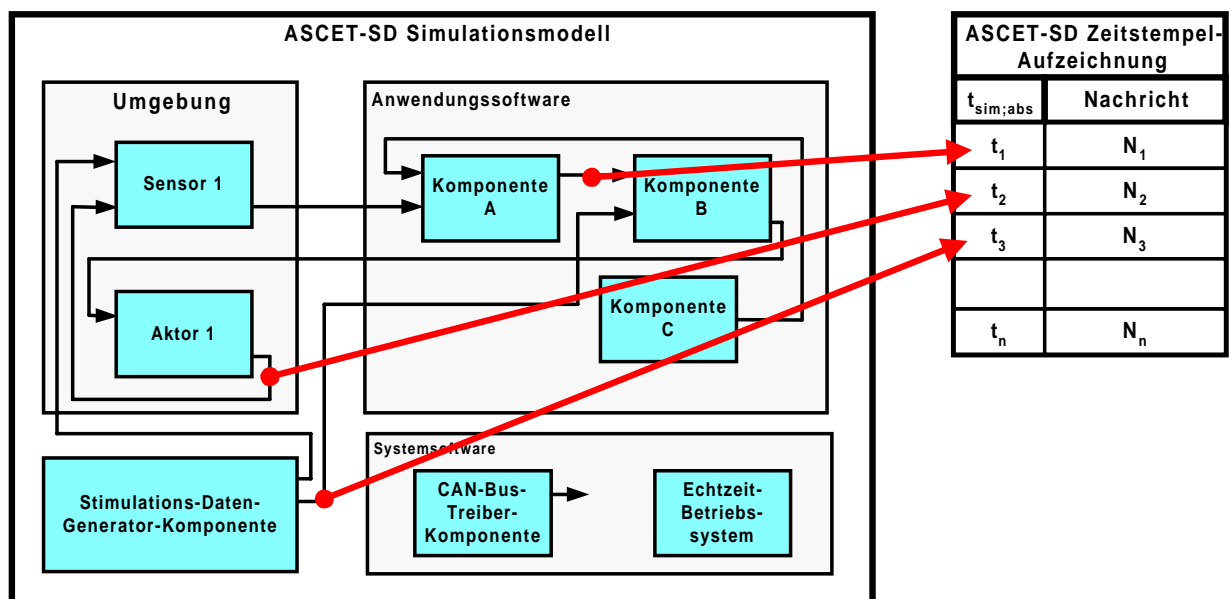


Abbildung 6.8: Aufbau eines Simulationsmodells für ein einzelnes Steuergerät mit ASCET-SD

Die Simulationsablaufsteuerung wird in ASCET-SD Simulationsmodellen durch die Echtzeitbetriebssystem-Komponente unterstützt. Damit bei der Echtzeitsimulation das zeitliche Verhalten der komponentenbasierten Software nicht gestört wird, werden die Komponenten der Umgebung mit niedrigerer Priorität durch das Echtzeitbetriebssystem ausgeführt. Das zur Echtzeitsimulation vorgesehene VME-Bus-basierte Simulationssystem verfügt über eine ausreichende Performance, so dass die Komponenten der Umgebung in den Ausführungspausen der höher

priorisierten Software-Komponenten periodisch ausgeführt werden können. Da das Werkzeug ASCET-SD speziell die Entwicklung komponentenbasierter Software für Steuergeräte im Kfz-Bereich unterstützt, wird für die Echtzeitsimulation bei der Systemsoftware neben dem OSEK/VDX-basierten Echtzeitbetriebssystem [OSEK00] das im Kfz ebenfalls stark verbreitete „Controller Area Network“ (CAN-Bus) [Ets94] standardmäßig durch eine CAN-Bus-Treiber-Komponente unterstützt.

Die für die Beobachtung und Aufzeichnung von Nachrichten notwendige Instrumentierung zur Zeitstempel-Aufzeichnung wird von ASCET-SD automatisch in das Simulationsmodell hinein generiert. Die Zeitstempel-Aufzeichnungen können nach der Simulation im ASCII-Dateiformat für die anschließende Prüfung gespeichert werden.

6.4 Erstellung eines Simulationsmodells für verteilte Steuergeräte mit den Werkzeugen ASCET-SD und CANoe

6.4.1 Übersicht über das Simulationswerkzeug CANoe

Für die Simulation einzelner Steuergeräte bietet das Werkzeug ASCET-SD [ETAS00] insgesamt eine komfortable Unterstützung. Die Simulation verteilter Steuergeräte wird jedoch von ASCET-SD nicht zufriedenstellend unterstützt. Deshalb wurde ein Konzept zur Simulationsmodellerstellung für verteilte Steuergeräte auf Basis des Simulationswerkzeugs CANoe in Kombination mit ASCET-SD Entwurfsmodellen entwickelt [Romm01], welches nachfolgend vorgestellt wird. Da von CANoe der Entwurf von komponentenbasierter Software nicht direkt unterstützt wird, wurde es in der Auswahl in Tabelle 6.1 nicht berücksichtigt.

Das Simulationswerkzeug CANoe [Vect00] unterstützt speziell die Echtzeitsimulation verteilter CAN-Bus-vernetzter Steuergeräte und wird im selben Anwendungsbereich wie ASCET-SD bei der Entwicklung von Software für Kfz-Steuergeräte eingesetzt. Die dem Simulationswerkzeug eigene Zielsetzung der Simulation liegt in der Modellierung und Validierung der Nachrichten-Kommunikation über den CAN-Bus. Die CANoe-Echtzeitsimulation kann außerdem zur Restbussimulation als Unterstützung bei der Systemintegration von verteilten CAN-Bus-vernetzten Steuergeräten verwendet werden. Bei der Restbussimulation werden sukzessive simulierte Steuergeräte durch die realen Steuergeräte ersetzt und damit in das Gesamtsystem integriert.

Um eine möglichst realitätsnahe Simulation der CAN-Bus-Nachrichten-Kommunikation durchführen zu können, unterstützt CANoe auch die Simulation von Automatisierungssoftware. Das Verhalten der Automatisierungssoftware für die verteilten Steuergeräte kann dazu mit der herstellerspezifischen Spezifikationssprache „CAN Application Programming Language“ (CAPL) modelliert werden. CAPL ist jedoch eine sehr einfache, prozedurale Spezifikations-

sprache, die sich nicht zur Modellierung von Strukturen komponentenbasierter Software eignet. Es besteht jedoch in CANoe die Möglichkeit komponentenbasierte Anwendungssoftware als C-Code zusammen mit einem konfigurierbaren OSEK/VDX-Echtzeitbetriebssystem direkt in das Simulationsmodell einzubinden. Die Einbindung eines, z.B. von ASCET-SD, generierten Simulations-C-Codes zur Verhaltenssimulation eines gesamten Steuergeräts in das Simulationsmodell erfolgt in Form einer „Windows Dynamic Link Library“ (Windows-DLL). Eine solche Windows-DLL kapselt bei der Simulation die gesamte komponentenbasierte Software eines Steuergeräts.

6.4.2 Vorgehensweise zur Erstellung eines Simulationsmodells mit ASCET-SD und CANoe

Die Struktur eines Simulationsmodells für verteilte Steuergeräte und die Verbindung zum CAN-Bus werden direkt in CANoe modelliert. Für jedes Steuergerät wird dann je eine Windows-DLL für die Verhaltenssimulation in CANoe erstellt. Abbildung 6.9 zeigt die Vorgehensweise zur Erstellung einer solchen Windows-DLL für die komponentenbasierte Software eines Steuergeräts.

Bei der Erstellung der Windows-DLLs wird ein möglichst hoher Automatisierungsgrad angestrebt. Deshalb erfolgt die Modellierung der verteilten, komponentenbasierten Anwendungssoftware, wie beim Entwurf eines einzelnen Steuergeräts, weiterhin mit dem Werkzeug ASCET-SD. Aus dem komponentenbasierten Entwurfsmodell der Software jedes einzelnen Steuergeräts wird anschließend mit ASCET-SD ein Simulationsmodell für die Echtzeitsimulation auf dem VME-Bus-basierten Simulationssystem generiert. Der dabei von ASCET-SD generierte Simulations-C-Code für die Komponenten der Anwendungssoftware ist an den Schnittstellen kompatibel zu OSEK/VDX-basierten [OSEK00] Echtzeitbetriebssystemen. Der generierte Simulations-C-Code der komponentenbasierten Anwendungssoftware kann deshalb in einem MS-Visual-C-Projekt zusammen mit einer CANoe-Simulationsmodell-Komponente für ein OSEK/VDX-Echtzeitbetriebssystem zur PC-basierten Echtzeitsimulation sehr einfach integriert werden. Die Konfigurierung der Komponente OSEK/VDX-Echtzeitbetriebssystem wird dabei mit Hilfe des CANoe-OIL-Konfigurators durchgeführt. Aus den Konfigurationsdaten wird automatisch C-Code für die CANoe-Simulationsmodell-Komponente des OSEK/VDX-Echtzeitbetriebssystems generiert. Weiterhin werden aus der zuvor für den verteilten Steuergeräteverbund definierten CAN-Bus-Nachrichten-Spezifikation die Konfigurationsdaten für die CAN-Bus-Treiber-Komponenten der einzelnen Steuergeräte ebenfalls automatisch generiert. Der dabei generierte C-Code wird in das MS-Visual-C-Projekt integriert. Aus dem gesamten MS-Visual-C-Projekt wird abschließend eine Windows-DLL kompiliert, die als ein Steuergerät in das CANoe Simulationsmodell eingebunden wird.

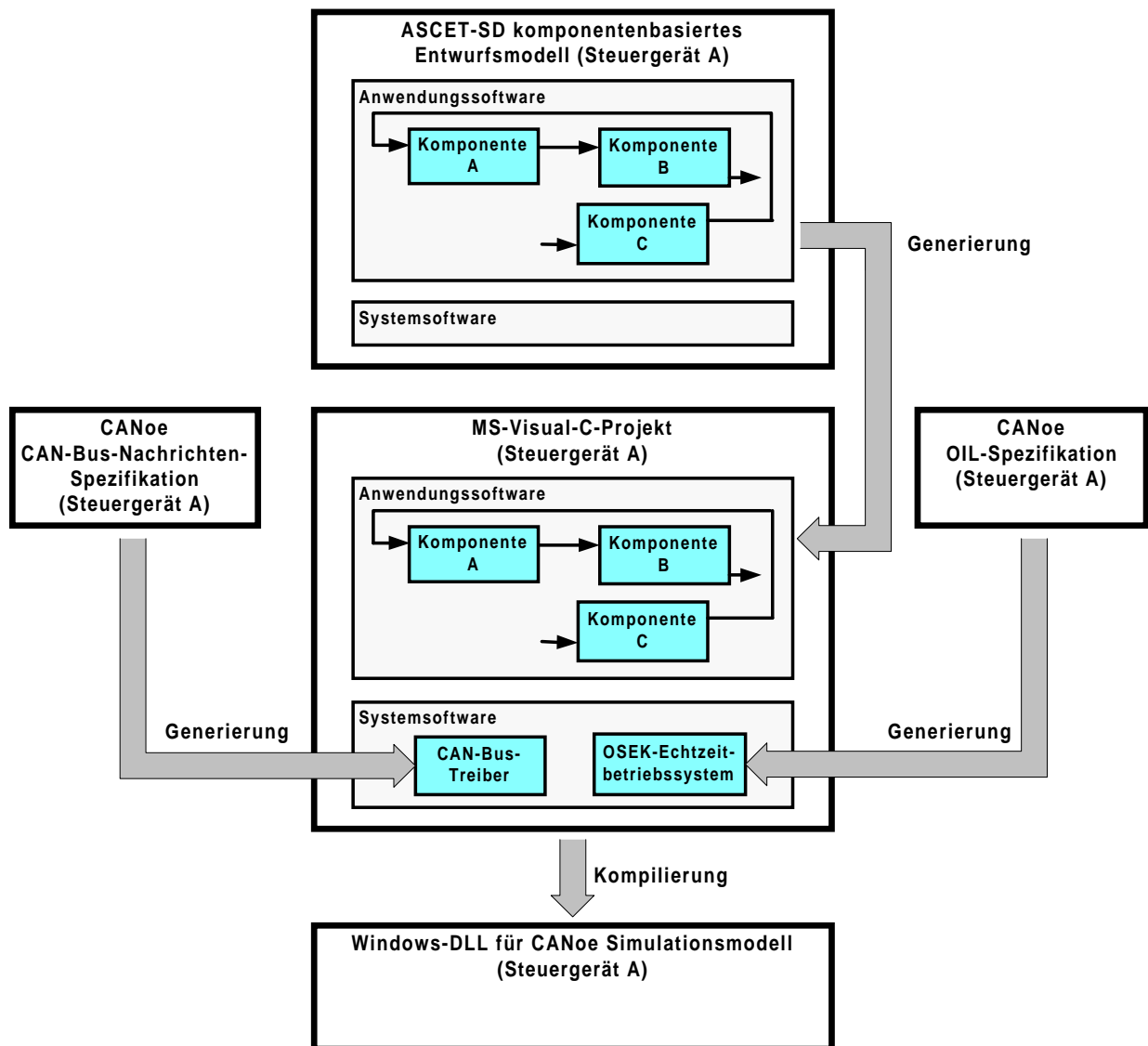


Abbildung 6.9: Erstellung der Windows-DLL zur Simulation der komponentenbasierten Software eines Steuergeräts in CANoe

6.4.3 Aufbau eines CANoe-Simulationsmodells

Die Simulationsausführung erfolgt mit CANoe auf einem Entwicklungsrechner als Echtzeit-simulation für mehrere verteilte Steuergeräte. Abbildung 6.10 zeigt den Aufbau des Simulationsmodells für verteilte Steuergeräte mit CANoe.

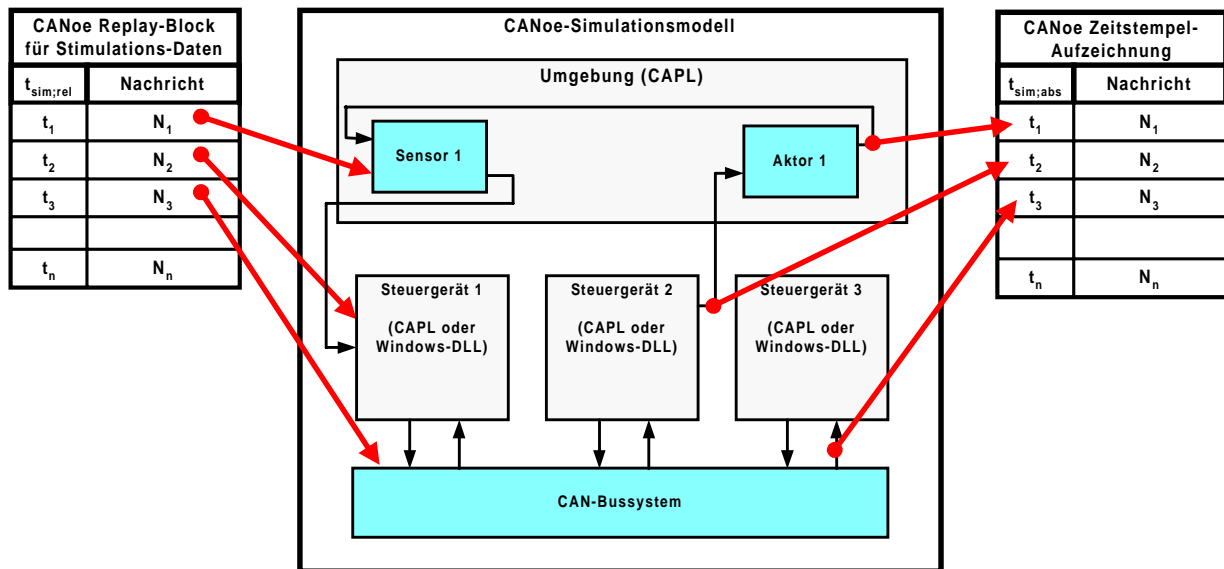


Abbildung 6.10: Aufbau eines Simulationsmodells für verteilte Steuergeräte mit CANoe

Jedes Steuergerät wird für die Simulation als Windows-DLL in das Simulationsmodell eingebunden und besitzt damit eine eigene Instanz des OSEK/VDX-Echtzeitbetriebssystems. Die nebenläufige Simulation in Echtzeit setzt aber schon für wenige simulierte Steuergeräte eine große Rechenleistung des Entwicklungsrechners voraus.

Die Simulation des CAN-Bussystems wird wahlweise mit einer Simulationsmodell-Komponente oder bei der Echtzeitsimulation mit dem real verbundenen CAN-Bus realisiert. Durch eine spezielle PC-Einschubkarte wird der Zugang vom Entwicklungsrechner zum realen CAN-Bus geschaffen.

Das Verhalten einfacherer Modelle der Umgebung wird mit der Spezifikationsprache CAPL modelliert. Bei komplexeren Simulationsmodell-Komponenten der Umgebung können ebenfalls Windows-DLLs in das Simulationsmodell eingebunden werden.

Die Stimulations-Daten werden im CANoe-Simulationsmodell über den „Replay-Block“ während der Simulationsausführung zeitgesteuert an herausgeführte Schnittstellen der einzelnen Steuergeräte versendet.

Die Beobachtung und Aufzeichnung von Nachrichten auf dem CAN-Bus und an den Schnittstellen der Simulationmodell-Komponenten wird mit Hilfe der CANoe-Zeitstempel-Aufzeichnung durchgeführt. Diese hat jedoch keine Beobachtungsmöglichkeiten innerhalb der Windows-DLLs. Aus diesem Grund müssen Schnittstellen zwischen den Komponenten innerhalb der Windows-DLLs, die beobachtet werden sollen, vorab manuell instrumentiert werden. Sollen die Eingänge von Schnittstellen innerhalb einer Windows-DLL stimuliert werden, so müssen sie explizit an die Schnittstelle der Windows-DLL herausgeführt werden. Die Zeitstempel-Aufzeichnungen können während der Simulation im ASCII-Dateiformat für die anschließende Prüfung protokolliert werden.

Als Fazit der Konzept-Evaluierung wurde festgestellt, dass durch die Verbindung der Werkzeuge ASCET-SD und CANoe bei der Erstellung des Simulationsmodells auch die Simulation von realitätsnahem Echtzeitverhalten für verteilte Automatisierungsrechner mit vertretbarem Aufwand machbar ist. Eine Umsetzung des entwickelten Konzepts in die beiden kommerziellen Simulationswerkzeuge durch die Werkzeughersteller könnte den Aufwand für die Modellerstellung für die Bestandteile der komponentenbasierten Software für verteilte Echtzeitsysteme, vergleichbar zur Modellerstellung für einzelne Steuergeräte mit ASCET-SD, ebenfalls auf einen Knopfdruck reduzieren.

Die in diesem Kapitel aufgestellten Anforderungen zeigen, dass für eine Prüfung des dynamischen Verhaltens von komponentenbasierter Software die Realität des gesamten Echtzeitsystems nachgebildet werden muss. Dazu ist ein Simulationsmodell notwendig, welches neben der komponentenbasierten Software auch alle anderen wesentlichen Bestandteile des Echtzeitsystems, wie z.B. den technischen Prozess, in einem Modell der Umgebung für die Simulation nachbildet. Bei der Auswahl eines geeigneten Simulationswerkzeugs kamen nur kommerzielle, kombinierte Entwurfs- und Simulationswerkzeuge in die engere Wahl, um den Erstellungs- und Änderungsaufwand für Simulationsmodelle in akzeptablen Grenzen zu halten. Dem Werkzeug ASCET-SD wurde wegen seiner Möglichkeiten zur Echtzeitsimulation dabei der Vorzug gegeben. Um die fehlende Unterstützung von ASCET-SD zur realitätsnahen Simulation verteilter Steuergeräte aufzuheben, wurde ein Lösungskonzept zur Echtzeitsimulation von verteilten, komponentenbasierten ASCET-SD-Entwurfsmodellen mit dem Simulationswerkzeug CANoe entwickelt. Mit den präsentierten Erstellungskonzepten können Simulationsmodelle sowohl für einzelne Steuergeräte als auch für CAN-Bus-vernetzte, verteilte Steuergeräte erstellt werden. Im nächsten Kapitel wird gezeigt, wie mit Hilfe eines derartigen Simulationsmodells die simulierte komponentenbasierte Software gegenüber den systematisch aus den Anwendungsfällen hergeleiteten Prüffällen validiert wird.

7 Prüfung des simulierten Verhaltens der komponentenbasierten Software

In diesem Kapitel werden die Prüfung und Bewertung des simulierten Verhaltens der komponentenbasierten Software detaillierter behandelt. Dazu wird eine Vorgehensweise zur systematischen Entwicklung von Prüffällen aus den Anwendungsszenarios der prüfbareren Anwendungsfällen festgelegt und anschließend die Strategie zur Simulationsausführung der Prüffälle entwickelt. Des Weiteren werden Kriterien für die Konformitätsprüfung zwischen dem in der Simulation aufgezeichneten Verhalten und dem in den Anwendungsszenarios geforderten Verhalten definiert und Regeln für die Ermittlung der Prüfergebnisse festgelegt. Die Prüfergebnisse liefern stets eine eindeutige Bewertung, ob das simulierte Verhalten validiert oder fehlerhaft ist. Abschließend sind einige Erfahrungswerte zur Aufdeckung und Lokalisierung von Entwurfsfehlern zusammengestellt.

7.1 Übersicht über das Prüfverfahren zur Validierung

Das Ziel des Prüfverfahrens ist die frühzeitige Validierung des Verhaltens von komponentenbasierter Software für Echtzeitsysteme. Abbildung 7.1 zeigt die einzelnen Schritte des Prüfverfahrens zur Validierung des simulierten Verhaltens eines komponentenbasierten Entwurfsmodells.

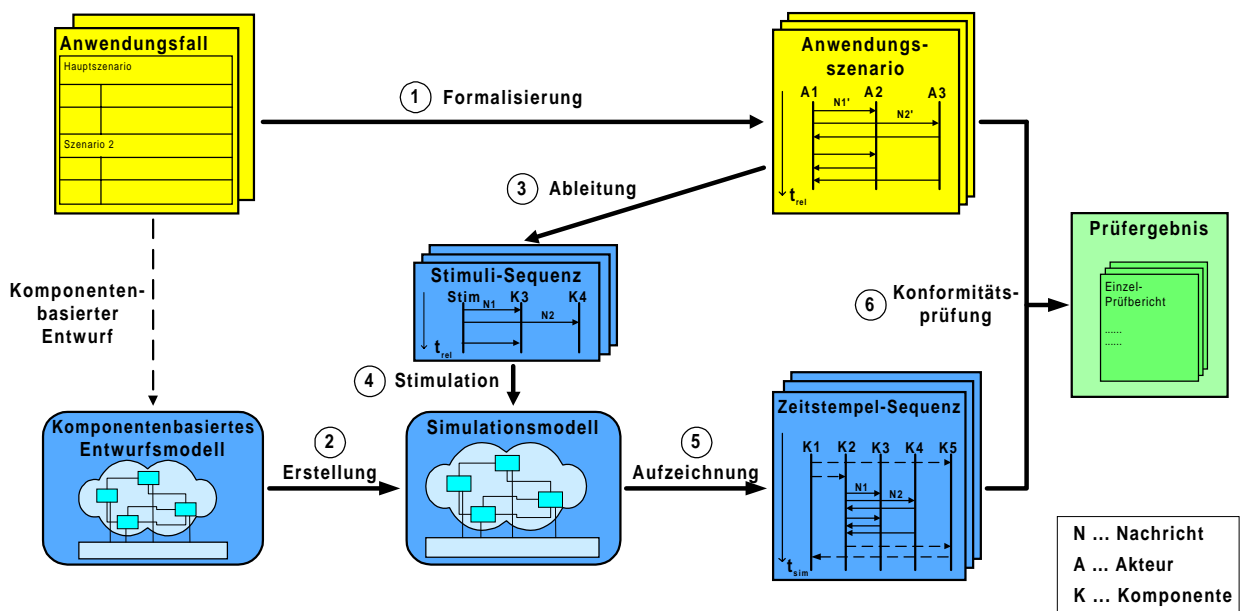


Abbildung 7.1: Schritte des Prüfverfahrens zur Validierung

Die Ausgangspunkte für das Prüfverfahren sind die Anwendungsfälle und das komponentenba-

sierte Entwurfsmodell der Software. In Kapitel 5 wurde gezeigt, wie informelle Anwendungsfälle mit Hilfe der formalen Notation der erweiterten UML Sequenzdiagramme in prüfbare Anwendungsfälle überführt werden. Jedes bei der Überführung erstellte Sequenzdiagramm ergibt ein Anwendungsszenario. Die prüfbaren Anwendungsfälle bestehen daher nach der Formalisierung im 1. Schritt aus einer Menge von Anwendungsszenarios und bilden das Referenz-Verhalten für die abschließende Konformitätsprüfung.

Damit die Validierung der komponentenbasierten Software bereits frühzeitig während des komponentenbasierten Entwurfs erfolgen kann, wird die Prüfung anhand des komponentenbasierten Entwurfsmodells vorgenommen. Das komponentenbasierte Entwurfsmodell der Software wird dazu im 2. Schritt in ein Simulationsmodell überführt, welches das wesentliche Verhalten des gesamten Echtzeitsystems realitätsnah nachbildet. In Kapitel 6 wurde die Erstellung dieses Simulationsmodells detailliert erläutert.

Für die Prüfung des simulierten Verhaltens der komponentenbasierten Software müssen im 3. Schritt des Prüfverfahrens Prüffälle entwickelt werden. Die Prüffälle werden dabei spezifikationsbasiert aus den Anwendungsszenarios abgeleitet. Für jedes Anwendungsszenario wird mindestens eine Stimuli-Sequenz abgeleitet. Die Stimuli-Sequenz ist eine Sequenz von Nachrichten, welche als Stimulations-Daten bei der Simulationsausführung an das Simulationsmodell angelegt werden. Jede Stimuli-Sequenz bildet zusammen mit dem im zugehörigen Anwendungsszenario beschriebenen Referenz-Verhalten einen Prüffall.

Diese Prüffälle werden bei der Simulation ausgeführt. Dazu werden im 4. Schritt die Nachrichten der Stimuli-Sequenzen zur Stimulation an das Simulationsmodell angelegt und im 5. Schritt die im Simulationsmodell kommunizierten Nachrichten in Form von Zeitstempel-Sequenzen aufgezeichnet. Die aufgezeichneten Zeitstempel-Sequenzen können ebenfalls als chronologische Sequenz von Nachrichten in einem Sequenzdiagramm dargestellt werden. An Stelle der Akteure enthalten die Zeitstempel-Sequenzen jedoch die Komponenten des Simulationsmodells, zwischen denen die aufgezeichneten Nachrichten versendet wurden.

Im 6. Schritt erfolgt die Konformitätsprüfung zwischen den aufgezeichneten Zeitstempel-Sequenzen und dem in den Anwendungsszenarios spezifizierten Referenz-Verhalten. Auf die Schritte 3 bis 6 wird in den nachfolgenden Abschnitten detaillierter eingegangen.

7.2 Entwicklung von Prüffällen durch Ableitung von Stimuli-Sequenzen

7.2.1 Systematik zur Ableitung von Stimuli-Sequenzen

Als grundlegende Systematik zur Prüffallentwicklung wird die Prüfung aller spezifizierten Anwendungsszenarios festgelegt [Flei00b]. Dies bedeutet, dass jedes Anwendungsszenario bei

der Simulation mindestens einmal ausgeführt werden muss. Um die Ausführung aller Anwendungsszenarios im Simulationsmodell gezielt zu stimulieren, muss von jedem Anwendungsszenario mindestens eine Stimuli-Sequenz abgeleitet werden. Zusätzliche Stimuli-Sequenzen werden durch die Variation von Systemvariablen, wie z.B. die mechanische Gegenkraft an einem Elektromotor, im Modell der Umgebung abgeleitet. Damit können die Anwendungsszenarios unter verschiedenen Umgebungsbedingungen simuliert werden.

Alternativ wäre auch eine zufällige⁶ Stimulation des Simulationsmodells denkbar. Bei komplexeren Modellen wären dazu aber sehr lange Simulationszeiten notwendig, um die vollständige Ausführung aller Anwendungsszenarios auf diese Weise zu erreichen. Speziell bei Verwendung von Echtzeitsimulation, wo die Simulationszeit synchron zur realen Uhrzeit voranschreitet, käme die erforderliche Simulationszeit sehr leicht in die Größenordnung von Jahren. Aus diesem Grund ist die zufällige Stimulation zur vollständigen Prüfung aller Anwendungsszenarios keine effiziente Simulationsstrategie und wird deshalb für das in dieser Arbeit entwickelte Prüfverfahren nicht verwendet.

7.2.2 Aufbau einer Stimuli-Sequenz

Für jedes Anwendungsszenario muss mindestens eine Stimuli-Sequenz abgeleitet werden. Auf die Ableitung von Simulations-Daten und die Generierung im Simulationsmodell während der Simulationsausführung wurde bereits in Abschnitt 6.1.6 eingegangen. Eine Stimuli-Sequenz besteht demnach aus einer Sequenz von Nachrichten. In Abbildung 7.2 ist der Aufbau einer Stimuli-Sequenz als Sequenzdiagramm dargestellt.

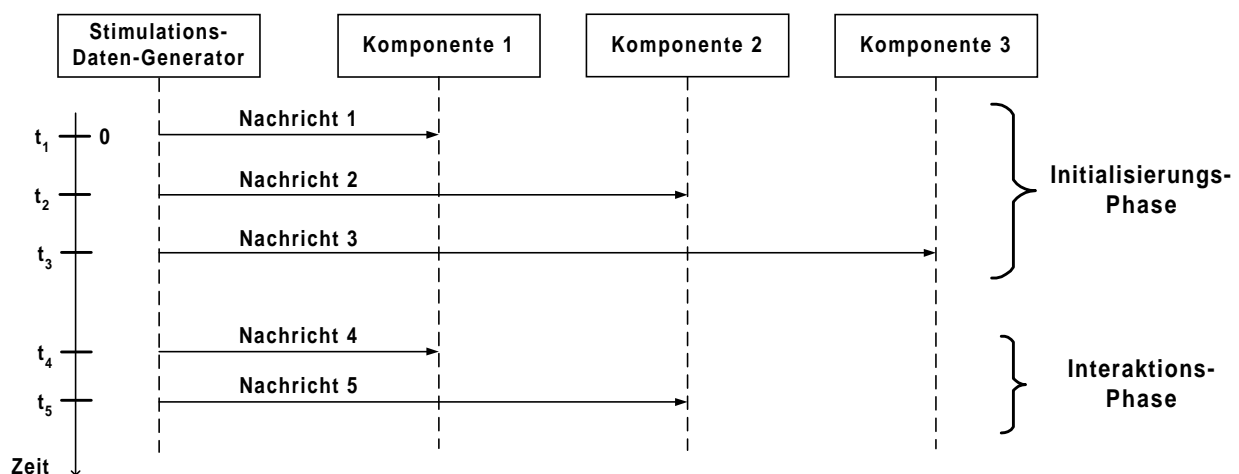


Abbildung 7.2: Aufbau einer Stimuli-Sequenz

Die erste Nachricht einer Stimuli-Sequenz legt den relativen Startzeitpunkt fest. Die nachfolgenden Nachrichten werden relativ zu diesem Startzeitpunkt mit einer definierten zeitlichen

⁶ mit stochastisch verteilten Simulations-Daten

Verzögerung versendet. Alle versendeten Nachrichten in der Stimuli-Sequenz gehen von der Komponente Stimulation-Daten-Generator aus und können an beliebige Eingänge von Komponenten im Simulationsmodell versendet werden. Eine Stimuli-Sequenz ist in zwei aufeinanderfolgende Phasen unterteilt:

1. **Initialisierungs-Phase:** Diese Sequenz von Nachrichten stellt den in der Vorbedingung des zugehörigen Anwendungsszenarios spezifizierten Ausgangszustand im Simulationsmodell her. Dabei müssen die initialisierenden Nachrichten die Systemgrößen des technischen Prozesses im Simulationsmodell in den für das Anwendungsszenario gewünschten Ausgangszustand überführen. Die Initialisierungs-Phase kann für viele Anwendungsszenarios gleich sein. In der Praxis tritt dieser Fall ein, wenn die Vorbedingung für verschiedene Anwendungsszenarios gleich ist, z.B. den Default-Wert „Wahr“ hat.
2. **Interaktions-Phase:** Die Interaktions-Phase folgt im Anschluss an die Initialisierungs-Phase. Diese Sequenz von Nachrichten stimuliert gezielt den Ablauf des zugehörigen Anwendungsszenarios. Dazu werden aus dem Anwendungsszenario alle Nachrichten übernommen, die das Anwendungsszenario anstoßen bzw. von externen Akteuren versendet werden und nicht von rückkopplungsbehafteten Streckenmodellkomponenten der Umgebung bei der Simulation erzeugt werden. Als externer Akteur kann z.B. ein Benutzer oder ein Sensor mit nicht geregelten Systemgrößen aus der Umwelt agieren. Auf diese Weise können z.B. Ausfallsituationen, die in den Anwendungsfällen als Ausnahmen spezifiziert werden, gezielt stimuliert werden. Im einfachsten Fall besteht die Interaktions-Phase nur aus der ersten Nachricht eines Anwendungsszenarios oder einer Sequenz von Nachrichten, die das Versenden dieser ersten Nachricht herbeiführt.

Da die Anwendungsszenarios und das Simulationsmodell auf zwei unterschiedlichen Abstraktionsebenen liegen, kann nicht vorausgesetzt werden, dass die Nachrichten in den Anwendungsszenarios und im Simulationsmodell gleich sind. Deshalb muss eine Zuordnung zwischen den verwendeten Nachrichten in den Anwendungsszenarios auf die Nachrichten im Simulationsmodell durchgeführt werden. Auf die Zuordnung der Nachrichten wird im Abschnitt 7.4.2 noch detaillierter eingegangen. Bei der Ableitung der Stimuli-Sequenzen werden die zugeordneten Nachrichten aus dem Simulationsmodell verwendet.

7.3 Simulation der Prüffälle

Nachdem mit der Ableitung der Stimuli-Sequenzen aus allen Anwendungsszenarios die Prüffälle systematisch entwickelt wurden, müssen sie im nächsten Schritt simuliert werden. Wie schon die Ableitung der Stimuli-Sequenzen, so unterliegt auch die Simulationsausführung der Anforderung, möglichst zielgerichtet das Verhalten der Anwendungsszenarios zu simulieren, um es anschließend bei der Konformitätsprüfung bewerten zu können.

Als Strategie zur Simulationsausführung der Menge von Prüffällen stehen prinzipiell zwei Möglichkeiten zur Verfügung:

- Sequenzielle Simulationsausführung
- Überlagerte Simulationsausführung

Bei der sequenziellen Simulationsausführung wird das Simulationsmodell sequenziell mit den Nachrichten der Stimuli-Sequenzen stimuliert, das bedeutet, dass ein Prüffall nach dem anderen ausgeführt wird. Für jeden simulierten Prüffall wird dazu das simulierte Verhalten in einer Zeitstempel-Aufzeichnung für die anschließende Konformitätsprüfung aufgezeichnet.

Die andere Möglichkeit ist die überlagerte Simulationsausführung mehrerer Prüffälle. Bei der überlagerten Simulationsausführung werden mehrere Stimuli-Sequenzen überlagert. Das bedeutet, dass das Simulationsmodell mit den Nachrichten mehrerer Stimuli-Sequenzen nebenläufig stimuliert wird. Die überlagerte Simulationsausführung erzeugt damit eine Stress-Simulation für die komponentenbasierte Software und das gesamte Echtzeitsystem. Die Überlagerung der Stimuli-Sequenzen kann jedoch nur für eine eingeschränkte Auswahl von Stimuli-Sequenzen erfolgen, die aus unterschiedlichen Anwendungsfällen abgeleitet wurden. Bei Stimuli-Sequenzen, die aus Anwendungsszenarios desselben Anwendungsfalls abgeleitet wurden, handelt es sich in der Regel um alternative Abläufe, deren stimulierte Vorbedingung und Interaktionen unterschiedlich sind. Die von den Stimuli-Sequenzen stimulierten Vorbedingungen und versendeten Nachrichten dürfen sich aber keinesfalls widersprechen. Dies gilt auch für die Überlagerung von Stimuli-Sequenzen die aus Anwendungsszenarios verschiedener Anwendungsfälle abgeleitet wurden.

Beim praktischen Einsatz des Prüfverfahrens zur Validierung empfiehlt es sich, zuerst die sequenzielle Simulationsausführung durchzuführen. Erst nachdem das Verhalten der Prüffälle einzeln validiert wurde, macht es Sinn mit einer zusätzlichen nebenläufigen, überlagerten Simulationsausführung auch Lastsituationen mit einer Stress-Simulation zu validieren.

7.4 Konformitätsprüfung

7.4.1 Prüfkriterien zwischen zwei Sequenzdiagrammen

Die Validierung des simulierten Verhaltens der komponentenbasierten Software soll durch eine Konformitätsprüfung gegenüber dem in den Anwendungsfällen geforderten Verhalten erfolgen. Das simulierte Verhalten wird dazu als Menge aller aufgezeichneten Zeitstempel-Sequenzen einer simulierten komponentenbasierten Software definiert. Das geforderte Verhalten wird als Menge aller spezifizierten Anwendungsszenarios festgelegt.

Bei der Konformitätsprüfung stellt sich die prinzipielle Frage, wann das simulierte Verhalten

konform mit dem geforderten Verhalten ist bzw. wann Abweichungen auf Fehler schließen lassen. Da sowohl Anwendungsszenarios als auch Zeitstempel-Sequenzen prinzipiell als Sequenzdiagramme dargestellt werden können, kann die Fragestellung auf die Konformitätsprüfung zwischen zwei Sequenzdiagrammen zurückgeführt werden. Im Folgenden werden dazu Prüfkriterien für die Konformitätsprüfung von zwei Sequenzdiagrammen aufgestellt, die festlegen, wann zwei Sequenzdiagramme „konform“ sind, und umgekehrt, wann sie „nicht konform“ sind.

Die Aufstellung der Prüfkriterien erfolgt in Anlehnung an bestehende Ansätze aus der Literatur. Interessante Ansätze existieren darin für die Konformitätsprüfung von „Message Sequence Charts“ (MSC), deren Notation größtenteils als Sequenzdiagramm in den UML Standard übernommen wurde. Muscholl [Musc99] beispielsweise definiert Prüfkriterien zur Konformitätsprüfung zwischen zwei MSCs und zugehörige Vergleichsalgorithmen. Alur, Holzmann und Peled [AHP96] erweitern den Ansatz von Muscholl um die Spezifikation und Prüfung von Zeitbedingungen zwischen Nachrichten in MSCs. Auf Basis dieser Ansätze werden für die Konformitätsprüfung einer Zeitstempel-Sequenz gegenüber einem Anwendungsszenario folgende Prüfkriterien definiert:

- Es existiert eine vollständige Abbildung aller Nachrichten des Anwendungsszenarios auf die Nachrichten der Zeitstempel-Sequenz. Abbildung 7.3 zeigt exemplarisch diesen Sachverhalt. Die Nachrichten N1', N2' und N3' des Anwendungsszenarios werden dabei vollständig auf die Nachrichten N1, N2 und N3 der Zeitstempel-Sequenz abgebildet. Die Zeitstempel-Sequenz darf, wie ebenfalls in Abbildung 7.3 dargestellt, zusätzlich noch weitere Nachrichten enthalten, die von der Abbildung nicht getroffen werden. Für den Spezialfall, dass eine oder mehrere periodische Teil-Sequenzen im Anwendungsszenario spezifiziert sind, muss jede Nachricht des Anwendungsszenarios mindestens einmal abgebildet werden.

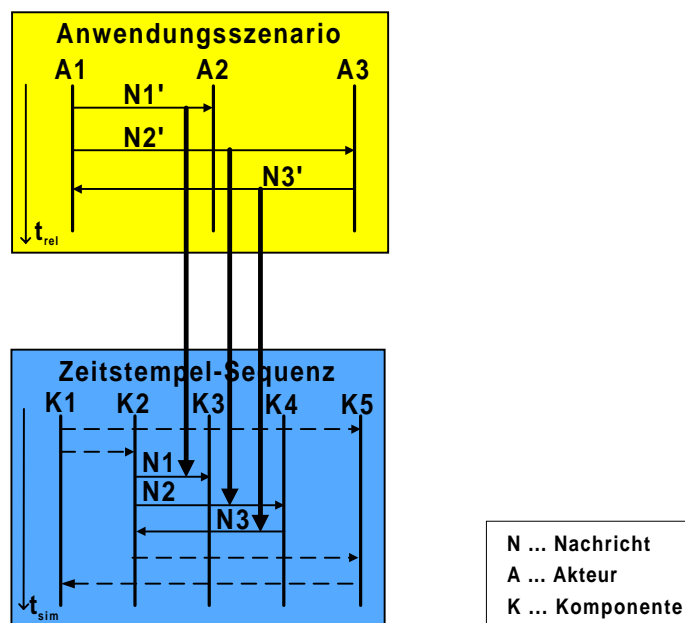


Abbildung 7.3: Vollständige Abbildung der Nachrichten eines Anwendungsszenarios auf die Nachrichten einer Zeitstempel-Sequenz

- Die kausale Ordnung der Nachrichten muss bei der Abbildung erhalten bleiben. Das heißt, dass die Reihenfolge der Nachrichten im Anwendungsszenario mit der Reihenfolge der abgebildeten Nachrichten in der Zeitstempel-Sequenz übereinstimmen muss.
- Die im Anwendungsszenario spezifizierten Zeitbedingungen zwischen Nachrichten müssen zwischen den abgebildeten Nachrichten der Zeitstempel-Sequenz eingehalten werden.

In der Praxis ergab sich bei der Konformitätsprüfung ein spezielles Problem. Bei Software für Echtzeitsysteme werden Nachrichten oftmals, über Zeitgeberdienste getriggert, periodisch versendet. Insbesondere bei verteilten Echtzeitsystemen mit Kommunikation über Bussysteme, wie z.B. dem CAN-Bus, ist dieses Entwurfsmuster häufig anzutreffen. Diese Redundanz wird speziell bei sicherheits- und zuverlässigkeitsrelevanten Anwendungen angewendet. Mit einem erweiterten Sequenzdiagramm kann das mehrfache Versenden einzelner Nachrichten an einer bestimmten Stelle im Anwendungsszenario spezifiziert werden. Auch die periodische Wiederholung von Teil-Sequenzen kann spezifiziert werden. Das Beschreibungsmittel eignet sich jedoch nicht, um das permanente periodische Versenden von Nachrichten zu spezifizieren.

Aus diesem Grund wird für die Konformitätsprüfung ein zusätzliches Attribut Vergleichstyp für die Anwendungsszenarios eingeführt. Der Vergleichstyp kann auf die zwei Werte „Periodische Nachrichten erlaubt“ oder „Periodische Nachrichten nicht erlaubt“ eingestellt werden. Ist der Vergleichstyp auf „Periodische Nachrichten erlaubt“ eingestellt, ergibt sich daraus ein weiteres Prüfkriterium:

- Sind „Periodische Nachrichten erlaubt“, so dürfen gleiche Nachrichten in der Zeitstempel-Sequenz zusätzlich auch vor oder nach der geforderten Stelle in der Sequenz auftreten. Das zu frühe bzw. zu späte Auftreten der geforderten Nachricht wird in diesen Fällen bei der Konformitätsprüfung nicht als Fehler bewertet. Die Nachricht muss aber an der geforderten Stelle in der Sequenz auf jeden Fall auftreten. Abbildung 7.4 zeigt an einem Beispiel die Abbildung der Nachrichten bei einer periodisch auftretenden Nachricht N2. Die Nachricht N2 tritt dabei bereits einmal vor der Nachricht N1 auf und tritt zweimal zwischen den Nachrichten N1 und N3 auf. Die Abbildung von N2' erfolgt dabei auf das erste Auftreten von N2 nach N1.

Ein anderes Praxisproblem, welches in der Literatur ebenfalls nicht behandelt wird, ist das mehrfache Auftreten der gleichen Nachricht in einem Anwendungsszenario. Wird innerhalb eines Anwendungsszenarios eine Nachricht an mehreren Stellen der Sequenz spezifiziert, so ist sie aus Sicht des Prüfalgorithmus nicht dieselbe Nachricht, da sie an einer anderen Stelle in der Sequenz auftritt. Bei der Konformitätsprüfung muss diese Nachricht in der Zeitstempel-Sequenz an allen geforderten Stellen auftreten.

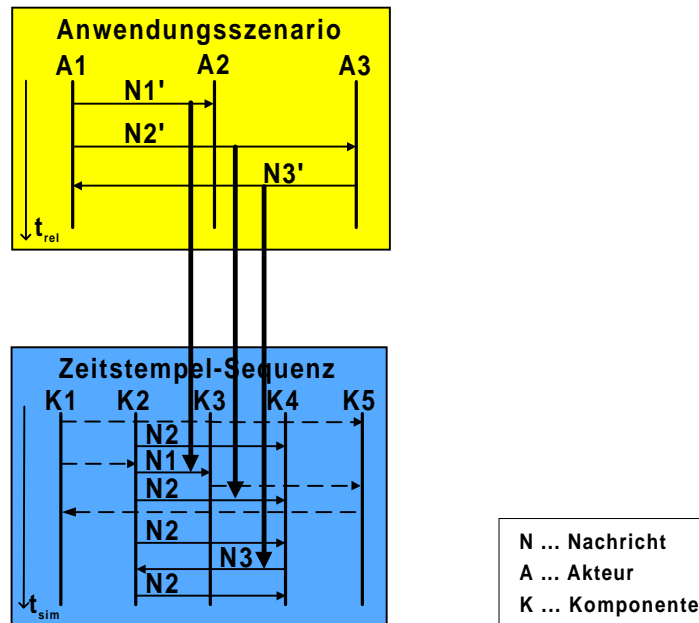


Abbildung 7.4: Periodisches Auftreten einer Nachricht in einer Zeitstempel-Sequenz

7.4.2 Zuordnung der Nachrichten in Anwendungsszenarios auf Nachrichten im Simulationsmodell

Die Verhaltensspezifikation, bestehend aus einer Menge von Anwendungsszenarios, und das Simulationsmodell liegen auf zwei unterschiedlichen Abstraktionsebenen. In den Anwendungsszenarios werden die Nachrichten zwischen den Akteuren versendet. Im Simulationsmodell des komponentenbasierten Entwurfsmodells hingegen werden die Nachrichten zwischen Komponenten versendet. Aus diesem Grund kann nicht vorausgesetzt werden, dass die Nachrichten und ihre Bezeichner in den Anwendungsszenarios und im Simulationsmodell gleich sind. Um eine automatisierte, rechnerunterstützte Konformitätsprüfung zu ermöglichen, muss deshalb eine Zuordnung der Nachrichten und der logischen Ausdrücke in den Nachrichten-Bedingungen durchgeführt werden. Abbildung 7.5 zeigt beispielhaft die Zuordnung der Nachrichten zwischen Anwendungsszenarios und den Komponenten-Schnittstellen im Simulationsmodell.

Bei der Zeitstempel-Aufzeichnung während der Simulation wird bei den meisten Simulationswerkzeugen zur Vereinfachung der Messtechnik nicht zwischen Versenden und Empfangen einer Nachricht unterschieden und nur das Versenden einer Nachricht als Zeitstempel protokolliert. Die Zuordnung der Nachrichten erfolgt deshalb primär auf die Ausgänge der Komponenten im Simulationsmodell.

Bei der Konformitätsprüfung werden alle spezifizierten Anwendungsszenarios vollständig geprüft. Deshalb müssen alle in den Anwendungsszenarios verwendeten Nachrichten und die gegebenenfalls zugehörigen Bedingungen auf Nachrichten und Systemvariablen im Simulationsmodell zugeordnet werden. Im einfachsten Fall sind sie in beiden Abstraktionsebenen

identisch. Andernfalls wird eine Nachricht im Anwendungsszenario auf das Versenden einer Nachricht am Ereignis-Ausgang einer Komponente im Simulationsmodell zugeordnet. Abhängig vom Inhalt der Nachricht kann die Zuordnung auch auf einen oder mehrere Daten-Ausgänge von Komponenten im Simulationsmodell erfolgen. Der Übergang der Daten-Ausgänge in einen festgelegten Zustand wird dabei als Zeitpunkt für das Versenden der Nachricht definiert. Der Zustand wird dabei über einen definierten Wert oder Wertebereich der zugeordneten Daten-Ausgänge festgelegt.

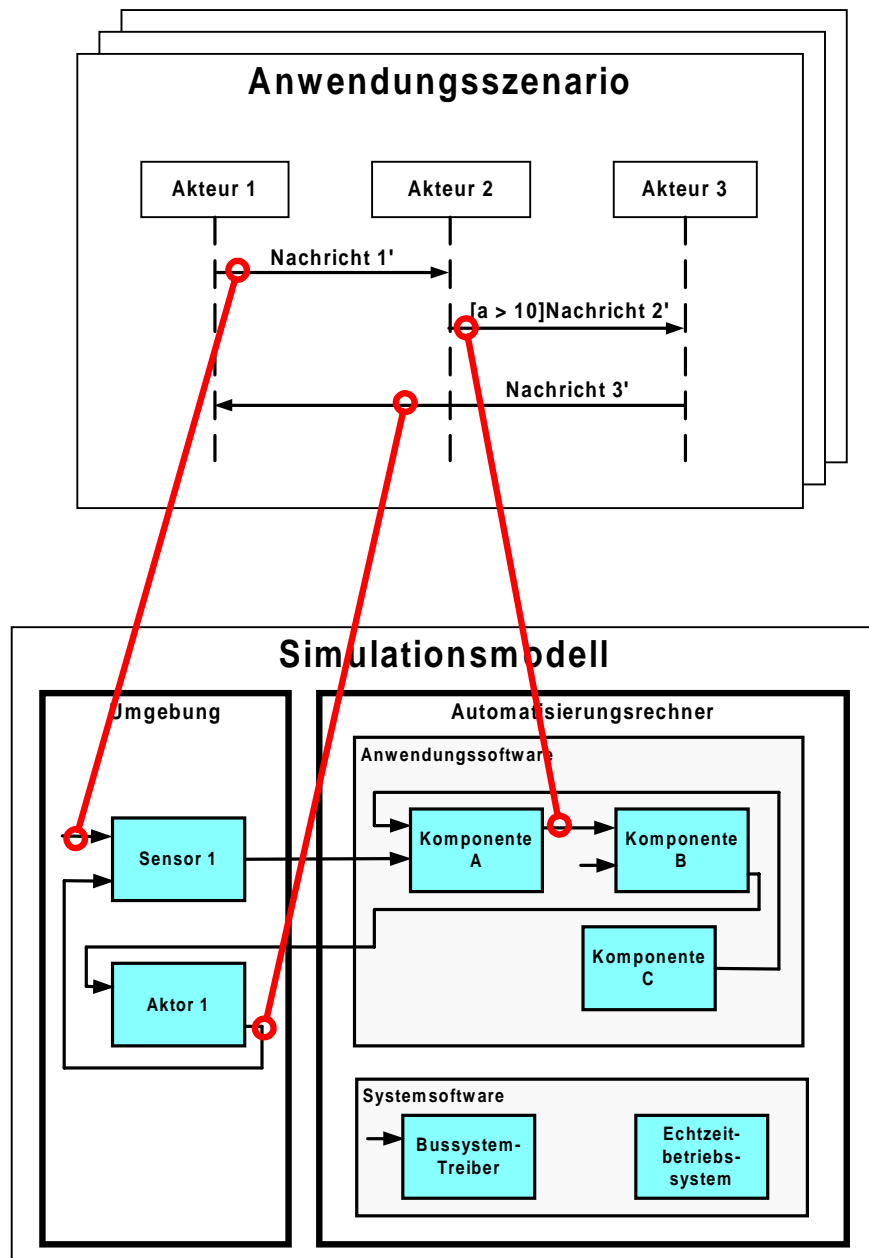


Abbildung 7.5: Zuordnung von Nachrichten zwischen den Anwendungsszenarios und den Komponenten-Schnittstellen in einem Simulationsmodell

Ist für eine Nachricht im Anwendungsszenario zusätzlich eine Bedingung durch einen logischen Ausdruck spezifiziert, so wird diese auf einen logischen Ausdruck – bestehend aus einem oder

mehreren Daten-Ausgängen – von Komponenten oder Systemvariablen zugeordnet. In der Zeitstempel-Sequenz wird die Nachricht nur dann protokolliert, wenn die Nachricht auftritt und gleichzeitig die zugehörige Bedingung erfüllt ist.

Das Ergebnis der Zuordnung wird in einer Zuordnungstabelle festgehalten. Tabelle 7.1 zeigt exemplarisch die Zuordnung von Nachrichten und Nachrichten-Bedingungen aus den Anwendungsszenarios auf Nachrichten und Nachrichten-Bedingungen im Simulationsmodell, die bei der Simulation beobachtet und aufgezeichnet werden.

Tabelle 7.1: Zuordnungstabelle für Nachrichten und Nachrichten-Bedingungen

Anwendungsszenarios	Simulationsmodell
Nachricht 1'	StimuliDataGenerator.Message1
[a > 10]Nachricht 2'	[KomponenteA.Ausgang1 > 10]
Nachricht 3'	...
...	...

Im ersten Eintrag der Zuordnungstabelle wird beispielsweise die im Anwendungsszenario spezifizierte Nachricht 1' auf den Ausgang StimuliDataGenerator.Message1 der Stimulations-Daten-Generator-Komponente des Simulationsmodells zugeordnet. Der zweite Eintrag zeigt die Zuordnung einer Nachricht auf den Daten-Ausgang einer Komponente im Simulationsmodell des komponentenbasierten Entwurfsmodells. Dem Auftreten der Nachricht 2' wird dabei im Simulationsmodell nur der logische Ausdruck [KomponenteA.Ausgang1 > 10] für den Daten-Ausgang der Komponente A zugeordnet.

Aus den gesammelten Zuordnungen der Nachrichten und Nachrichten-Bedingungen einer Menge von Anwendungsszenarios ergeben sich insgesamt auch die notwendigen Instrumentierungspunkte im Simulationsmodell zur Aufzeichnung der Zeitstempel-Sequenzen (siehe auch Abschnitt 6.1.7).

Für Anwendungsszenarios, deren Ablauf sich über mehrere Automatisierungsrechner verteilt, ist bei der Zuordnung (siehe Abbildung 7.6) folgendes zu berücksichtigen: Die Zuordnung von Nachrichten, die eine Zeitbedingung zu einer Vorgänger-Nachricht haben und bei denen das Versenden auf einem Automatisierungsrechner 1 und das Empfangen auf einem anderen Automatisierungsrechner 2 stattfindet, sollte auf die Ausgänge der Stellvertreter-Komponente auf dem empfangenden Automatisierungsrechner 2 erfolgen, damit bei der Konformitätsprüfung auch die Zeitverzögerung durch das Bussystem berücksichtigt wird.

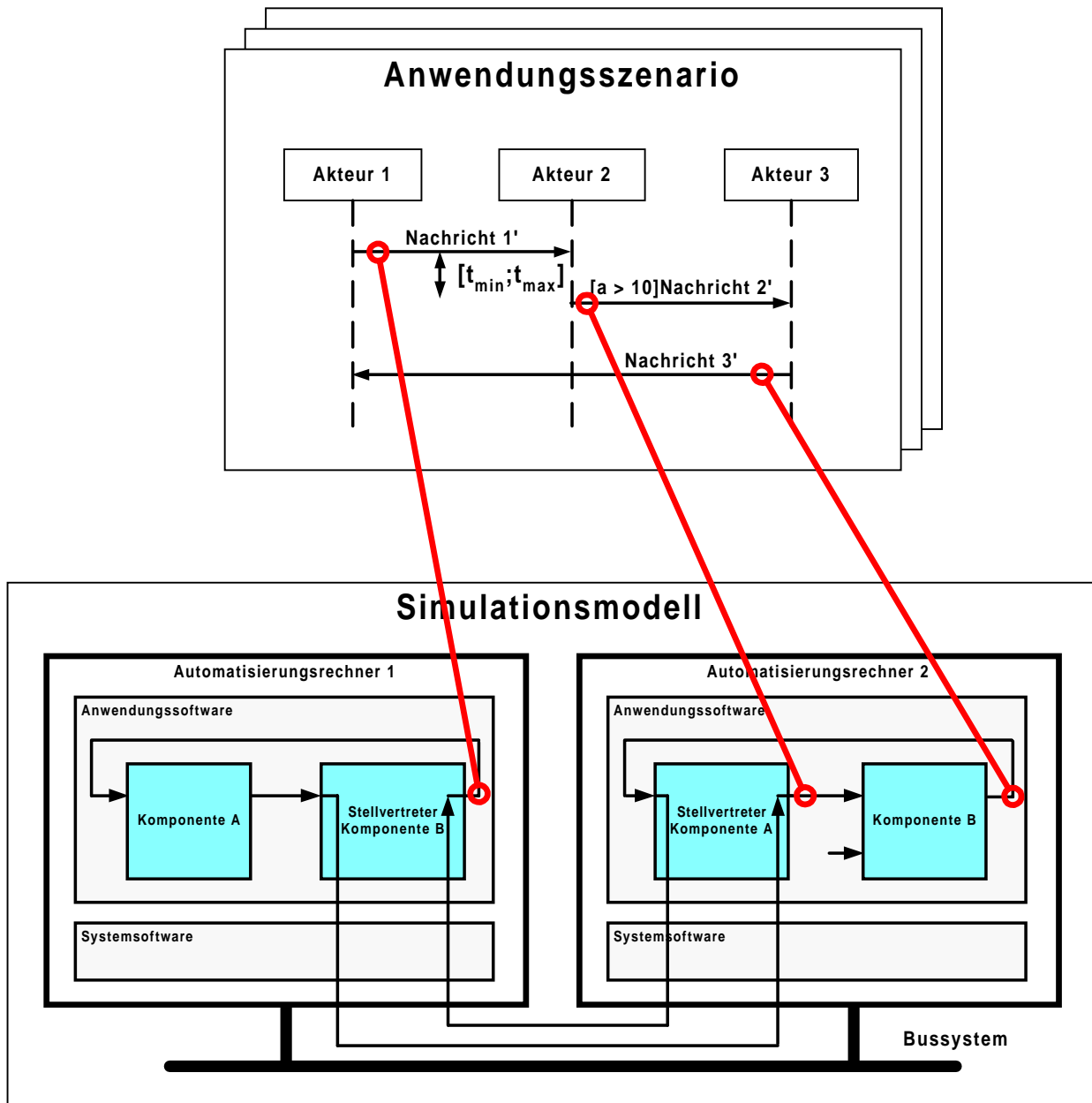


Abbildung 7.6: Zuordnung von Nachrichten bei verteilten Echtzeitsystemen

7.4.3 Prüfung des Verhaltens eines gesamten komponentenbasierten Entwurfsmodells

Zur Prüfung des Verhaltens eines gesamten komponentenbasierten Entwurfsmodells werden alle spezifizierten Anwendungsszenarios gegenüber den aufgezeichneten Zeitstempel-Sequenzen auf Konformität überprüft. Das Ziel der Prüfung ist dabei die Aufdeckung möglichst vieler Fehler im simulierten Verhalten des komponentenbasierten Entwurfsmodells und der vollständige Nachweis aller spezifizierten Anwendungsszenarios. Dazu wird die nachfolgend beschriebene Prüfstrategie festgelegt. Abbildung 7.7 zeigt eine Übersicht über die Prüfstrategie.

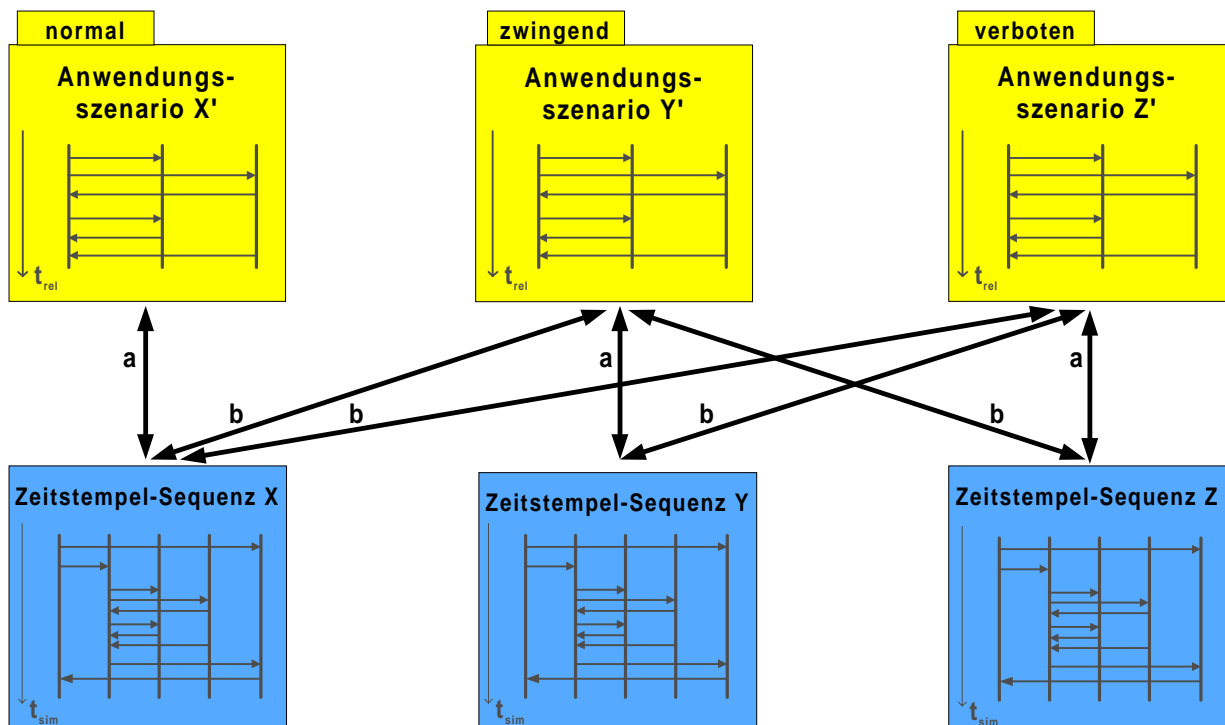


Abbildung 7.7: Prüfstrategie zur Prüfung des Verhaltens eines gesamten komponentenbasierten Entwurfsmodells

Zur systematischen Prüfung wird zunächst jede aufgezeichnete Zeitstempel-Sequenz gegenüber ihrem zugehörigen Anwendungsszenario geprüft (a). Die zugehörigen Paare, bestehend aus Anwendungsszenario und Zeitstempel-Sequenz, z.B. X' und X, sind in Abbildung 7.7 in vertikaler Richtung untereinander dargestellt. Die aus dem Anwendungsszenario abgeleitete Stimuli-Sequenz versucht bei der Simulationsausführung explizit die Vorbedingung und das im Anwendungsszenario beschriebene Verhalten in der Simulation herbeizuführen. Bei Anwendungsszenarios vom Sequenztyp „normal“ und „zwingend“ muss deshalb die Sequenz von Nachrichten in der aufgezeichneten Zeitstempel-Sequenz vollständig enthalten sein. Bei Anwendungsszenarios vom Sequenztyp „verboten“ darf nur die Vorbedingung erfüllt sein und die Sequenz von Nachrichten in der aufgezeichneten Zeitstempel-Sequenz nicht vollständig enthalten sein.

Für alle aufgezeichneten Zeitstempel-Sequenzen werden weitere Prüfungen vorgenommen. Jede Zeitstempel-Sequenz wird dazu auch gegenüber allen nicht-zugehörigen Anwendungsszenarios vom Sequenztyp „zwingend“ und „verboten“ geprüft (b). Nicht geprüft wird sie gegen nicht-zugehörige Anwendungsszenarios vom Sequenztyp „normal“, weil diese von ihrer Bestimmung her nur bei explizit herbeigeführter Vorbedingung das geforderte Verhalten aufweisen müssen. Dadurch werden unbegründete Fehlermeldungen im Prüfergebnis vermieden.

Die Prüfung einer Zeitstempel-Sequenz gegen die nicht-zugehörigen Anwendungsszenarios vom Sequenztyp „zwingend“ und „verboten“ läuft prinzipiell gleich ab, wie gegen die zugehörige Zeitstempel-Sequenz. Da aber nicht explizit über eine zugehörige Stimuli-Sequenz

stimuliert wurde, muss das nicht-zugehörige Anwendungsszenario nicht zwingend in der Zeitstempel-Sequenz enthalten sein. Die Schnittmenge ist also oft leer. Nur wenn die Vorbedingung in der Zeitstempel-Sequenz erfüllt ist, muss auch das nicht-zugehörige Anwendungsszenario vom Sequenztyp „zwingend“ vollständig enthalten sein. Bei der Prüfung gegenüber einem Anwendungsszenario vom Sequenztyp „verboten“ darf die Sequenz von Nachrichten natürlich nicht vollständig enthalten sein.

7.5 Prüfergebnis

7.5.1 Prüfergebnis für einzelne Zeitstempel-Sequenzen

Jede einzelne Konformitätsprüfung zwischen einer Zeitstempel-Sequenz und einem Anwendungsszenario liefert ein eindeutiges Prüfergebnis, „validiert“ oder „fehlerhaft“. Tabelle 7.2 zeigt eine Übersicht über die möglichen Prüfergebnisse.

Tabelle 7.2: Entscheidungstabelle zur Ermittlung des Prüfergebnisses

Erfüllung der Prüfkriterien durch die Zeitstempel-Sequenz		Prüfergebnis abhängig vom Sequenztyp des Anwendungsszenarios		
Vorbedingung	Sequenz der Nachrichten (einschl. Nachbedingung)	„normal“	„zwingend“	„verboten“
erfüllt	vollständig	validiert	validiert	fehlerhaft
erfüllt	vollständig, aber mindestens eine Zeitbedingung nicht erfüllt	fehlerhaft	fehlerhaft	validiert
erfüllt	unvollständig oder falsche Reihenfolge	fehlerhaft	fehlerhaft	validiert
nicht erfüllt	-	fehlerhaft	validiert / fehlerhaft ⁷	validiert

Das Prüfergebnis „validiert“ ergibt sich immer dann, wenn eine Zeitstempel-Sequenz bei der Konformitätsprüfung gegen ein Anwendungsszenario alle Prüfkriterien erfüllt. Dies ist ebenso der Fall, wenn die Vorbedingung nicht erfüllt ist und der Sequenztyp „zwingend“ oder „verboten“ ist. Bei der Prüfung gegen Anwendungsszenarios vom Sequenztyp „verboten“ dürfen für das Prüfergebnis „validiert“ nie alle Prüfkriterien erfüllt sein.

⁷ fehlerhaft nur bei der Prüfung gegen die zugehörige Zeitstempel-Sequenz

Alle anderen auftretenden Fälle liefern das Prüfergebnis „fehlerhaft“ und decken damit Abweichungen zwischen dem simulierten Verhalten des komponentenbasierten Entwurfsmodells und dem in den Anwendungsszenarios geforderten Verhalten auf.

Im Prüfergebnis stehen neben dem Endergebnis „validiert“ oder „fehlerhaft“ zusätzliche Informationen zur Verfügung, die für die Aufdeckung von Fehlern hilfreich sind. So wird das Auftreten der einzelnen Nachrichten als Zeitstempel, bestehend aus aktueller Simulationszeit und Nachrichten-Bezeichner, protokolliert. Ist bei einer Nachricht die Zeitbedingung zu einer Vorgänger-Nachricht verletzt, wird dies ebenfalls protokolliert. Am Ende jeder einzelnen Konformitätsprüfung wird festgehalten, ob die Sequenz der Nachrichten vollständig oder unvollständig enthalten ist.

7.5.2 Fehlerursachen und Auswirkungen

Ein ungültiges Prüfergebnis lässt auf Entwurfsfehler im komponentenbasierten Entwurfsmodell oder auf Fehler bei der Spezifikation der Anwendungsfälle bzw. der Anwendungsszenarios schließen. Tabelle 7.3 zeigt eine Übersicht über die charakteristischen Fehlerursachen und deren Auswirkungen beim Entwurf komponentenbasierter Software für Echtzeitsysteme.

Tabelle 7.3: Charakteristische Fehlerursachen und deren Auswirkungen beim Entwurf komponentenbasierter Software für Echtzeitsysteme

Entwurfstätigkeit	Fehlerursache	Auswirkung
Auswahl	falsche Komponente	Zuordnung der Nachrichten nicht eindeutig möglich
Konfigurierung	falsch eingestellte Parameter	Nachricht <ul style="list-style-type: none"> • fehlt • zu spät • zu früh • mit falscher Information
Instanziierung	zu viele Instanzen	Zuordnung der Nachrichten nicht eindeutig möglich
	zu wenig Instanzen	
Verbindung	fehlende Verbindung	Nachricht <ul style="list-style-type: none"> • fehlt • zu spät • zu früh • mit falscher Information
	falsche Verbindung	

Die typischen Fehlerursachen bei der Auswahl und Instanziierung von Komponenten sind die Auswahl falscher Komponenten und die Instanziierung zu vieler bzw. zu weniger Komponenten. Diese Fehler zeigen sich in der Praxis meistens schon bei der statischen Zuordnung der Nachrichten. Fehlt beispielsweise eine Instanz einer Komponente oder wurde die falsche Komponente instanziiert, lassen sich in der Regel die Nachrichten mindestens eines Anwendungsszenarios nicht vollständig zuordnen. Die Konfigurierung von Komponenten mit falschen Parametern oder fehlende bzw. falsche Verbindungen zeigen sich meistens erst bei der Konformitätsprüfung der aufgezeichneten Zeitstempel-Sequenzen. Typische Auswirkungen sind dabei das Ausbleiben sowie das zu frühe oder zu späte Auftreten von geforderten Nachrichten. Teilweise entstehen dadurch auch Nachrichten mit falschen Informationen, die dann ebenfalls in fehlerhaftem Verhalten resultieren.

Prinzipiell besteht auch die Möglichkeit, dass die Fehlerursache davon herrührt, dass ein Anwendungsszenario falsch spezifiziert oder nicht prüfbar spezifiziert wurde. Die festgelegte Reihenfolge der Nachrichten eines Anwendungsszenarios kann beispielsweise falsch gewählt worden sein, oder es wird an einer Stelle eines Anwendungsszenarios eine falsche Nachricht gefordert. Durch die Erweiterung der UML Sequenzdiagramme zur Spezifikation von Zeitbedingungen besteht theoretisch auch die Möglichkeit, sich widersprechende Zeitbedingungen innerhalb eines Anwendungsszenarios zu spezifizieren. Dieser Fall ist bei den bisher durchgeführten praktischen Evaluierungen des Prüfverfahrens jedoch noch nicht aufgetreten.

7.5.3 Aufdeckung und Lokalisierung von Entwurfsfehlern

Die Aufdeckung der im vorigen Abschnitt beschriebenen Entwurfsfehler im komponentenbasierten Entwurfsmodell erfolgt anhand der fehlerhaften Prüfergebnisse der geprüften Zeitstempel-Sequenzen. Nachfolgend sind einige Erfahrungswerte für die Aufdeckung und Lokalisierung solcher Entwurfsfehler zusammengestellt. Ausgangspunkt sind die folgenden drei Abweichungen von den Prüfkriterien im Prüfergebnis:

- Vorbedingung nicht erfüllt
- Sequenz der Nachrichten unvollständig
- Zeitbedingung zwischen zwei Nachrichten nicht erfüllt

Bei nicht erfüllter Vorbedingung liegt die Ursache meist in einer unzureichenden Stimuli-Sequenz, welche die Vorbedingung nicht explizit herbeiführen konnte.

Ist die Sequenz der Nachrichten unvollständig, kann in der Regel auf fehlende oder falsche Verbindungen, auf falsche Komponenten oder auf falsch konfigurierte Komponenten geschlossen werden. Ausgangspunkt für die Eingrenzung des Fehlers ist die letzte protokollierte Nachricht in der Zeitstempel-Sequenz, welche von einer Komponente gesendet wurde. Ausgehend von dieser Komponente kann die Verbindung zur empfangenden Komponente zurückverfolgt

werden, um die Ursache für die fehlende nachfolgende Nachricht ausfindig zu machen. Beispielsweise kann die Verbindung zu einer Komponente fehlen oder die empfangende Komponente wurde falsch konfiguriert und reagiert nicht auf die empfangene Nachricht.

Ist die Zeitbedingung zwischen zwei Nachrichten nicht erfüllt, liegt ein Fehler im Entwurf des zeitlichen Verhaltens vor. Die Eingrenzung des Fehlers orientiert sich an der Komponente, die die Referenz-Nachricht sendet, und an der Komponente, deren versendete Nachricht die Zeitbedingung verletzt. Als Ursachen kommen zu große Laufzeiten der dazwischen liegenden Komponenten oder eine falsche Konfigurierung der Ausführungssteuerungs-Komponente, wie z.B. einem Echtzeitbetriebssystem, in Frage.

Zu wenige Instanzen von Komponenten fallen in der Praxis schon bei der Zuordnung der Nachrichten auf. Eine ungeeignete Zuordnung der Nachrichten kann auch die Ursache für ein fehlerhaftes Prüfergebnis sein.

Die praktischen Erfahrungen haben gezeigt, dass zwar vergleichbar zu herkömmlichen Prüfverfahren eine individuelle Lokalisierung der aufgedeckten Entwurfsfehler notwendig ist, dass aber die Fehlerursache im komponentenbasierten Entwurfsmodell meistens sehr schnell lokalisiert werden kann.

Das in diesem Kapitel vorgestellte Prüfverfahren ermöglicht die systematische Prüfung des simulierten Verhaltens einer komponentenbasierten Software. Durch die Ableitung von Stimuli-Sequenzen aus den spezifizierten Anwendungsszenarios werden systematisch Prüffälle entwickelt, die in der Simulation sequenziell oder überlagert ausgeführt werden. Mit Hilfe der Prüfkriterien für die Konformitätsprüfung zwischen 2 Sequenzdiagrammen und der festgelegten Prüfstrategie wird das simulierte Verhalten gegenüber einer Menge von Anwendungsszenarios validiert. Da die einzelnen Schritte des Prüfverfahrens überwiegend definierte Abläufe enthalten, ergeben sich daraus Automatisierungsmöglichkeiten. Außerdem bietet es sich an die im Kapitel 5 konzipierte grafische Notation der erweiterten UML Sequenzdiagramme zur Formalisierung der Anwendungsfälle durch Softwarewerkzeuge zu unterstützen. Im nachfolgenden Kapitel wird deshalb eine Validierungsumgebung präsentiert, die das konzipierte Prüfverfahren durch Softwarewerkzeuge unterstützt und damit in der Praxis anwendbar macht.

8 Realisierung einer Validierungsumgebung

Im Rahmen der vorliegenden Arbeit wurde eine Validierungsumgebung für die Prüfung von komponentenbasierter Software für Echtzeitsysteme realisiert. Die Validierungsumgebung unterstützt durch Softwarewerkzeuge das gesamte Prüfverfahren und bietet Schnittstellen zu den kommerziellen Modellierungs- und Simulationswerkzeugen ASCET-SD und CANoe. Die Unterstützung des Prüfverfahrens reicht von der systematischen Entwicklung von Prüffällen bis hin zur automatisierten Prüfung des in der Simulation aufgezeichneten Verhaltens der komponentenbasierten Software.

8.1 Systemarchitektur der Validierungsumgebung

Die Systemarchitektur der Validierungsumgebung "Automated Validation Environment" (AVE) ist in Abbildung 8.1 dargestellt.

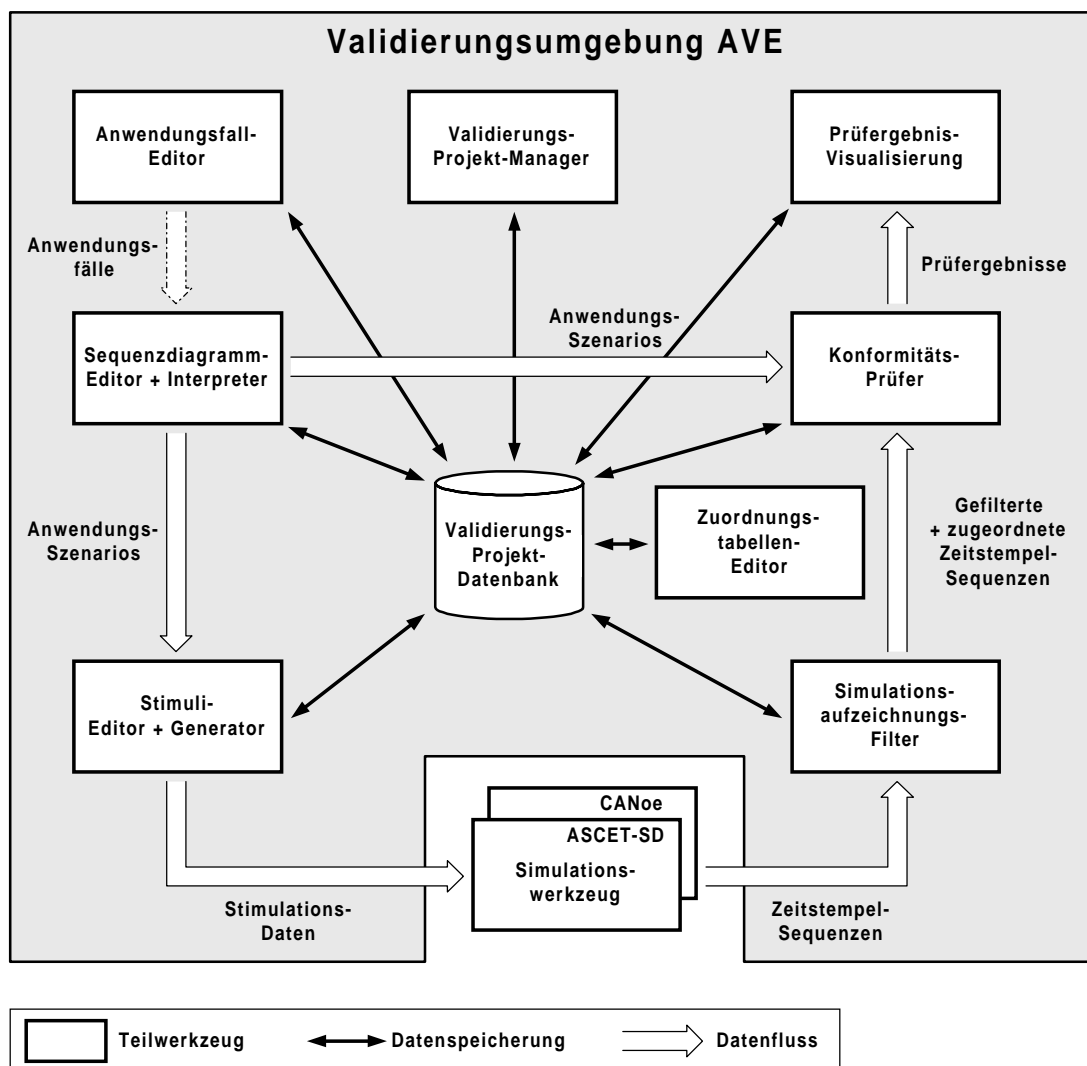


Abbildung 8.1: Systemarchitektur der Validierungsumgebung AVE

Die Validierungsumgebung AVE umfasst folgende Teilwerkzeuge:

- Anwendungsfall-Editor
- Sequenzdiagramm-Editor und Interpreter
- Zuordnungstabellen-Editor
- Stimuli-Editor und Generator
- Simulationsaufzeichnungs-Filter
- Konformitäts-Prüfer
- Prüfergebnis-Visualisierung

Zwischen den Teilwerkzeugen werden Daten zur Weiterverarbeitung ausgetauscht. Die einzige Ausnahme besteht zwischen dem Anwendungsfall-Editor und dem Sequenzdiagramm-Editor, da in diesem Schritt die informellen Anwendungsfälle in formalisierte Anwendungsszenarios ausschließlich manuell überführt werden. Alle persistenten Daten der Teilwerkzeuge werden in einer zentralen Validierungs-Projekt-Datenbank gespeichert.

Für die Echtzeitsimulation von komponentenbasierter Software verfügt AVE über Schnittstellen zu den kommerziellen Simulationswerkzeugen ASCET-SD 4.0 [ETAS00] und CANoe 3.0 [Vect00].

AVE wurde objektorientiert mit MS-Visual C++/MFC 5.0 entwickelt [Romm99]. Abbildung 8.2 zeigt das Menü der grafischen Benutzeroberfläche.

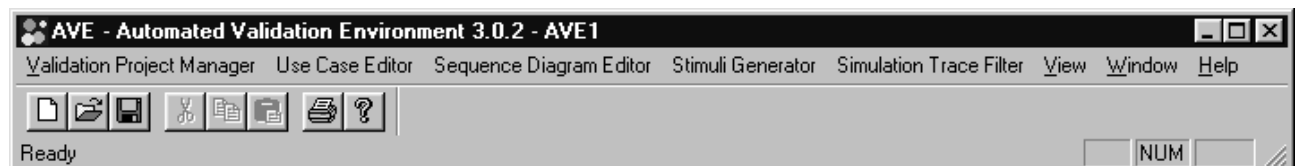


Abbildung 8.2: Benutzeroberfläche der Validierungsumgebung AVE

Über die Pull-Down-Menüs der grafischen Benutzeroberfläche werden die Funktionalitäten der verschiedenen Teilwerkzeuge aufgerufen. Die Teilwerkzeuge werden in den nachfolgenden Abschnitten detaillierter vorgestellt.

8.2 Validierungs-Projekt-Manager

Mit dem Validierungs-Projekt-Manager werden Validierungs-Projekte verwaltet. Ein Validierungs-Projekt besteht aus Anwendungsfällen, Anwendungsszenarios, Zuordnungstabelle, Stimuli-Sequenzen, Zeitstempel-Sequenzen und Prüfergebnissen. Bei der Neuerstellung bzw. beim Öffnen eines Validierungs-Projekts werden die wesentlichen Konfigurations-Daten, wie z.B. der Pfadname des Basis-Verzeichnisses der Validierungs-Projekt-Datenbank und das verwendete Simulationswerkzeug, festgelegt. Abbildung 8.3 zeigt das Menü des Validierungs-

Projekt-Managers.



Abbildung 8.3: Validierungs-Projekt-Manager-Menü

Neben der Validierungs-Projekt-Verwaltung wird über das Menü des Validierungs-Projekt-Managers auch die Prüfung einzelner Zeitstempel-Sequenzen gegenüber einzelnen Anwendungsszenarios und die Prüfung des ganzen Validierungs-Projekts gestartet. Der Validierungs-Projekt-Manager steuert dabei die anderen Teilwerkzeuge.

8.3 Validierungs-Projekt-Datenbank

Für jedes Validierungs-Projekt wird im ausgewählten Basis-Verzeichnis die zentrale Validierungs-Projekt-Datenbank angelegt. Die Validierungs-Projekt-Datenbank der prototypisch implementierten Validierungsumgebung basiert auf einer kombinierten Verzeichnis- und Dateistruktur.

Die Validierungs-Projekt-Datenbank speichert im Basis-Verzeichnis die Konfigurations-Daten des Validierungs-Projekts. Wesentliche Konfigurations-Daten sind der Pfadname des Basis-Verzeichnisses und das verwendete Simulationswerkzeug. Ebenso werden im Basis-Verzeichnis die zentrale Zuordnungstabelle für die Zuordnung der Nachrichten zwischen Anwendungsszenarios und Simulationsmodell und der automatisch generierte Prüfbericht für das Validierungs-Projekt gespeichert.

In den Unterverzeichnissen werden die Daten bezüglich der einzelnen Anwendungsszenarios angelegt. Die vergebenen Namen für die Unterverzeichnisse sind identisch mit den Bezeichnern der darin gespeicherten Anwendungsszenarios.

8.4 Anwendungsfall-Editor

Als Anwendungsfall-Editor wird von AVE das Textverarbeitungsprogramm MS-Word über das Menü aufgerufen, wie in Abbildung 8.4 zu sehen ist. Die Anwendungsfall-Spezifikations-schablone, die in Abschnitt 5.1.2 vorgestellt wurde, ist dazu als MS-Word Dokument-Vorlage hinterlegt.



Abbildung 8.4: Anwendungsfall-Editor-Menü

8.5 Sequenzdiagramm-Editor und Interpreter

Die Anwendungsszenarios können mit Hilfe des Sequenzdiagramm-Editors grafisch spezifiziert werden. Zur grafischen Modellierung von erweiterten Sequenzdiagrammen wurde das Grafikprogramm Visio Professional 5.0 [Micro98] in die Validierungsumgebung integriert. Abbildung 8.5 zeigt das Menü des Sequenzdiagramm-Editors und Interpreters, über das mit Hilfe der MS-COM-Schnittstelle das Grafikprogramm Visio gesteuert wird.

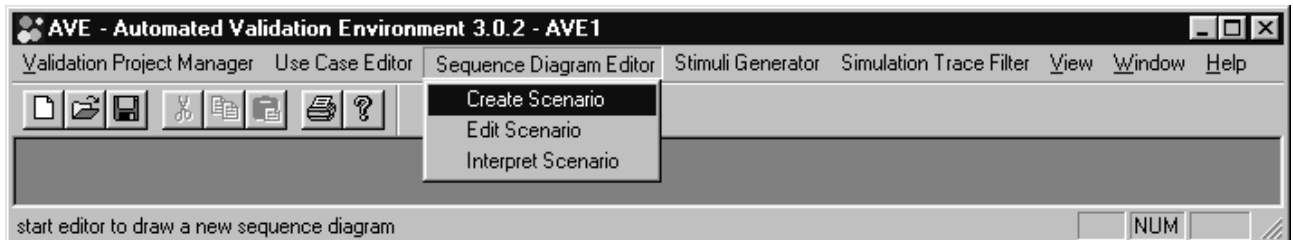


Abbildung 8.5: Sequenzdiagramm-Editor und Interpreter-Menü

Abbildung 8.6 zeigt den grafischen Sequenzdiagramm-Editor. Im linken Teilfenster stehen die Modellierungselemente für Akteure und Nachrichten zur Verfügung, die bei der Erstellung eines Anwendungsszenarios per „Drag and Drop“ mit der Maus in die rechte Hälfte gezogen werden.

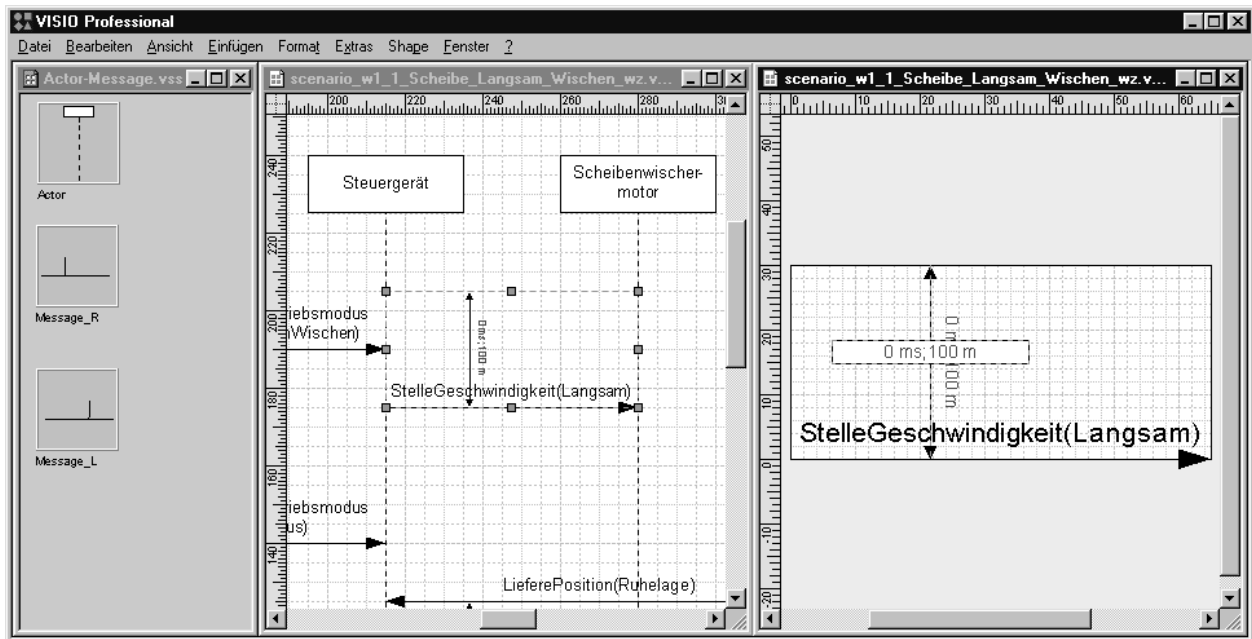


Abbildung 8.6: Grafischer Editor für erweiterte UML Sequenzdiagramme

Der auf Basis von Visio entwickelte Sequenzdiagramm-Editor unterstützt die Spezifikation aller in Abschnitt 5.3 entwickelten Eigenschaften der erweiterten UML Sequenzdiagramme. Durch Doppelklick auf einen Akteur oder Nachricht öffnet sich ein neues Teilfenster für das instanziierte Modellierungselement, in welchem die Attribute des Akteurs oder der Nachricht, wie z.B. Bedingung, Nachrichten-Bezeichner oder Zeitbedingung, eingegeben werden können.

Die grafisch spezifizierten Anwendungsszenarios werden mit Hilfe des Sequenzdiagramm-Interpreters ausgewertet. Dazu werden die in der Grafik enthaltenen Informationen von AVE über die Visio-COM-Schnittstelle ausgelesen und zur Weiterverarbeitung in der Validierungsprojekt-Datenbank gespeichert. Beim Auslesen der Grafik führt der Sequenzdiagramm-Interpreter zusätzlich einen Abgleich mit der Zuordnungstabelle durch. Ist ein Nachrichten-Bezeichner oder eine Systemvariable aus der Bedingung noch nicht zugeordnet, so erfolgt ein automatischer Eintrag mit Default-Zuordnung in die Zuordnungstabelle. Der Benutzer wird in diesem Fall durch ein Dialog-Fenster informiert, dass eine Default-Zuordnung eingetragen wurde und dass die Default-Zuordnung vom ihm gegebenenfalls verändert werden soll. Abbildung 8.7 zeigt das Dialog-Fenster bei einem automatischen Neueintrag in die Zuordnungstabelle.



Abbildung 8.7: Dialog-Fenster bei automatischer Änderung der Zuordnungstabelle

Außerdem prüft der Sequenzdiagramm-Interpreter bei der Interpretation der Grafik die Einhaltung der definierten Syntax für die Nachrichten-Bedingungen und Zeitbedingungen. Bei Nicht-Einhaltung der Syntax wird der Benutzer ebenfalls über Dialog-Fenster über die Syntax-Fehler informiert.

8.6 Zuordnungstabellen-Editor

Zur Bearbeitung der Zuordnungstabelle wird über das Pull-Down-Menü des Validierungs-Projekt-Managers ein Standard-ASCII-Editor geöffnet. Die Zuordnungstabelle ist abhängig von der Struktur der Zeitstempel-Aufzeichnungen des verwendeten Simulationswerkzeugs. Ist für das ausgewählte Validierungs-Projekt noch keine Zuordnungstabelle vorhanden, so bietet ein Dialog-Fenster beim Öffnen die Übernahme einer bestehenden Zuordnungstabelle aus einem anderen Validierungs-Projekt an.

8.7 Stimuli-Editor und Generator

Der Stimuli-Editor und Generator unterstützt die Ableitung von Stimuli-Sequenzen aus den Anwendungsszenarios. Abbildung 8.8 zeigt die Funktionen des Stimuli-Editors und Generators.

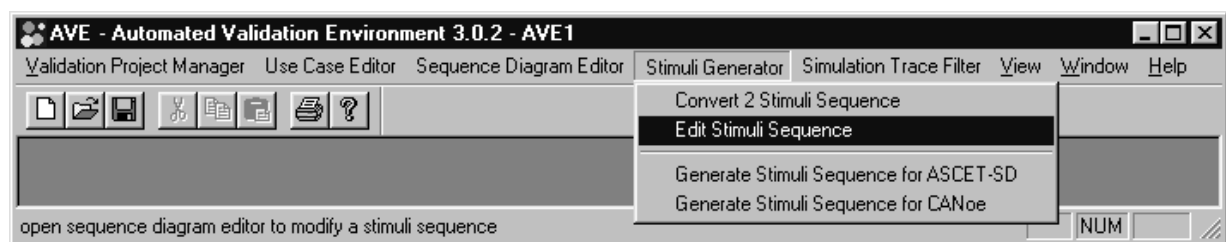


Abbildung 8.8: Stimuli-Editor und Generator-Menü

Mit „Convert 2 Stimuli Sequence“ wird aus einem ausgewählten Anwendungsszenario eine Ausgangsbasis für eine Stimuli-Sequenz generiert. Bei der Generierung werden aus dem Anwendungsszenario nur die versendeten Nachrichten des Akteurs übernommen, welcher die erste Nachricht im Anwendungsszenario versendet. Dieser Akteur wird nach der Konvertierung als Stimuli-Generator-Komponente bezeichnet. Die Nachrichten in der Stimuli-Sequenz werden alle an die Empfänger-Komponente Simulationsmodell versendet. Bei der Konvertierung des Anwendungsszenarios werden die logischen Ausdrücke in den Nachrichten-Bedingungen und die Nachrichten-Bezeichner durch die zugeordneten Einträge für das Simulationsmodell aus der Zuordnungstabelle ersetzt. Enthält eine Nachricht im Anwendungsszenario eine Bedingung, wie z.B. die PRECONDITION bei der ersten Nachricht, werden bei der Konvertierung die Bedingung und die Nachricht in zwei getrennte sequenzielle Nachrichten aufgeteilt. Die Pseudo-Nachricht mit POSTCONDITION wird bei der Konvertierung nicht übernommen, da sie nur für die Prüfung relevant ist.

Mit dem Stimuli-Editor, wie in Abbildung 8.9 zu sehen, wird die Stimuli-Sequenz als Sequenzdiagramm manuell vervollständigt.

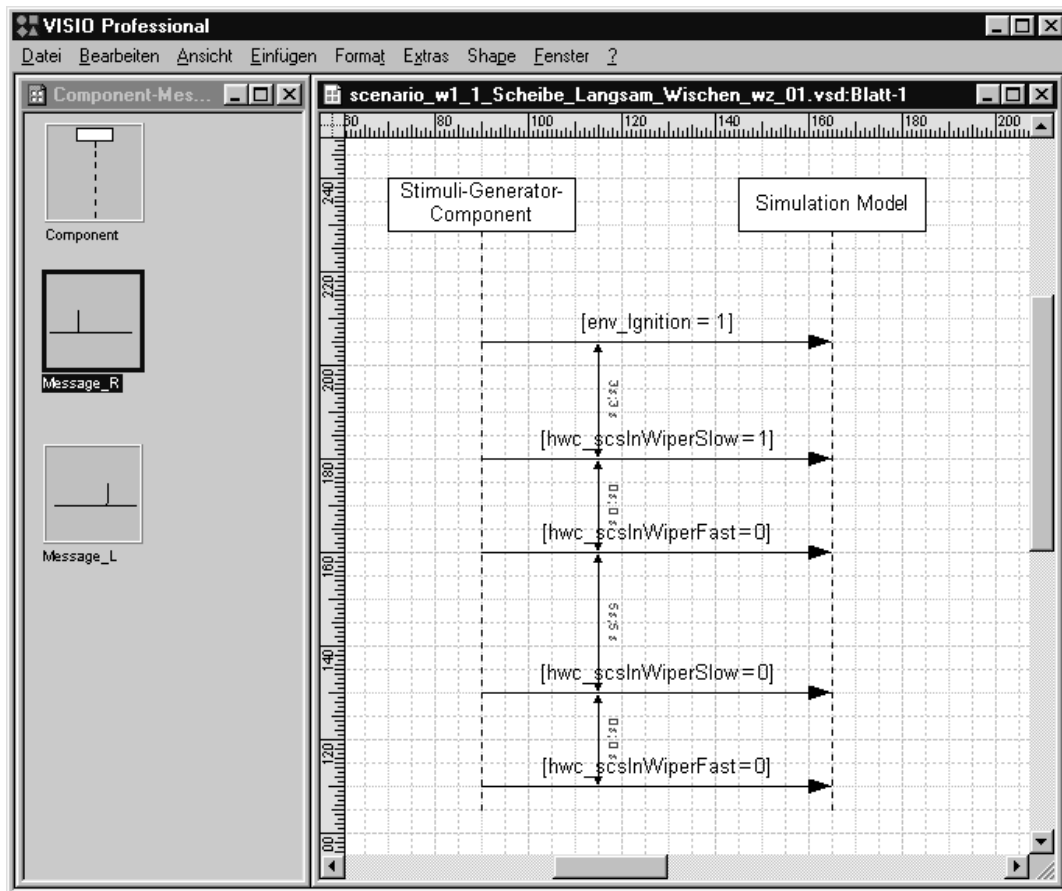


Abbildung 8.9: Grafischer Stimuli-Editor

In den Stimuli-Sequenzen werden im Gegensatz zu den Anwendungsszenarios keine Zeitbedingungen sondern Zeitpunkte spezifiziert. Da der grafische Stimuli-Editor auf dem Sequenzdiagramm-Editor basiert, werden die Zeitpunkte durch Intervalle mit gleichem Anfangs- und Endwert spezifiziert. Als Default-Zeitpunkt bei der automatischen Konvertierung wird Null verwendet. Die konvertierte Stimuli-Sequenz muss in der Praxis fast immer manuell nachbearbeitet werden. Auch alternative Stimuli-Sequenzen werden mit Hilfe des Stimuli-Editors erstellt.

Aus den grafisch spezifizierten Stimuli-Sequenzen werden über die Visio-COM-Schnittstelle die unterschiedlichen ASCII-Formate der Stimulations-Daten für die Simulationswerkzeuge ASCET-SD und CANoe generiert und in der Validierungs-Projekt-Datenbank gespeichert. Diese Stimulations-Daten werden dann zur Stimulierung des Simulationsmodells bei der Simulationsausführung verwendet.

8.8 Simulationsaufzeichnungs-Filter

Da jedes Simulationswerkzeug ein eigenes Daten-Format bei der Aufzeichnung von Zeitstempel-Sequenzen verwendet, wurden Simulationsaufzeichnungs-Filter entwickelt. Die verfügbaren Simulationsaufzeichnungs-Filter für ASCET-SD und CANoe transformieren die aufgezeichneten Zeitstempel-Sequenzen in ein von AVE prüfbares, einheitliches Format. Dabei werden zur Vorbereitung der Konformitätsprüfung gegenüber den Anwendungsszenarios die relevanten Nachrichten und Systemvariablen mit Hilfe der Zuordnungstabelle aus den aufgezeichneten Zeitstempel-Sequenzen herausgefiltert und zugeordnet. Für die Prüfung einzelner Zeitstempel-Sequenzen können diese einzeln über das Simulationsaufzeichnungs-Filter Menü gefiltert werden. Abbildung 8.10 zeigt den Aufruf zur Filterung in Abhängigkeit vom verwendeten Simulationswerkzeug.



Abbildung 8.10: Simulationsaufzeichnungs-Filter-Menü

8.9 Konformitäts-Prüfer

Der Konformitäts-Prüfer realisiert die Konformitätsprüfung von Zeitstempel-Sequenzen gegenüber den Anwendungsszenarios. Er besitzt kein eigenes Pull-Down-Menü, sondern wird über das Menü des Validierungs-Projekt-Managers aufgerufen.

Mit „Validate Single Scenario“ kann die Konformitätsprüfung einer einzelnen Zeitstempel-Sequenz gegenüber einem Anwendungsszenario durchgeführt werden. Vor der Konformitätsprüfung wird über ein Dialog-Fenster zwischen zwei unterschiedlichen Detaillierungsgraden des Prüfberichts ausgewählt. Abbildung 8.11 zeigt das Auswahl-Dialog-Fenster.

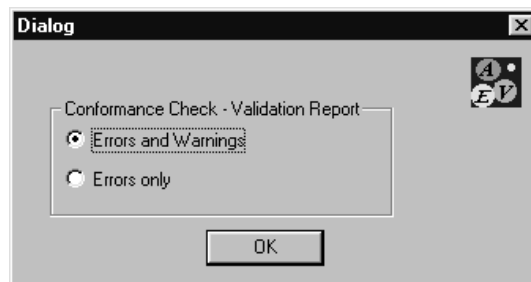


Abbildung 8.11: Auswahl-Dialog-Fenster für den Detaillierungsgrad des Prüfberichts

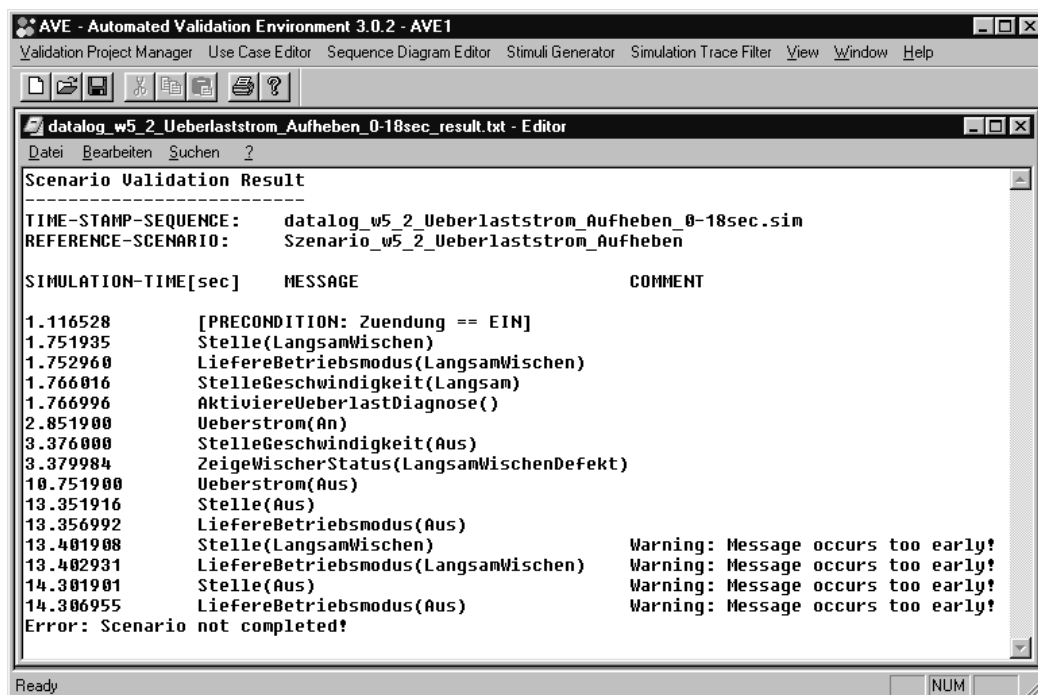
Mit „Validate Project“ wird die Prüfung eines ganzen Validierungs-Projekts durchgeführt. Der

Konformitäts-Prüfer führt dazu für alle in der Validierungs-Projekt-Datenbank gespeicherten Zeitstempel-Sequenzen eine Prüfung gegenüber allen vorhandenen Anwendungsszenarios gemäß der in Abschnitt 7.4.3 beschriebenen Prüfstrategie für das Verhalten eines gesamten komponentenbasierten Entwurfsmodells durch.

8.10 Prüfergebnis-Visualisierung

Die Prüfergebnis-Visualisierung liefert textuelle Prüfberichte und eine grafische Fehler-visualisierung im Sequenzdiagramm des verletzten Anwendungsszenarios.

Für jede einzelne Konformitätsprüfung einer Zeitstempel-Sequenz gegenüber einem Anwendungsszenario wird ein textueller Prüfbericht generiert. Abbildung 8.12 zeigt ein Beispiel für das Ergebnis einer einzelnen Konformitätsprüfung.



The screenshot shows a window titled "datalog_w5_2_Ueberlaststrom_Aufheben_0-18sec_result.txt - Editor" within the "AVE - Automated Validation Environment 3.0.2 - AVE1" application. The window displays a "Scenario Validation Result" report with the following content:

```

Scenario Validation Result
-----
TIME-STAMP-SEQUENCE:   datalog_w5_2_Ueberlaststrom_Aufheben_0-18sec.sin
REFERENCE-SCENARIO:    Szenario_w5_2_Ueberlaststrom_Aufheben

SIMULATION-TIME[sec]  MESSAGE                                COMMENT
1.116528              [PRECONDITION: Zuendung == EIN]
1.751935              Stelle(LangsamWischen)
1.752960              LiefereBetriebsmodus(LangsamWischen)
1.766016              StelleGeschwindigkeit(Langsam)
1.766996              AktiviereUeberlastDiagnose()
2.851900              Ueberstrom(An)
3.376000              StelleGeschwindigkeit(Aus)
3.379984              ZeigeWischerStatus(LangsamWischenDefekt)
10.751900             Ueberstrom(Aus)
13.351916             Stelle(Aus)
13.356992             LiefereBetriebsmodus(Aus)
13.401908             Stelle(LangsamWischen)                Warning: Message occurs too early!
13.402931             LiefereBetriebsmodus(LangsamWischen)  Warning: Message occurs too early!
14.301901             Stelle(Aus)                            Warning: Message occurs too early!
14.306955             LiefereBetriebsmodus(Aus)             Warning: Message occurs too early!
Error: Scenario not completed!

```

Abbildung 8.12: Prüfbericht für eine einzelne Konformitätsprüfung

Bei der Einzelprüfung wird im Fehlerfall das Prüfergebnis auch grafisch anhand des Anwendungsszenarios visualisiert. Dabei werden nicht aufgetretene Nachrichten oder Nachrichten, bei denen die Zeitbedingungen verletzt sind, im grafischen Sequenzdiagramm-Editor farblich hinterlegt angezeigt. Abbildung 8.13 zeigt ein Beispiel für die grafische Fehlervisualisierung. In diesem Fall war die Zeitbedingung für die Nachricht „StelleGeschwindigkeit(Aus)“ im Anwendungsszenario „W2_1 Scheibe Schnell Wischen“ überschritten, was durch die farbliche Hinterlegung der Nachricht im Sequenzdiagramm gekennzeichnet ist.

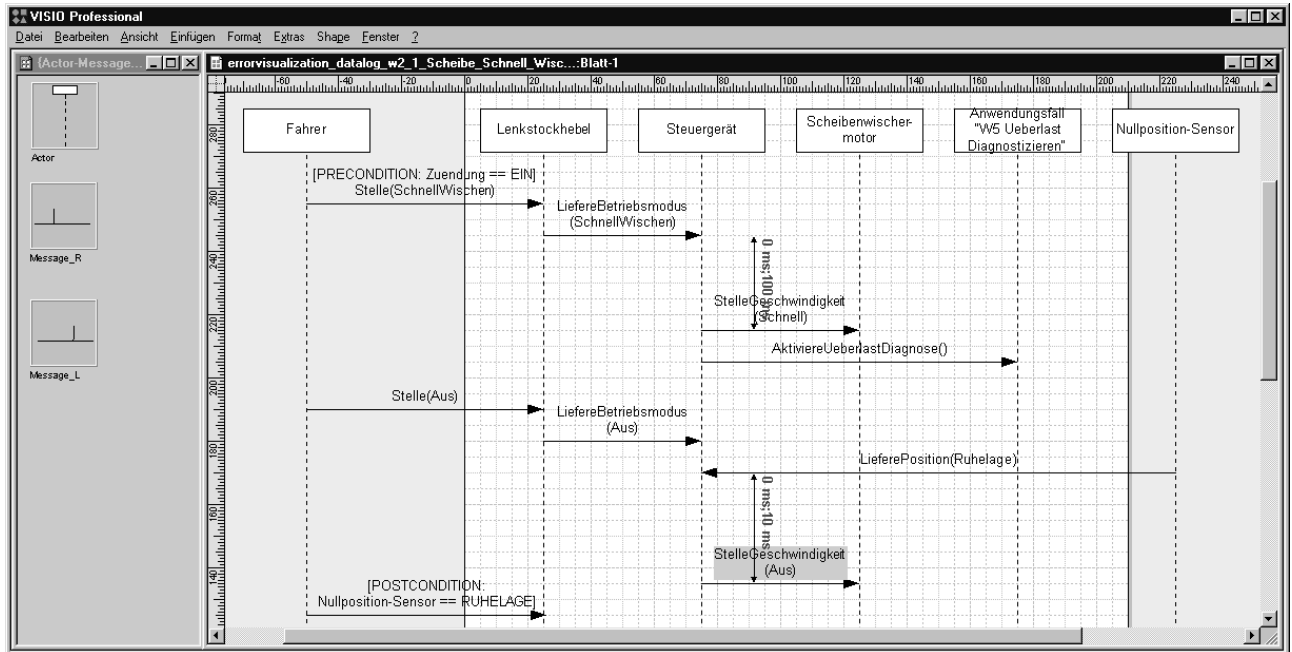


Abbildung 8.13: Fehlervisualisierung im geprüften Anwendungsszenario

Wird ein gesamtes Validierungs-Projekt geprüft, so wird zusätzlich zu den Einzel-Prüfberichten ein Gesamtergebnis-Prüfbericht generiert. Abbildung 8.14 zeigt einen Auszug aus dem Validierungs-Projekt-Prüfbericht.

```

AVE - Automated Validation Environment 3.0.2 - AVE1
Validation Project Manager Use Case Editor Sequence Diagram Editor Stimuli Generator Simulation Trace Filter View Window Help

validation_report.txt - Editor
Datei Bearbeiten Suchen ?

Validation Project Summary
-----
DB-PATH: c:\f1-doku\Anwendungsbeispiele\010311-Wischeranlage

No 1
TIME-STAMP-SEQUENCE: datalog_w1_1_Scheibe_Langsam_Wischen_0-15sec.sim
RELATED-SCENARIO: Szenario_w1_1_Scheibe_Langsam_Wischen
VALIDATION-RESULTS:
Szenario_w1_1_Scheibe_Langsam_Wischen validated
Szenario_w4_1_Scheibe_Tipp_Wischen validated
Szenario_w5_1_Ueberlaststrom_Diagnostizieren validated

No 2
TIME-STAMP-SEQUENCE: datalog_w1_2_Scheibe_L_und_S_Wischen_30-44sec_result.txt
RELATED-SCENARIO: Szenario_w1_2_Scheibe_L_und_S_Wischen
VALIDATION-RESULTS:
Szenario_w1_2_Scheibe_L_und_S_Wischen validated
Szenario_w4_1_Scheibe_Tipp_Wischen validated
Szenario_w5_1_Ueberlaststrom_Diagnostizieren validated

No 3
TIME-STAMP-SEQUENCE: datalog_w2_1_Scheibe_Schnell_Wischen_15-29sec_result.txt
RELATED-SCENARIO: Szenario_w2_1_Scheibe_Schnell_Wischen
VALIDATION-RESULTS:
Szenario_w2_1_Scheibe_Schnell_Wischen error
Szenario_w4_1_Scheibe_Tipp_Wischen validated
Szenario_w5_1_Ueberlaststrom_Diagnostizieren validated

No 4
TIME-STAMP-SEQUENCE: datalog_w3_1_Scheibe_Automatisch_Langsam_Wischen_45-59sec.sim
RELATED-SCENARIO: Szenario_w3_1_Scheibe_Automatisch_Langsam_Wischen

Ready NUM

```

Abbildung 8.14: Gesamtergebnis-Prüfbericht für ein gesamtes Validierungs-Projekt

Im Gesamtergebnis-Prüfbericht werden alle einzelnen Prüfergebnisse nach Zeitstempel-Sequenzen geordnet aufgelistet. Zu jeder Zeitstempel-Sequenz wird neben dem zugehörigen Anwendungsszenario auch das Ergebnis der Prüfung gegen alle zwingenden und verbotenen Anwendungsszenarios des Validierungs-Projekts aufgelistet.

8.11 Bewertung der Werkzeugunterstützung

Die einzelnen Schritte des Prüfverfahrens zur Validierung komponentenbasierter Software für Echtzeitsysteme sind in der realisierten Validierungsumgebung weitestgehend automatisiert. Der Schwerpunkt der Werkzeugunterstützung liegt dabei auf der systematischen Herleitung von Prüffällen aus den Anwendungsfällen und auf der Konformitätsprüfung des simulierten Verhaltens gegenüber dem in den Anwendungsszenarios spezifizierten Verhalten.

Für diejenigen Schritte des Prüfverfahrens, die prinzipiell nicht voll automatisiert werden können und deshalb manuell vom Entwickler bzw. Prüfer vorgenommen werden müssen, bietet die Validierungsumgebung unterstützende Editoren an. Die Formalisierung der mit dem Anwendungsfall-Editor textuell spezifizierten Anwendungsfälle in prüfbare Anwendungsszenarios wird dabei durch den Sequenzdiagramm-Editor unterstützt. Auch die Zuordnung der Nachrichten-Bezeichner aus den Anwendungsszenarios auf die Nachrichten-Bezeichner im Simulationsmodell erfolgt manuell mit Hilfe des Zuordnungstabellen-Editors. Der Stimuli-Generator generiert zwar anhand einfacher Abbildungsvorschriften Stimuli-Sequenzen aus den Anwendungsszenarios, jedoch sind zur Vervollständigung der einzelnen Stimuli-Sequenzen tiefergehende Kenntnisse über den Aufbau des gesamten Simulationsmodells erforderlich und können daher ebenfalls nur mit dem Wissen aus der Simulationsmodellerstellung durchgeführt werden. Alle anderen Schritte des Prüfverfahrens sind durch die Werkzeuge der Validierungsumgebung voll automatisiert.

Für die zur Simulation verwendeten Simulationswerkzeuge ASCET-SD und CANoe wurden in der Validierungsumgebung spezifische Schnittstellen für die Stimulations-Daten der Stimuli-Sequenzen und für die Zeitstempel-Aufzeichnungen der Zeitstempel-Sequenzen entwickelt. Durch Anpassung dieser Schnittstellen kann die Validierungsumgebung mit wenig Aufwand für den Einsatz mit anderen Simulationswerkzeugen erweitert werden, sofern diese Simulationswerkzeuge über offene Schnittstellen für Stimulations-Daten und Zeitstempel-Aufzeichnungen verfügen. Das Starten und Stoppen der Simulationsausführung ausgewählter Prüffälle in ASCET-SD und CANoe erfolgt derzeit ebenfalls noch manuell. In Anlehnung an verfügbare Lösungen aus dem Bereich der Testautomatisierung könnte eine Fernsteuerung der Simulationsablaufsteuerung auch diesen Schritt voll automatisieren.

Erste praktische Erfahrungen [Lind00, Romm01] mit der prototypischen Validierungsumgebung haben gezeigt, dass sie den Entwickler bei der Spezifikation von prüfbaren Anwendungsfällen und bei der systematischen Entwicklung von Prüffällen sehr gut unterstützt. Die voll

automatisierte Konformitätsprüfung ermöglicht effiziente Regressionsprüfungen von bereits erstellten Prüffällen.

Um die Leistungsfähigkeit des Prüfverfahrens und der entwickelten Softwarewerkzeuge der Validierungsumgebung nachzuweisen, werden sie im folgenden Kapitel an einem Fallbeispiel angewendet.

9 Anwendung des Prüfverfahrens zur Validierung der komponentenbasierten Software einer Kfz-Scheibenwischeranlage

In diesem Kapitel wird die Anwendung des Prüfverfahrens zur Validierung am Beispiel der Entwicklung einer komponentenbasierten Software für das Steuergerät einer modernen Scheibenwischeranlage im Kfz gezeigt.

9.1 Übersicht über die Scheibenwischeranlage

Abbildung 9.1 zeigt den Labor-Aufbau einer Scheibenwischeranlage im Modellprozess IAS-Cockpit.

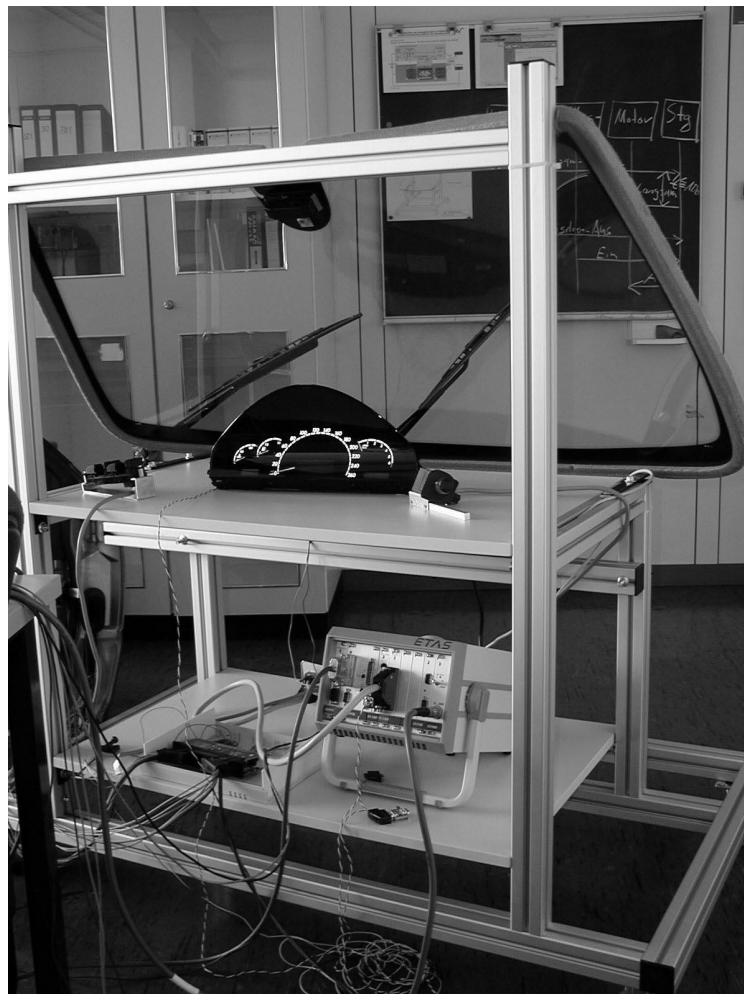


Abbildung 9.1: Scheibenwischeranlage im Modellprozess IAS-Cockpit

Die Scheibenwischeranlage wurde als Anwendungsbeispiel ausgewählt, weil sie ein einfaches und überschaubares, aber auch typisches Echtzeitsystem repräsentiert. Scheibenwischeranlagen im Kfz haben die Aufgabe, die gesetzliche Forderung nach stets ausreichender Rundumsicht zu erfüllen [Bosc99]. Daraus leiten sich einige spezifische Echtzeitanforderungen ab. Das Abstellen des Scheibenwischermotors muss beispielsweise innerhalb einer definierten Zeitspanne erfolgen, damit die Wischerblätter in der Parkposition stoppen und nicht das Sichtfeld des Fahrers stören. Ebenso darf keine für den Fahrer wahrnehmbare Ein- oder Ausschaltverzögerung beim Betätigen des Lenkstockhebels existieren. Weitere Schwierigkeiten können durch unterschiedliche Eingabewerte der konkurrierenden Sensoren entstehen.

Abbildung 9.2 zeigt die wesentlichen Hardware-Komponenten der Scheibenwischeranlage.

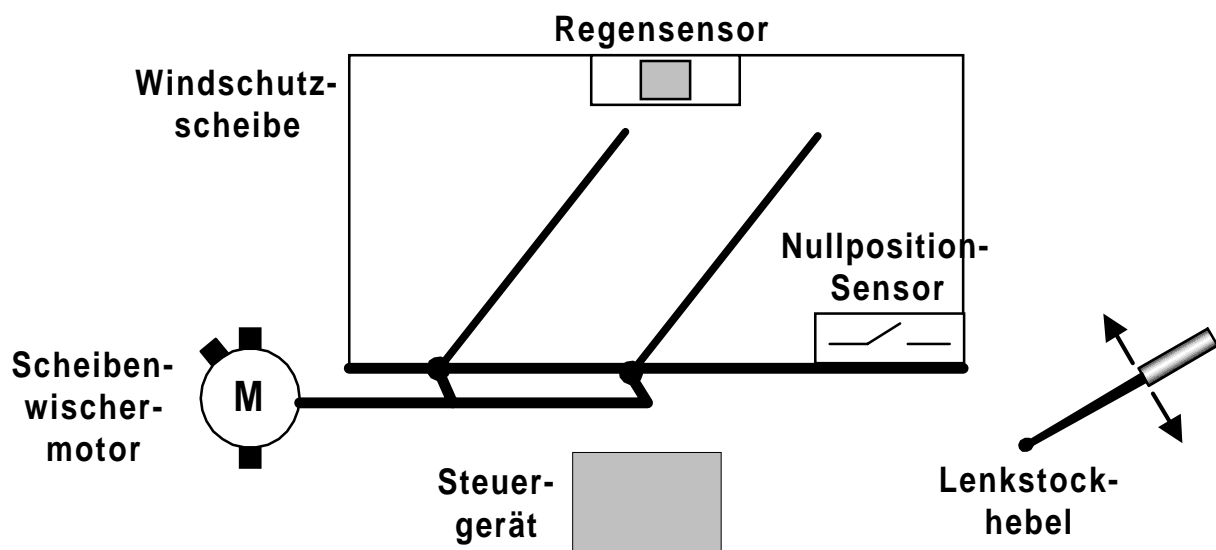


Abbildung 9.2: Hardware-Komponenten der Scheibenwischeranlage

Das elektronische Steuergerät ist mit den Sensoren Lenkstockhebel, Regensensor und Nullposition-Sensor sowie mit dem Aktor Scheibenwischermotor elektrisch verbunden. Außerdem enthält es einen Überstrom-Sensor zur Erkennung von mechanischen Blockierungen des Scheibenwischermotors.

Mit dem Lenkstockhebel kann der Fahrer die Betriebsmodi „Tipp-Wischen“, „Aus“, „Automatik-Wischen“, „Langsam Wischen“ und „Schnell Wischen“ auswählen. Der Nullposition-Sensor dient zur Feststellung der Parklage für die Wischerblätter auf der Windschutzscheibe und ist im Scheibenwischermotor integriert. Der Regensensor erkennt die Wassermenge auf der Windschutzscheibe. Er liefert für das „Automatik-Wischen“ die Informationen über die Regenstärke. Die vom Regensensor gelieferten Informationen über die Regenstärke dienen in modernen Karosserieelektroniksystemen nicht nur der Scheibenwischeranlage, sondern werden auch von anderen Software-Komponenten für übergeordnete Komfortfunktionen genutzt. Bei abgestelltem Fahrzeug kann z.B. die vom Regensensor ermittelte Regen-

stärke zum Witterungsschutz des Fahrzeuginnenraums durch ein automatisches Schließen des Schiebedachs bei einsetzendem Regen genutzt werden.

9.2 Anwendungsfälle

Abbildung 9.3 zeigt in einem UML Anwendungsfall-Diagramm die bei der Anforderungsanalyse identifizierten 5 Anwendungsfälle der Scheibenwischeranlage und ihre Beziehungen untereinander.

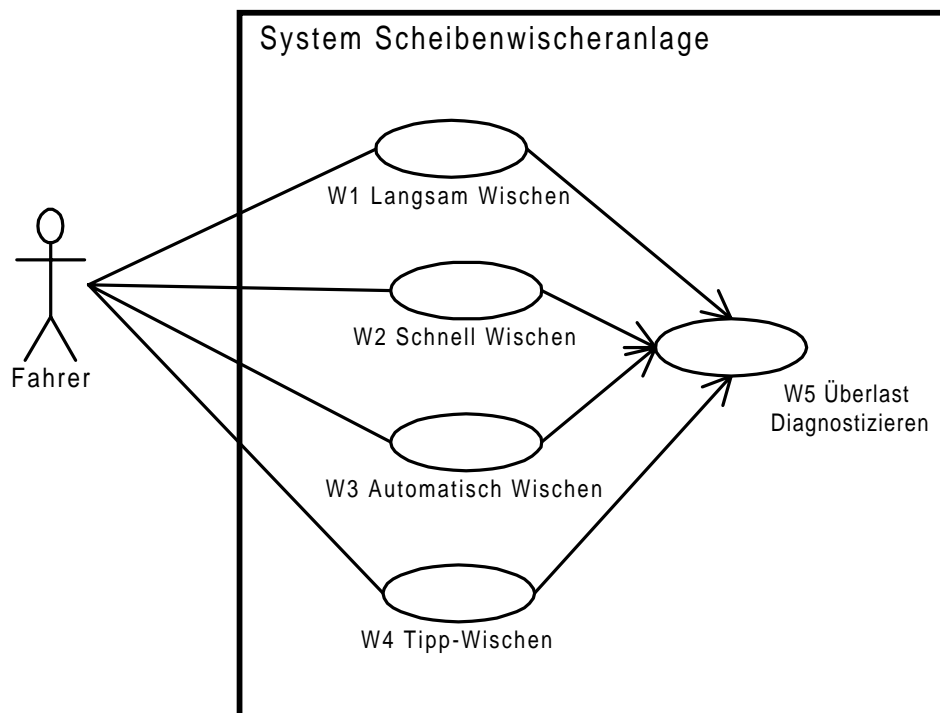


Abbildung 9.3: Anwendungsfall-Diagramm für die Scheibenwischeranlage

Exemplarisch ist in Abbildung 9.4 der einfache Anwendungsfall „W2 Schnell Wischen“ dargestellt. Der Anwendungsfall besteht aus einem Hauptszenario und möglichen Variationen. Die Vorbedingung, dass die Zündung eingeschaltet sein muss, ist charakteristisch für Anwendungen im Bereich der Karosserieelektronik. Das Ziel des Anwendungsfalls ist erreicht, wenn sich die Wischerblätter in der Parkposition befinden und damit die Nachbedingung erfüllt ist. In den einzelnen Schritten des Hauptszenarios sind neben dem spezifizierten funktionalen Ablauf zum Teil auch Anforderungen an das zeitliche Verhalten spezifiziert. In Schritt 1.3 des Hauptszenarios wird außerdem der Anwendungsfall „W5 Überlast Diagnostizieren“ aktiviert.

Anwendungsfall	W2 Schnell Wischen	
Ziel	Scheibe von viel Regenwasser befreien	
Vorbedingung	Zündung ein (Klemme 15 „Ein“)	
Nachbedingung	Wischerblätter in Parkposition	
Akteure	Fahrer, Lenkstockhebel, Steuergerät, Scheibenwischermotor, Nullposition-Sensor	
Hauptszenario	Schritt	Aktion
	1.0	Fahrer stellt Lenkstockhebel auf Position „Schnell Wischen“
	1.1	Lenkstockhebel liefert dem Steuergerät den Betriebsmodus „Schnell Wischen“
	1.2	Steuergerät startet Scheibenwischermotor mit Wischgeschwindigkeit „Schnell“ nach max. 100 ms
	1.3	Steuergerät aktiviert die kontinuierliche Überlast-Diagnose (→ Anwendungsfall „W5 Überlast Diagnostizieren“)
	1.4	Fahrer stellt Lenkstockhebel auf Position „Aus“
	1.5	Lenkstockhebel liefert dem Steuergerät den Betriebsmodus „Aus“
	1.6	Nullposition-Sensor liefert dem Steuergerät die Position „Ruhelage“
Variationen	1.7	Steuergerät stoppt den Scheibenwischermotor nach max. 10ms
	1.4a	Fahrer stellt Lenkstockhebel auf Position „Langsam Wischen“. Fortfahren bei (→ Anwendungsfall „W1 Langsam Wischen“).
	1.4b	Fahrer stellt Lenkstockhebel auf Position „Automatisch Wischen“. Fortfahren bei (→ Anwendungsfall „W3 Automatisch Wischen“).
Ausnahmen	- keine -	
Anmerkungen	- keine -	

Abbildung 9.4: Anwendungsfall „W2 Schnell Wischen“

9.3 Anwendungsszenarios

Aus den 5 Anwendungsfällen der Scheibenwischeranlage wurden insgesamt 9 Anwendungsszenarios entwickelt. Tabelle 9.1 gibt eine Übersicht über die entwickelten Anwendungsszenarios der Scheibenwischeranlage.

Tabelle 9.1: Anwendungsszenarios für die Scheibenwischeranlage

Anwendungsszenario	Sequenztyp	Kommentar
W1_1 Scheibe Langsam Wischen	normal	
W1_2 Scheibe Langsam und Schnell Wischen	normal	Variation: Lenkstockhebel von „Aus“ auf „Langsam“ und anschließend auf „Schnell“ umschalten.
W2_1 Scheibe Schnell Wischen	normal	
W3_1 Scheibe Automatisch Langsam Wischen	normal	
W3_2 Scheibe Automatisch Schnell Wischen	normal	
W3_3 Scheibe Automatisch Langsam und Schnell Wischen	normal	Variation: Regensensor von „Kein Wasser“ auf „Wenig Wasser“ und anschließend auf „Viel Wasser“ umschalten.
W4_1 Scheibe Tipp-Wischen	zwingend	
W5_1 Überlaststrom Diagnostizieren	zwingend	
W5_2 Überlaststrom Aufheben	normal	Variation: Bei diagnostiziertem Überlaststrom am Überstrom-Sensor anschließend den Lenkstockhebel durch den Fahrer zurücksetzen.

Wie an der Nummerierung der Anwendungsszenarios zu erkennen ist, wurden sie aus den Hauptszenarios (W_{x_1}) sowie aus den alternativen Szenarios und Variationen (W_{x_2} - W_{x_n}) der Anwendungsfälle hergeleitet. Für fast alle Anwendungsszenarios wurde der Sequenztyp mit „normal“ spezifiziert, da Teile der Nachrichten-Sequenzen auch in anderen Anwendungsszenarios mit beispielsweise veränderten Echtzeitanforderungen spezifiziert sind. Nur für die Anwendungsszenarios „W4_1 Scheibe Tipp-Wischen“ und „W5_1 Überlaststrom Diagnostizieren“ wurde der Sequenztyp „zwingend“ festgelegt, da diese Nachrichten-Sequenzen nach Erfüllung der spezifizierten Vorbedingung in jedem Fall bei der Simulation so ablaufen müssen. In Abbildung 9.5 ist exemplarisch das aus dem Hauptszenario des Anwendungsfalls „W2 Schnell Wischen“ hergeleitete Anwendungsszenario „W2_1 Scheibe Schnell Wischen“ dargestellt.

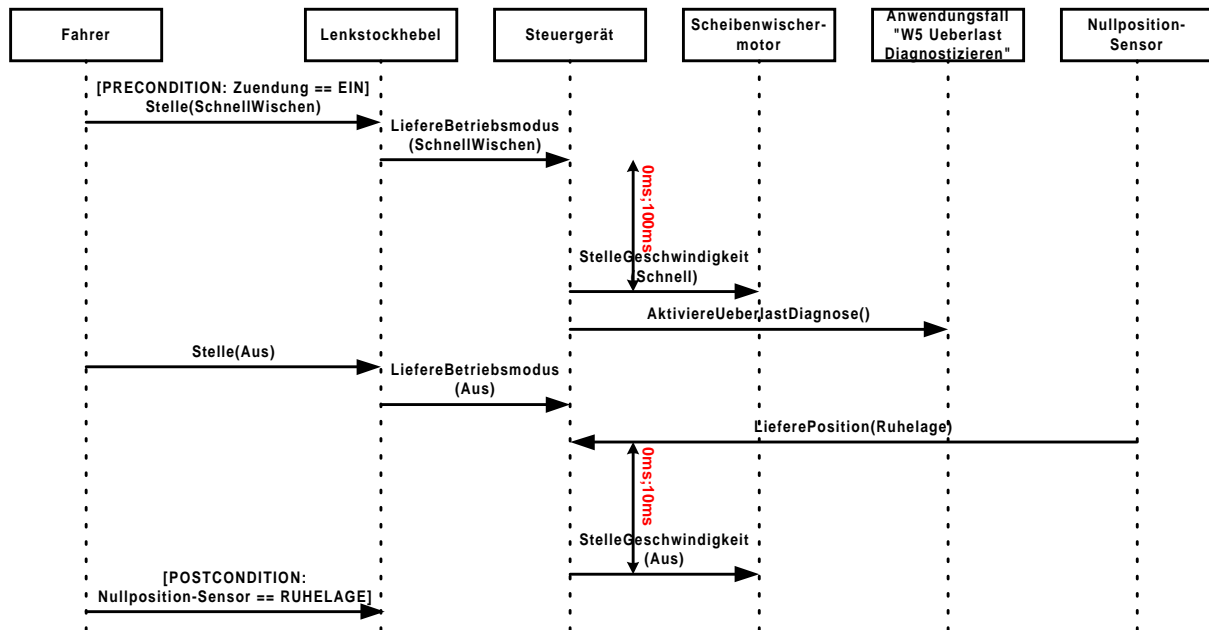


Abbildung 9.5: Anwendungsszenario „W2_1 Scheibe Schnell Wischen“

Die Sequenz von Nachrichten des Anwendungsszenarios spezifiziert den charakteristischen Ablauf der Scheibenwischeranlage vom Einschalten auf schnelle Geschwindigkeit bis hin zum Stoppen der Wischerblätter in der vorgesehenen Parkposition. Über die Pseudo-Nachricht mit der Nachbedingung am Ende der Sequenz wird explizit die Parkposition der Wischerblätter über den Zustand des Nullposition-Sensors gefordert. Da der Ablauf des Anwendungsszenarios durch andere Bedieneingriffe des Fahrers im zweiten Teil der Sequenz auch variieren kann, wurde der Sequenztyp auf „normal“ festgelegt. Das bedeutet, dass dieses Anwendungsszenario nur dann so ablaufen muss, wenn es gezielt so stimuliert wird.

9.4 Simulationsmodell des Echtzeitsystems Scheibenwischeranlage

Unter Berücksichtigung der Anwendungsfälle wurde ein komponentenbasiertes Entwurfsmodell für die Software der Scheibenwischeranlage entworfen und mit dem Werkzeug ASCET-SD modelliert. Aus dem komponentenbasierten Entwurfsmodell der Software wurden mit Hilfe von ASCET-SD vollautomatisch die Komponenten für das Simulationsmodell generiert. Diese wurden noch um ein Modell der Umgebung erweitert, welches das Verhalten des technischen Prozesses in der Simulation nachbildet. Abbildung 9.6 zeigt die Komponenten der Anwendungssoftware und das Modell der Umgebung (engl. environment model) im Simulationsmodell.

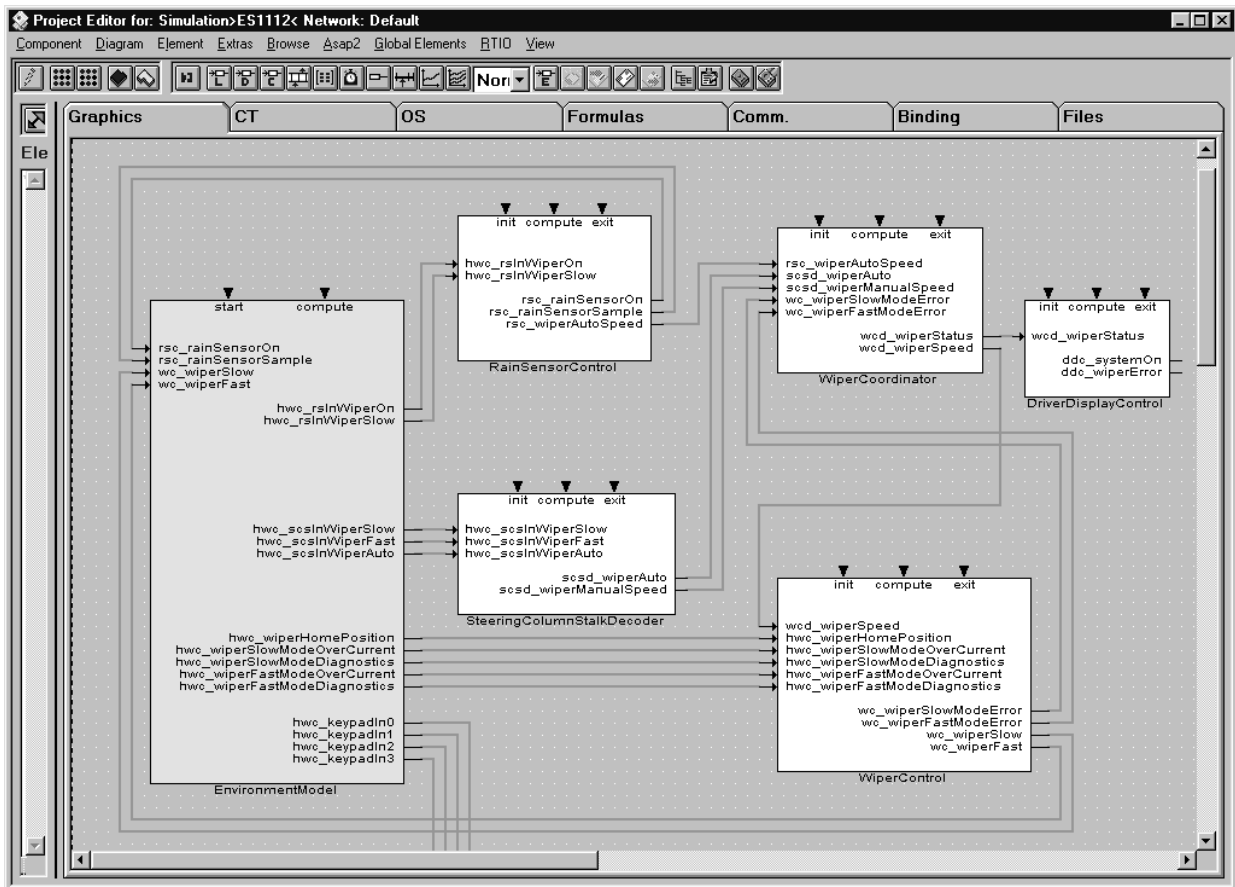


Abbildung 9.6: Simulationsmodell der Scheibenwischeranlage in ASCET-SD

Das hellgrau dargestellte Modell der Umgebung kapselt als Baugruppe einfache Simulationsmodell-Komponenten für die elektromechanischen Sensoren und Aktoren der Scheibenwischeranlage. Dazu gehört unter anderem ein einfaches Streckenmodell, das sich aus den Simulationsmodell-Komponenten für den Scheibenwischermotor und den Nullposition-Sensor zusammensetzt.

Die weißen Komponenten des Simulationsmodells sind die Komponenten des komponentenbasierten Entwurfsmodells der Anwendungssoftware für die Scheibenwischeranlage.

Die Stimuli-Generator-Komponente, mit denen die Stimuli-Sequenzen während der Simulationsausführungen generiert werden, befinden sich ebenfalls innerhalb der hellgrauen Baugruppe „environment model“.

9.5 Stimuli-Sequenzen

Bei der systematischen Entwicklung der Prüffälle wurde für jedes Anwendungsszenario eine Stimuli-Sequenz zur Stimulation des Simulationsmodells der Scheibenwischeranlage abgeleitet. Abbildung 9.7 zeigt als Beispiel einen Teil der vom Anwendungsszenario „W2_1 Scheibe Schnell Wischen“ abgeleiteten Stimuli-Sequenz.

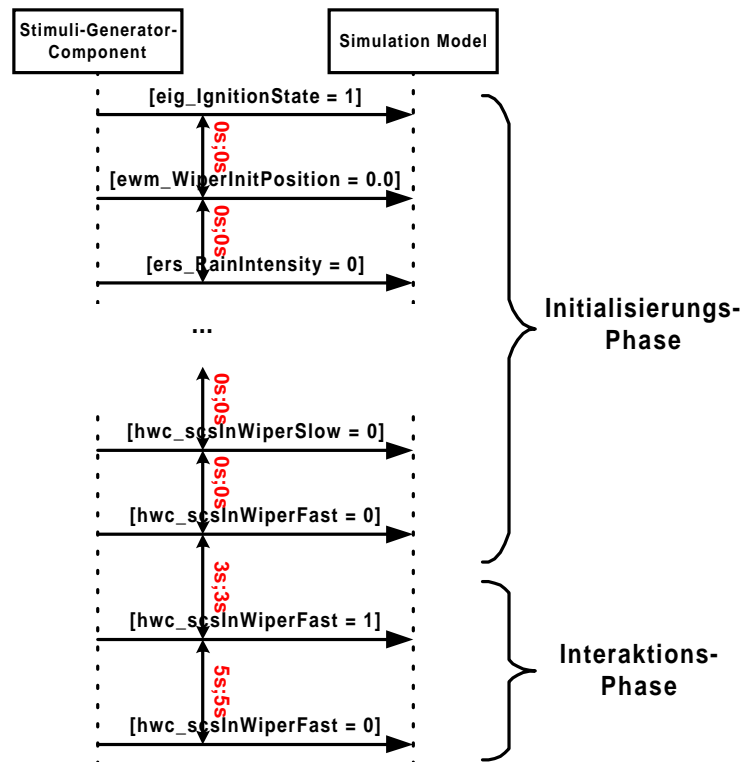


Abbildung 9.7: Stimuli-Sequenz zum Anwendungsszenario „W2_1 Scheibe Schnell Wischen“

Die Initialisierungs-Phase initialisiert mit ihren Nachrichten alle notwendigen Systemvariablen im Modell der Umgebung und ist bei allen 9 abgeleiteten Stimuli-Sequenzen identisch. Die sehr einfache Sequenz von zwei Nachrichten in der Interaktions-Phase stimuliert gezielt den Ablauf des zugehörigen Anwendungsszenarios „W2_1 Scheibe Schnell Wischen“. Die erste Nachricht stimuliert das Umschalten der Simulationsmodell-Komponente des elektromechanischen Lenkstockhebels durch den Fahrer von „Aus“ auf „Schnell Wischen“. Mit der zweiten Nachricht wird der Lenkstockhebel wieder in den Zustand „Aus“ zurückgesetzt.

9.6 Prüfergebnisse

Mit den abgeleiteten Stimuli-Sequenzen wurden alle 9 verschiedenen Anwendungsszenarios bei der Simulationsausführung sequenziell stimuliert und das simulierte Verhalten in 9 Zeitstempel-Sequenzen aufgezeichnet. Diese 9 Zeitstempel-Sequenzen wurden jeweils gegen das zugehörige Anwendungsszenario und gegen die zwei Anwendungsszenarios vom Sequenztyp „zwingend“ unter Verwendung der Validierungsumgebung AVE geprüft. Die Prüfung des gesamten Validierungs-Projekts „Scheibenwischeranlage“ umfasste damit insgesamt 25 (= 9 + 8 + 8) einzelne Konformitätsprüfungen. Die Prüfergebnisse für die ersten Simulationsaufzeichnungen sind in Tabelle 9.2 aufgeführt.

Tabelle 9.2: Prüfergebnisse des Validierungs-Projekts „Scheibenwischeranlage“

		Prüferferenz: Anwendungsszenario		
		Zugehöriges Anwendungs- szenario	W4_1 (Sequenztyp „zwingend“)	W5_1 (Sequenztyp „zwingend“)
Zeitstempel-Sequenz zum stimulierten Anwendungsszenario	W1_1 Scheibe Langsam Wischen	v	v	v
	W1_2 Scheibe Langsam und Schnell Wischen	v	v	v
	W2_1 Scheibe Schnell Wischen	f	v	v
	W3_1 Scheibe Automatisch Langsam Wischen	v	v	v
	W3_2 Scheibe Automatisch Schnell Wischen	f	v	v
	W3_3 Scheibe Automatisch Langsam und Schnell Wischen	v	v	v
	W4_1 Scheibe Tipp-Wischen	v	-	v
	W5_1 Überlaststrom Diagnostizieren	v	v	-
	W5_2 Überlaststrom Aufheben	f	v	v

(v = validiert, f = fehlerhaft, - = redundant)

Neben den 22 validierten Prüfergebnissen resultierten bei den ersten Simulationsaufzeichnungen 3 fehlerhafte Prüfergebnisse für die aufgezeichneten Zeitstempel-Sequenzen. Die Ursachen für die Fehler konnten anhand der protokollierten Einzel-Prüfergebnisse schnell lokalisiert werden.

Abbildung 9.8 zeigt das fehlerhafte Prüfergebnis der zum Anwendungsszenario „W2_1 Scheibe Schnell Wischen“ zugehörigen Zeitstempel-Sequenz.

```

Scenario Validation Result
-----
TIME-STAMP-SEQUENCE:   datalog_w2_1_Scheibe_Schnell_Wischen_15-29sec.sim
REFERENCE-SCENARIO:    Szenario_w2_1_Scheibe_Schnell_Wischen

SIMULATION-TIME[sec]  MESSAGE                                COMMENT
15.126528             [PRECONDITION: Zuendung == EIN]
19.541569             Stelle(SchnellWischen)
19.542591             LiefereBetriebsmodus(SchnellWischen)
19.582592             StelleGeschwindigkeit(Schnell)
19.583551             AktiviereUeberlast()
26.443584             Stelle(Aus)
26.452608             LiefereBetriebsmodus(Aus)
26.585535             LieferePosition(Ruhelage)
26.602560             StelleGeschwindigkeit(Aus)           Error: Timing condition (Maximum) violated!
26.602560             [POSTCONDITION: Nullposition-Sensor == RUHELAGE]
Scenario completed!

```

Abbildung 9.8: Fehlerhaftes Prüfergebnis der zugehörigen Zeitstempel-Sequenz zum Anwendungsszenario „W2_1 Scheibe Schnell Wischen“

Das Anwendungsszenario „W2_1 Scheibe Schnell Wischen“ war zwar mit allen geforderten Nachrichten und Bedingungen in der richtigen Reihenfolge vollständig in der Zeitstempel-Sequenz enthalten, aber die Nachricht „StelleGeschwindigkeit(Aus)“ verletzte in der Simulation die geforderte maximale Zeitbedingung zur Vorgänger-Nachricht. Im Anwendungsszenario wurde die Zeitbedingung durch ein Intervall zwischen 0 ms und 10 ms spezifiziert (siehe Abbildung 9.5). In der Simulationsaufzeichnung existierte zwischen den beiden Nachrichten jedoch eine Zeitdifferenz von 17,025 ms, was zu der Fehlermeldung im Prüfergebnis führte. Die Fehlermeldung trat nur bei schneller Geschwindigkeit des Scheibenwischermotors auf, was auch das zweite fehlerhafte Prüfergebnis des Anwendungsszenarios „W3_2 Scheibe Automatisch Schnell Wischen“ erklärt. In der Realität würde durch diese Verletzung der Echtzeitanforderung der Wischermotor zu spät abgeschaltet, so dass die Wischerblätter nicht exakt in der vorgesehenen Parkposition zum Stillstand kommen. Sie könnten dadurch die Sicht des Fahrers behindern. Die Fehlerursache lag in einer falschen Konfigurierung der Echtzeitbetriebssystem-Komponente. Durch die Änderung eines Konfigurierungs-Parameters der Echtzeitbetriebssystem-Komponente konnte dieser Entwurfsfehler behoben werden.

Das dritte fehlerhafte Prüfergebnis führte zu einem rein funktionalen Entwurfsfehler, der bei der Konformitätsprüfung der Zeitstempel-Sequenz gegenüber dem zugehörigen Anwendungsszenario „W5_2 Überlaststrom Aufheben“ aufgedeckt wurde. Abbildung 9.9 zeigt das fehlerhafte Prüfergebnis.

```

Scenario Validation Result
-----
TIME-STAMP-SEQUENCE:   datalog_w5_2_Ueberlaststrom_Aufheben_0-18sec.sim
REFERENCE-SCENARIO:    Szenario_w5_2_Ueberlaststrom_Aufheben

SIMULATION-TIME[sec]  MESSAGE                                     COMMENT
1.116528              [PRECONDITION: Zuendung == EIN]
1.751935              Stelle(LangsamWischen)
1.752960              LiefereBetriebsmodus(LangsamWischen)
1.766016              StelleGeschwindigkeit(Langsam)
1.766996              AktiviereUeberlastDiagnose()
2.851900              Ueberstrom(An)
3.376000              StelleGeschwindigkeit(Aus)
3.379984              ZeigeWischerStatus(LangsamWischenDefekt)
10.751900             Ueberstrom(Aus)
13.351916             Stelle(Aus)
13.356992             LiefereBetriebsmodus(Aus)
13.401908             Stelle(LangsamWischen)                Warning: Message occurs too early!
13.402931             LiefereBetriebsmodus(LangsamWischen)  Warning: Message occurs too early!
14.301901             Stelle(Aus)                            Warning: Message occurs too early!
14.306955             LiefereBetriebsmodus(Aus)            Warning: Message occurs too early!
Error: Scenario not completed!

```

Abbildung 9.9: Fehlerhaftes Prüfergebnis der zugehörigen Zeitstempel-Sequenz zum Anwendungsszenario „W5_2 Überlaststrom Aufheben“

Die Fehlermeldung, dass das Anwendungsszenario nicht vollständig in der aufgezeichneten Zeitstempel-Sequenz enthalten ist, und die Warnmeldungen, dass Nachrichten zu früh in der Sequenz enthalten sind, lassen darauf schließen, dass eine oder mehrere Nachrichten nach der letzten korrekt aufgefundenen Nachricht ausblieben. Damit das im Anwendungsszenario „W5_2 Überlaststrom Aufheben“ geforderte Verhalten validiert werden kann, wurde in der abgeleiteten Stimuli-Sequenz die Systemvariable für einen anliegenden Überstrom bewusst länger als die tolerierte Zeitspanne von 500 ms angelegt. Die in der Software enthaltene Überlaststrom-Diagnose reagierte darauf zunächst auch korrekt mit der Abschaltung des Wischermotors durch „StelleGeschwindigkeit(Aus)“ zur Simulationszeit 3.37600 Sekunden. Mit dem Zurücksetzen der Systemvariable für den Überstrom wurde dann das Zurücksetzen des Lenkstockhebels durch den Fahrer durch „Stelle(Aus)“ stimuliert. Beim stimulierten Wiedereinschalten des Scheibenwischermotors mit „Stelle(LangsamWischen)“ durch den Fahrer zeigte sich jedoch eine nach dem Abschalten des Scheibenwischermotors entstandene Deadlock-Situation zwischen den zwei Komponenten „WiperCoordinator“ und „WiperControl“ der Anwendungssoftware. Ab dieser Nachricht erfüllte das simulierte Echtzeitsystem nicht mehr das geforderte Anwendungsszenario. Die Fehlerursache lag in der falschen Auswahl der Komponente „WiperCoordinator“. Durch den Austausch der Komponente durch eine andere Komponente mit geeignetem inneren Verhalten wurde dieser Entwurfsfehler behoben.

Nach der Behebung der aufgedeckten Entwurfsfehler und erneuter Simulation lieferten alle 25 Einzelprüfungen gültige „validierte“ Prüfergebnisse.

9.7 Zusammenfassung der Ergebnisse für das Anwendungsbeispiel

In diesem Kapitel wurde die Anwendung des Prüfverfahrens zur frühen Validierung des dynamischen Verhaltens der komponentenbasierten Software einer Kfz-Scheibenwischeranlage gezeigt. Zunächst wurden aus den informell spezifizierten Anwendungsfällen systematisch formalisierte Anwendungsszenarios entwickelt. Das in den Anwendungsszenarios spezifizierte Verhalten wurde mit Hilfe der Validierungsumgebung AVE gegenüber dem in der Simulation aufgezeichneten Verhalten des komponentenbasierten Entwurfsmodells der Software geprüft. Die Konformitätsprüfungen der in der Simulation aufgezeichneten Zeitstempel-Sequenzen gegenüber den Anwendungsszenarios deckten zwei grundlegende funktionale Entwurfsfehler in der komponentenbasierten Software für die Scheibenwischeranlage auf. Durch den Einsatz von Echtzeitsimulation konnte ein weiterer Entwurfsfehler im zeitlichen Verhalten der komponentenbasierten Software bereits während der Entwurfsphase aufgedeckt werden. Nach der Behebung der Entwurfsfehler konnte das simulierte Verhalten gegenüber dem in den Anwendungsszenarios geforderten Verhalten vollständig validiert werden. Die später fehlerfreie Inbetriebnahme der komponentenbasierten Software im realen Modellprozess IAS-Cockpit [Lind00] bestätigte die Zuverlässigkeit der Prüfergebnisse aus der Validierung des simulierten Verhaltens.

Im nächsten Kapitel werden abschließend die Eigenschaften sowie die Vorteile und Grenzen des entwickelten Prüfverfahrens zur Validierung komponentenbasierter Software für Echtzeitsysteme zusammengefasst.

10 Schlussfolgerungen und Ausblick

10.1 Zusammenfassung der Ergebnisse

Im Rahmen dieser Arbeit wurde ein praxisnahes und anwenderorientiertes Prüfverfahren zur Validierung des dynamischen Verhaltens von komponentenbasierter Software für Echtzeitsysteme in der Entwurfsphase entwickelt. Das Prüfverfahren ermöglicht die Validierung des simulierten Verhaltens von komponentenbasierter Software für Echtzeitsysteme gegenüber den Anforderungen aus Anwendersicht.

Die Verwendung von Simulation zur Prüfung hat den Vorteil, dass bereits auf Basis des komponentenbasierten Entwurfsmodells der Software, ohne real vorhandenen technischen Prozess und Automatisierungssystem, geprüft werden kann. Das entwickelte Prüfverfahren nutzt dazu die besondere Eigenschaft komponentenbasierter Software, dass die beim Entwurf verwendeten Black-Box-Komponenten gezielt für die Mehrfachverwendung entwickelt wurden und deshalb deren Verhalten bereits beim Entwurf vollständig spezifiziert ist. Aus diesem Grund ist ein aus Komponenten zusammengesetztes komponentenbasiertes Entwurfsmodell bezüglich funktionalem und zeitlichem Verhalten ausreichend spezifiziert, um es gegenüber den Anforderungen mit Hilfe von Simulation validieren zu können.

Die Prüfung mit Hilfe von Simulation setzt ein Simulationsmodell des gesamten Echtzeitsystems voraus, das nicht nur das rein funktionale Verhalten, sondern auch das Echtzeitverhalten der komponentenbasierten Software und der nicht vorhandenen realen Umgebung realitätsnah nachbildet. Die Umgebung wird dazu mit einfachen Simulationsmodell-Komponenten zur Nachbildung des Verhaltens des technischen Prozesses und des Automatisierungssystems modelliert. Für die Erstellung derartiger Simulationsmodelle wurde ein Konzept auf Basis der kommerziellen Modellierungs- und Simulationswerkzeuge ASCET-SD und CANoe entwickelt, welches die Echtzeitsimulation von komponentenbasierter Software für einzelne und verteilte Steuergeräte ermöglicht.

In dem im Rahmen dieser Arbeit definierten Entwicklungsprozess für die komponentenbasierte Softwareentwicklung für Echtzeitsysteme werden die Anforderungen an das dynamische Verhalten aus Sicht der Anwender bei der Anforderungsanalyse mit Hilfe von Anwendungsfällen spezifiziert. Aus diesen Anwendungsfällen wird das Referenz-Verhalten für die Prüfung systematisch hergeleitet. Da die Anwendungsfälle bereits als Teilprodukt des Entwicklungsprozesses vorliegen, ist dies im Vergleich zu herkömmlichen Prüfverfahren mit weniger Aufwand möglich.

Um die in den spezifizierten Anwendungsfällen enthaltenen Anforderungen in eine formale,

automatisiert prüfbare Notation überführen zu können, wurden Erweiterungen in der Notation der UML Sequenzdiagramme vorgenommen. Die erweiterten Sequenzdiagramme bieten, neben den standardisierten Beschreibungsmitteln für Sequenzen von Nachrichten, zusätzliche Beschreibungsmittel zur vollständigen Überführung aller in den Anwendungsfällen spezifizierten Anforderungen an, wie z.B. auch zustandsbehaftete Vor- und Nachbedingungen. Um den speziellen Eigenschaften von Echtzeitsystemen gerecht zu werden, bieten sie außerdem umfangreiche Beschreibungsmittel zur Spezifikation von Echtzeitanforderungen und zur exakten Festlegung des Geltungsbereichs bei der Prüfung. Zur Vorbereitung der Prüfung wird mit Hilfe der erweiterten Sequenzdiagramme aus einer Menge von Anwendungsfällen eine Menge von Anwendungsszenarios hergeleitet.

Die festgelegte Prüfstrategie ermöglicht die systematische Prüfung sämtlicher Anwendungsszenarios, die bei der Formalisierung der Anwendungsfälle hergeleitet wurden. Das bei der Simulationsausführung in Form von Zeitstempel-Sequenzen aufgezeichnete Verhalten der komponentenbasierten Software wird dazu anhand festgelegter Prüfkriterien gegenüber dem in den Anwendungsszenarios spezifizierten Referenz-Verhalten auf Konformität geprüft.

Durch die Verwendung von Echtzeitsimulation ist das simulierte Verhalten der komponentenbasierten Software und der Umgebung des Echtzeitsystems realitätsnah. Erste praktische Erfahrungen bei der Anwendung des Prüfverfahrens haben gezeigt, dass aufgedeckte Fehler im simulierten Verhalten der komponentenbasierten Software auch im realen Echtzeitsystem reproduzierbar auftreten.

Die prototypisch realisierte Validierungsumgebung unterstützt die einzelnen Schritte des Prüfverfahrens, angefangen von der systematischen Entwicklung der Anwendungsfälle und Anwendungsszenarios über die Generierung von Stimulations-Daten bis hin zur automatischen Prüfung und Generierung von Prüfergebnis-Berichten. Die Validierungsumgebung besitzt Schnittstellen zu den kommerziellen Modellierungs- und Simulationswerkzeugen ASCET-SD und CANoe. Für den komponentenbasierten Entwurf der Software wird stets ASCET-SD verwendet. Neben der sehr guten Entwurfsunterstützung für komponentenbasierte Software unterstützt ASCET-SD auch die voll automatische Generierung von Simulationsmodell-Komponenten aus dem komponentenbasierten Entwurfsmodell der Software und die Erstellung des Modells der Umgebung. Der für die Komponenten generierte Simulations-C-Code liegt dabei auch zur Verwendung in anderen Simulationswerkzeugen offen. Da ASCET-SD die Echtzeitsimulation von verteilten Steuergeräten nur unzureichend unterstützt, wurde für diesen Spezialfall eine Lösung auf Basis des Simulationswerkzeugs CANoe entwickelt. Simulationsmodelle für CAN-Bus-basierte Steuergeräteverbunde können dabei im Simulationswerkzeug CANoe erstellt werden. Die von ASCET-SD generierten Simulationsmodell-Komponenten der komponentenbasierten Software eines verteilten Echtzeitsystems werden bei dieser Lösung zur realitätsnahen Echtzeitsimulation ebenfalls in das CANoe Simulationsmodell mit eingebunden.

Die Kopplung der prototypisch realisierten Werkzeuge der Validierungsumgebung mit den professionellen CASE-Werkzeugen zeigt die Anwendbarkeit des Prüfverfahrens in der Praxis. Durch die Anpassung weniger Schnittstellen können die Werkzeuge der Validierungsumgebung leicht für den Einsatz mit anderen Simulationswerkzeugen bzw. kombinierten Entwurfs- und Simulationswerkzeugen angepasst werden.

10.2 Vorteile und Grenzen des entwickelten Prüfverfahrens

Bei herkömmlicher Softwareentwicklung entstehen in den frühen Entwicklungsphasen der Anforderungsanalyse und des Entwurfs der Großteil aller Fehler. Durch die Verwendung vorgefertigter Komponenten reduzieren sich zwar insgesamt die Fehlermöglichkeiten, aber die verbleibenden Fehlermöglichkeiten konzentrieren sich verstärkt auf die Tätigkeiten in der Entwurfsphase. Aus diesem Grund gewinnt die frühe Validierung von komponentenbasierter Software eine noch größere Bedeutung. Die Validierung der komponentenbasierten Software ermöglicht dabei den Nachweis von Anforderungen aus Sicht der Anwender und die Aufdeckung von Entwurfsfehlern.

Der für die frühe Validierung während der Entwurfsphase zusätzliche Aufwand ist kein wirklicher Mehraufwand, da spätestens in der Abnahmephase des entwickelten Echtzeitsystems die komponentenbasierte Software mit vergleichbaren Prüfungen validiert werden muss. Der Aufwand für die Validierung verlagert sich also nur von den späten in die frühen Entwicklungsphasen. Ein geringer Mehraufwand entsteht durch die Erstellung des Simulationsmodells und die Simulationsausführungen, wobei an dieser Stelle eine professionelle Werkzeugunterstützung, wie von den beiden integrierten Werkzeugen angeboten, den Aufwand in vertretbaren Grenzen hält. Neben den von ASCET-SD aus dem komponentenbasierten Entwurfsmodell voll automatisch generierten Simulationsmodell-Komponenten für die komponentenbasierte Software, muss dabei der nicht vorhandene technische Prozess durch zusätzliche Simulationsmodell-Komponenten nachgebildet werden. Für die Validierung von Steuerungssoftware für technische Prozesse mit überwiegend zustandsbehaftetem Verhalten, wie im Beispiel der Scheibenwischeranlage, reichen in der Regel sehr einfache Umgebungsmodelle aus. In der Praxis stehen hierzu häufig bereits fertige Simulationsmodelle aus vorangegangenen Machbarkeitsstudien oder der Funktionsentwicklung zur Verfügung, so dass die Simulationsmodell-Komponenten für das Modell der Umgebung meistens mit wenig Aufwand übernommen oder erstellt werden können.

Weniger gut geeignet ist das Prüfverfahren zur Validierung von Regelungsalgorithmen für regelungstechnische Prozesse mit überwiegend kontinuierlichen Systemgrößen, da in den Anwendungsfällen schwerpunktmäßig Anforderungen an Abläufe zwischen Komponenten spezifiziert werden. Für die Validierung der Regelung von kontinuierlichen Systemgrößen sind andere Beschreibungsmittel, wie z.B. mathematische Differenzialgleichungen, zur Spezifikation

von Anforderungen wesentlich besser geeignet. Gleichwohl erfolgt auch in diesem Fall die Validierung in der Praxis oftmals mit Hilfe von Simulation.

Der Mehraufwand für die Erstellung des Simulationsmodells und die Simulationsausführungen rechnet sich auf jeden Fall, da durch die frühe Aufdeckung von schwerwiegenden Entwurfsfehlern hohe Fehlerbeseitigungskosten von spät entdeckten Fehler eingespart werden. Außerdem ermöglicht die frühzeitige Validierung der komponentenbasierten Software eine zuverlässigere Planung von Entwicklungszeiten, da weniger „böse“ Überraschungen bei den Abnahmetests auftreten.

Auf die Validierung eines entwickelten Echtzeitsystems im Rahmen eines Abnahmetests kann dennoch nicht verzichtet werden, da bei der Verwendung von Simulationsmodellen das reale Verhalten nicht identisch nachgebildet wird. Insbesondere im Bezug auf das Echtzeitverhalten sind auch bei Echtzeitsimulation kleinere Abweichungen der Normalfall. Bei der Echtzeitsimulation von verteilten Steuergeräten mit CANoe ist außerdem eine große Rechenleistung des Entwicklungsrechners erforderlich, um das Echtzeitverhalten simulieren zu können. Die Erfahrungen mit dem Simulationswerkzeug CANoe haben gezeigt, dass bei einem Entwicklungsrechner der Größenordnung Pentium II / 233 MHz bereits bei der Simulation von 4 verteilten Steuergeräten mit ungefähr 30 Software-Komponenten die Grenzen der Echtzeitsimulation erreicht wurden.

Die für die frühe Validierung systematisch entwickelten Prüffälle können beim Abnahmetest wiederverwendet und am realen Automatisierungssystem wiederholt werden. Die Zeit für den bereits erfolgten Prüffallentwurf wird dabei in dieser späten Entwicklungsphase eingespart. Das entwickelte Prüfverfahren bildet damit eine nützliche Ergänzung zum herkömmlichen Abnahmetest ohne großen Mehraufwand zu verursachen.

10.3 Ausblick

Die einzelnen Schritte des im Rahmen dieser Arbeit entwickelten Prüfverfahrens sind in der prototypisch realisierten Validierungsumgebung weitestgehend automatisiert bzw. werden durch komfortable Editorwerkzeuge unterstützt. Die Steuerung der Simulationsabläufe in den angekoppelten Simulationswerkzeugen ASCET-SD und CANoe erfolgt derzeit noch manuell durch den Entwickler bzw. Prüfer der komponentenbasierten Software. Im Hinblick auf die Durchführung von Regressionsprüfungen beim praktischen Einsatz des Prüfverfahrens in größeren Entwicklungsprojekten würde sich an dieser Stelle eine Erweiterung der Validierungsumgebung um eine automatisierte Simulationsausführung anbieten. Der Aufwand für die Validierung komponentenbasierter Software für Echtzeitsysteme ließe sich dadurch nochmals reduzieren.

Neue Aufgabenstellungen für weiterführende Forschungsarbeiten ergeben sich beispielsweise

aus der Kombination des entwickelten Prüfverfahrens zur Validierung komponentenbasierter Software mit bestehenden Ansätzen zur Validierung regelungstechnischer Funktionen. Die Herausforderung liegt dabei in der Zusammenführung der unterschiedlichen Sichtweisen von Regelungstechnik und Softwaretechnik bei der Spezifikation von Anforderungen und deren Validierung für Echtzeitsysteme.

Literaturverzeichnis

- [AHP96] R. Alur, G. Holzmann, D. Peled: An Analyzer for Message Sequence Charts, *Software Concepts and Tools* (1996), 17 (2), S. 70-77
- [Balz99] Heide Balzert: Lehrbuch der Objektmodellierung: Analyse und Entwurf, Heidelberg, Spektrum, Akad. Verlag, 1999
- [Balz96] Helmut Balzert: Lehrbuch der Software-Technik: Software-Entwicklung, Heidelberg, Spektrum, Akad. Verlag, 1996
- [Balz98] Helmut Balzert: Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Heidelberg, Spektrum, Akad. Verlag, 1998
- [Bels97] R. Belschner: Ein Verfahren zur simulationsbasierten Analyse von verteilten Prozessautomatisierungssystemen in der Spezifikationsphase, Dissertation, Universität Stuttgart, 1997
- [Berg98] J. Bergmann: Funktionsprüfung der Steuerungssoftware intelligenter technischer Produkte, München, Utz Verlag, zugl. Dissertation, Technische Universität München, 1998
- [BeBe99] J. Bergmann, K. Bender: Funktionsprüfung eingebetteter Systeme der dezentralen Automatisierungstechnik, *at – Automatisierungstechnik* 47 (1999), Nr. 7, S. 320-325
- [Bert00] R. Bertsch: Konzeption des Testfallentwurfs für komponentenbasierte Software auf Basis von Use Cases und UML Sequenzdiagrammen, Wahlstudienarbeit Nr. 1759, IAS, Universität Stuttgart, 2000
- [Beus96] G. Beuselinck: Risks of Computers in Automobiles, *Risks Forum*, Vol. 18, Issue 25 (1996), <http://catless.ncl.ac.uk/Risks>, 1996
- [BGB459] Bürgerliches Gesetzbuch vom 18. August 1896 (RGBl. S. 195), (BGBl. III 400-2), <http://www.wbs.cs.TU-Berlin.de/cgi/gesetze/BGB/>
- [Bind99] R. V. Binder: Testing Object-Oriented Systems: A Status Report, <http://www.rbsc.com/pages/ootstat.html>, 1999
- [BMI92] Der Bundesminister des Innern: V-Modell 1992: Planung und Durchführung von IT-Vorhaben, Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt), 1992
- [Boeh81] B.W. Boehm: *Software Engineering Economics*, Englewood Cliffs, Prentice Hall, 1981
- [Boeh84] B.W. Boehm: Verifying and Validating Software Requirements and Design Specifications, *IEEE Software* Vol. 1 (1984), S. 75-88
- [BJR99] G. Booch, J. Rumbaugh, I. Jacobson: *Das UML-Benutzerhandbuch*, Bonn, Addison Wesley Longman, 1999
- [Bosc99] Bosch: *Kraftfahrtechnisches Taschenbuch*, Braunschweig, Vieweg Verlag, 1999

- [BrDr95] A. P. Bröhl, W. Dröschl: Das V-Modell – Der Standard für die Softwareentwicklung mit Praxisleitfaden, München, Oldenbourg Verlag, 1995
- [BGG+96] M. Broy, E. Geisberger, R. Grosu, F. Huber, B. Rumpe, B. Schätz, A. Schmidt, O. Slotosch, K. Spies: Das AUTOFOCUS-Bilderbuch – eine anwenderorientierte Beschreibung, Forschungsbericht, Institut für Informatik, Technische Universität München, 1996
- [CKB+99] I. S. Chung, H. S. Kim, H. S. Bae, Y. R. Kwon, B. S. Lee: Testing of Concurrent Programs based on Message Sequence Charts, in Proceedings of International Symposium on Parallel and Distributed System Engineering (PDSE), Los Angeles, CA, USA, 1999, S. 72-82
- [Cock97] A. Cockburn: Structuring Use Cases with Goals, The Journal of Object-Oriented Programming, Vol. 10 (5/6) Sept.-Dec. (1997), 1997
- [Daim98] Daimler-Benz AB: Presse Information zur Technologiepressekonferenz, PRI_0428.DOC, Stuttgart, 1998
- [DaHa98] W. Damm, D. Harel: LSCs: Breathing Life into Message Sequence Charts, Weizmann Institute Tech. Report CS-98-09, Weizmann Institute of Science, Rehovot, Israel, 1998
- [Dasa85] B. Dasarathy: Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them, IEEE Transactions on Software Engineering Vol. 11 (1985), No. 1, S. 80-86
- [DIN12119] DIN ISO/IEC 12119: Software-Erzeugnisse – Qualitätsanforderungen und Prüfbestimmungen, DIN Deutsches Institut für Normung e.V., Berlin, Beuth Verlag, 1995
- [DIN40041] DIN 40041: Zuverlässigkeit in der Elektrotechnik, DIN Deutsches Institut für Normung e.V., Berlin, Beuth Verlag, 1990
- [DIN44300] DIN 44300: Informationsverarbeitung: Begriffe, DIN Deutsches Institut für Normung e.V., Berlin, Beuth Verlag, 1985
- [DIN8402] DIN ISO 8402: Qualitätsmanagement und Qualitätssicherung, Begriffe, DIN Deutsches Institut für Normung e.V., Berlin, Beuth Verlag, 1992
- [Doug98] B. P. Douglass: Real-Time UML: Developing efficient Objects for Embedded Systems, Reading, Addison Wesley Longman, 1998
- [Etsch94] K. Etschberger: CAN – Controller Area Network, Hanser Verlag, München, 1994
- [Erns98] R. Ernst: Objektorientierte Softwareentwicklung für ein verteiltes, eingebettetes Echtzeitsystem mit dem Werkzeug ObjecTime Developer / ROOM, Diplomarbeit Nr. 1677, IAS, Universität Stuttgart, 1998
- [ESA96] ESA: Ariane 5 Flight 501 Failure – Report of Inquiry Board, Report No. 33-1996, http://www.esa.int/export/esaCP/Pr_33_1996_p_EN.html, Paris, Frankreich, 1996
- [ETAS00] ETAS GmbH: ASCET-SD V4.0 User's Guide, <http://www.etas.de/>, 2000
- [ETAS01] ETAS GmbH: ERCOS^{EK} V4.1 User's Guide, <http://www.etas.de/>, 2001

- [FRB97] W. Fleisch, Th. Ringler, R. Belschner: Simulation of Application Software for a TTP Real-Time Subsystem, in Proceedings of European Simulation Multiconference (ESM97), Istanbul, Türkei, 1997, S. 553-558
- [Flei99] W. Fleisch: 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, Frankreich, 1999, S. 75-84
- [Flei00a] W. Fleisch: Simulation and Validation of Component-Based Automotive Control Software, in Proceedings of 12th European Simulation Symposium (ESS2000), Hamburg, 2000, S. 417-421
- [Flei00b] W. Fleisch: Test Case Design for the Validation of Component-Based Embedded Systems, in Proceedings of International IFIP WG10.3/WG10.4/WG10.5 Workshop on Distributed and Parallel Embedded Systems (DIPES 2000), Schloss Eringerfeld bei Paderborn, 2000, S. 151-160
- [Föll78] O. Föllinger: Regelungstechnik – Einführung in die Methoden und ihre Anwendung, Berlin, Elitera-Verlag, 1978
- [FLS95] K. Frühauf, J. Ludewig, H. Sandmayr: Software-Prüfung: Eine Anleitung zum Test und zur Inspektion, Stuttgart, Teubner Verlag, 1995
- [FGG+91] B. Furht, D. Grostick, D. Gluch, G. Rabatt, J. Parker, M. McRoberts: Real-time UNIX Systems: Design and Application Guide, Norwell, MA, Kluwer Academic Publishers, 1991
- [GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Reading, MA, Addison Wesley, 1995
- [Göhn98] P. Göhner: Komponentenbasierte Entwicklung von Automatisierungssystemen, VDE-GMA Kongress 1998 (VDI Berichte Nr. 1397), Ludwigsburg, 1998, S. 513-521
- [Göhn01] P. Göhner: Softwaretechnik 1, Skript zur gleichnamigen Vorlesung, Universität Stuttgart, 2001
- [HHL93] W. A. Halang, G. Hommel, R. Lauber: Perspektiven der Informatik in der Echtzeitverarbeitung, Informatik-Spektrum (1993), Nr. 16, S. 357-362
- [Hare87] D. Harel: Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, 8(3), 1987, S. 231-274
- [HoRi99] M. Hoang, P. Rieger: Komponentenbasierte Automatisierungssoftware: Objektorientiert – anwendungsnah, München, Hanser Verlag, 1999
- [HrRu01] P. Hruschka, C. Rupp: Echtzeit für Use-Cases, OBJEKTspektrum (2001), 4/2001, S. 64-71
- [IEC61131-3] IEC 61131-3: Programmable controllers – Part 3: Programming Languages, IEC Standard 61131-3, International Electrotechnical Commission, <http://www.iec.ch/>, 1993

- [IEC61499] IEC 61499: Function blocks for industrial-process measurement and control systems – Part 1+2, IEC Standard 61499, International Electrotechnical Commission, <http://www.iec.ch/>, 2000
- [IEEE610.12] IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990 (Revision and redesignation of IEEE Standard 729-1983), New York, NY, IEEE, 1990
- [IPL01] IPL Corp.: CANTATA – Technical Brief, <http://www.iplbath.com/>, 2001
- [ISG01] ISG GmbH: ASPECT Programmierwerkzeug für Speicherprogrammierbare Steuerungen, Produktbeschreibung, <http://www.isg-stuttgart.de/>, 2001
- [ISO9126-1] ISO IEC 9126-1: Software Engineering – Qualität von Softwareprodukten - Teil 1: Qualitätsmodell, International Standardisation Organisation, Berlin, Beuth Verlag, 2001
- [ITU100] ITU-TS Recommendation Z.100: Specification and Description Language (SDL), ITU-TS, Genf, 1988
- [ITU120] ITU-TS Recommendation Z.120: Message Sequence Chart (MSC), ITU-TS, Genf, 1994
- [JCJÖ92] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard: Object-Oriented Software Engineering – A Use Case Driven Approach, Reading, Massachusetts, Addison Wesley, 1992
- [JBR98] I. Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process, Reading, Massachusetts, Addison Wesley Longman, 1998
- [GJSB00] J. Gosling, B. Joy, G. Steele, G. Bracha: The Java Language Specification, Addison-Wesley, Boston, MA, 2000
- [KoBo98] W. Kozaczynski, G. Booch: Component-Based Software Engineering, IEEE Software Vol. 5 (1998), S. 34-36
- [LaGö99] R. Lauber, P. Göhner: Prozessautomatisierung I, 3.Aufl., Berlin, Springer-Verlag, 1999
- [LiLe00] J. Liu, E. A. Lee: Component-based Hierarchical Modeling of Systems with Continuous and Discrete Dynamics, in Proceedings of 2000 IEEE International Conference on Control Applications and IEEE Symposium on Computer-Aided Control System Design (CCA/CACSD'00), Anchorage, AK, 2000, S. 95-100
- [Ligg90] P. Liggesmeyer: Modultest und Modulverifikation: State of the art, Mannheim, BI-Wissenschaftsverlag, 1990
- [Ligg93] P. Liggesmeyer: Wissensbasierte Qualitätsassistenz zur Konstruktion von Prüfstrategien für Software-Komponenten, Mannheim, BI-Wissenschaftsverlag, zugl. Dissertation, Ruhr-Universität Bochum, 1993
- [Ligg00] P. Liggesmeyer: Qualitätssicherung softwareintensiver technischer Systeme, Heidelberg, Spektrum, Akad. Verlag, 2000
- [LiRü96] P. Liggesmeyer, P. Ruppel: Die Prüfung von objektorientierten Systemen, OBJEKTSpektrum (1996), 6/96, S. 68-78

- [Lind00] P. Linder: Evaluierung eines rechnerunterstützten Validierungsverfahrens für komponentenbasierte ASCET-SD Spezifikationen, Diplomarbeit Nr. 1755, IAS, Universität Stuttgart, 2000
- [Lude97] J. Ludewig: Software Engineering, Skript zur gleichnamigen Vorlesung, Universität Stuttgart, 1997
- [MaMe88] F. Mattern, H. Mehl: Effiziente Simulation - Prinzipien, Probleme und Perspektiven, Sonderforschungsbereich 124 „VLSI Entwurfsmethoden und Parallelität“, Bericht Nr. SFB 124-39/88, Universität Kaiserslautern, 1988
- [MaRa98] L. Mattingly, H. Rao: Writing Effective Use Cases and Introducing Collaboration Cases, *The Journal of Object-Oriented Programming* (1998), Vol. 11 (6), S. 77-87
- [Micro95] Microsoft: Component Object Model Specification Draft Version 0.9, <http://www.microsoft.com/>, 1995
- [Micro98] Microsoft: Developing Visio Solutions, <http://www.microsoft.com/>, 1998
- [Moor01] A. Moore: UML und der Lauf der Zeit, *Elektronik* (2001), 7/2001, S. 93-99
- [MSR99] MSR-MEGMA: Standardization of Library Blocks for Graphical Model Exchange Specification, <http://www.msr-wg.de/>, 1999
- [Musc99] A. Muscholl: Matching Specifications for Message Sequence Charts, *FoSSaCS*, 1999, S. 273-287
- [Myer91] G. J. Myers: *Methodisches Testen von Programmen*, München, Oldenbourg Verlag, 1991
- [OMG01] OMG: CORBA Specification 2.4.2, <http://www.omg.org/>, 2001
- [Obj97] ObjecTime: *ObjecTime Developer 5.1 User's Guide*, ObjecTime Limited, 340 March Road, Kanata, Ontario, 1997
- [OSEK00] OSEK/VDX: Operating System Specification 2.1r1, <http://www.osek-vdx.org/>, 2000
- [Pere95] C. E. Pereira: Verfahren zur Beschreibung und Überprüfung von Zeitbedingungen bei der objektorientierten Entwicklung von Automatisierungssystemen, Dissertation, Universität Stuttgart, 1995
- [PoRi96] H. Pohlmann, E. H. Riedemann: Begriffsdefinitionen im Testbereich, GI-Fachgruppe 2.1.7 Test, Analyse und Verifikation von Software (TAV) der Gesellschaft für Informatik (GI), <http://www.fbe.hs-bremen.de/spillner/begriffe/home.html>, 1996
- [Post85] R. M. Poston: Software Standards – Preventing Software Requirements Specification Errors with IEEE 830, *IEEE Software* Vol. 2 No1 (1985), S. 83-86
- [Rapi97] Rapide Design Team: *Draft Guide to the Rapide 1.0 Language Reference Manuals*, Program Analysis and Verification Group, Computer Systems Lab, Stanford University, CA, <http://pavg.stanford.edu/rapide/rapide-pubs.html>, 1997
- [Rati01] Rational: *Rational Rose RealTime User's Guide*, <http://www.rational.com/>, 2001

- [Romm99] T. Rommel: Entwicklung eines Analysewerkzeugs zur Validierung komponentenbasierter Software für Echtzeitsysteme unter C++/MFC, Studienarbeit Nr. 1737, IAS, Universität Stuttgart, 1999
- [Romm01] T. Rommel: Konzeption zur Modellierung und Validierung verteilter Steuergerätesoftware mit ASCET-SD und CANoe, Diplomarbeit Nr. 1784, IAS, Universität Stuttgart, 2001
- [Schn91] H.-J. Schneider: Lexikon der Informatik und Datenverarbeitung, München, Oldenbourg Verlag, 1991
- [Seif01] U. Seiffert: Automobile Elektronik – Zukünftige Anforderungen an die Kraftfahrzeug-Elektrik/Elektronik, *Elektronik Automotive* (2001), 9/2001, S. 84-87
- [SGW94] B. Selic, G. Gullekson, P. Ward: Real-Time Object-Oriented Modeling, New York, John Wiley & Sons Inc., 1994
- [Stan92] J. A. Stankovic: Distributed Real-Time Computing: The Next Generation, *Journal of the Society of Instrument and Control Engineers of Japan*, 1992
- [StRa92] J. A. Stankovic, K. Ramamritham: *Advances in Real-Time Systems*, Los Alamitos, IEEE Computer Society Press, 1992
- [StWi00] D. Stolze, A. Willert: Qualitätssicherungs-Maßnahmen für den Software-Entwicklungsprozeß in Embedded Projekten, *Embedded Intelligence 2000 – Grundlagen, Architekturen, Werkzeuge und Lösungen* 16.-18.02.2000
- [SLSW99] A. Storr, R. Lutz, M. Seyfarth, M. Weiner: Softwaretechnik und CASE-Tools für Steuerungssoftware, ISW Steuerungstechnisches Kolloquium '99, Stuttgart, 1999
- [Stra97] D. Strassacker: Testumgebung für die Implementierung und Inbetriebnahme eines adaptierbaren Leitsteuerungssystems, Berlin, Springer-Verlag, zugl. Dissertation, Universität Stuttgart, 1997
- [Sun00] Sun Microsystems: Enterprise JavaBeans Specification Version 2.0, Proposed Final Draft, <http://www.javasoft.com/products/ejb/>, 2000
- [Szyp98] C. Szyperski: *Component Software – Beyond Object-Oriented Programming*, Reading, MA, Addison-Wesley Longman, 1998
- [Thal97] G.E. Thaller: *Software Engineering für Echtzeit und Embedded Systems*, bhv Verlags GmbH, 1997
- [UML99] Unified Modeling Language - UML - Notation Guide version 1.3, Rational Software Corporation, Santa Clara, CA, <http://www.rational.com>, 1999
- [Vect00] Vector Informatik GmbH: CANoe Arbeitshandbuch V3.0, <http://www.vector-informatik.de/>, 2000
- [WaCo96] S. Waligora, R. Coon: *Improving the Software Testing Process in NASA's Software Engineering Laboratory*, 1995 Software Engineering Laboratory Study, NASA, 1996
- [Wall90] E. Wallmüller: *Software-Qualitätssicherung in der Praxis*, München, Hanser Verlag, 1990

[Zale01] A. Zalevski: Qualitätssicherung bei Bordsystemen, Embedded Engineering (2001), 9/2001, S. 42-46

Lebenslauf

Persönliche Daten

08. 08. 1968 geboren in Tübingen

Schulbildung

Sept. 1975 – Aug. 1979 Grundschule Calw-Heumaden

Sept. 1979 – Mai 1988 Hermann Hesse-Gymnasium Calw, Abschluss Abitur

Wehrdienst

Juli 1988 – Sept. 1989 2. Fallschirmjägerbataillon 251, Calw

Studium

Okt. 1989 – Apr. 1996 Studium der Elektrotechnik an der Universität Stuttgart
Studienmodell: Regelungstechnik und Prozessautomatisierung
Hilfswissenschaftler am Institut für Mikroelektronik Stuttgart,
Abteilung Spezialprozessoren und Test, Stuttgart

Auslandspraktikum bei Nokia Mobile Phones,
R&D DSP Software Development, Oulu, Finnland

30.04.1996 Abschluss als Diplom-Ingenieur

Berufstätigkeit

Mai 1996 – April 1999 Stipendiat im Graduiertenkolleg „Parallele und Verteilte Systeme“ der Universität Stuttgart

Mai 1996 – Feb. 2002 Wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart

seit April 2002 Mitarbeiter der Robert Bosch GmbH,
Geschäftsbereich Diesel Systems, Stuttgart