Audi
Dissertationsreihe

Datenbankgestützte Prozessautomatisierung bei Software-Tests

Vladimir Entin

Cuvillier Verlag Göttingen

INI.FAU
INGOLSTADT INSTITUTE OF THE FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG

Datenbank-gestützte Prozessautomatisierung bei Software-Tests

Der Technischen Fakultät der
Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Vladimir Entin

Erlangen - 2010

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
1. Aufl. - Göttingen: Cuvillier, 2010
Zugl.: Erlangen-Nürnberg, Univ., Diss., 2010

978-3-86955-354-2

Audi Dissertationsreihe, Band 33

Als Dissertation genehmigt von der Technischen Fakultät der Universität Erlangen-Nürnberg

Tag der Einreichung: 24.02.2010
Tag der Promotion: 07.05.2010
Dekan: Prof. Dr.-Ing. Reinhard German
Berichterstatter: Prof. Dr.-Ing. Klaus Meyer-Wegener

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage, 2010
Gedruckt auf säurefreiem Papier

978-3-86955-354-2

Моим родителям с любовью и глубокой признательностью.

Zusammenfassung

Die vorliegende Arbeit wurde im Rahmen der wissenschaftlichen Kooperation zwischen Friedrich-Alexander-Universität Erlangen-Nürnberg und Audi AG Ingolstadt am Ingolstadt Institute der Friedrich-Alexander-Universität Erlangen-Nürnberg (INI.FAU) erstellt. In dieser Arbeit wurde eine systematische Methodik (ein Vorgehensmodell) zur formalen Definition von Testspezifikationen für eingebettete reaktive Systeme entwickelt. Die Methodik basiert auf dem Grundgedanken der Modell-getriebenen Software-Entwicklung (MDSD), bei der es darum geht, aus formalen und plattformunabhängigen Modellen mittels Verfeinerungsschritten und der automatisierten Transformation in ein bestimmtes Zielformat lauffähige Applikationen zu erzeugen. Mit dem vorgestellten Ansatz soll folgendes erreicht werden:

1. Vereinheitlichung der Testspezifikationsbeschreibung: Unter der Berücksichtigung der komplexen Organisationsstruktur eines Unternehmens soll eine minimale gemeinsame Basis für alle laufenden Projekte und eingesetzten Testverfahren geschaffen werden, die es erlaubt komplette Testspezifikationen oder ihre Teile untereinander, also Projektgrenzenübergreifend innerhalb einer Entwicklungsphase oder gar über die Entwicklungsphasen hinweg, auszutauschen.

2. Formatunabhängigkeit der Testspezifikationsbeschreibung: Die erzeugten Testspezifikationen sollen formatunabhängig sein. Dies bedeutet, dass sie weder als elektronische Dokumente noch als sonstige gängige Formate wie zum Beispiel XML verwaltet werden. Vielmehr geschieht die Verwaltung von Testspezifikationen in einem (Meta)Modell.

3. Mehrplattform-Fähigkeit der Testspezifikationsbeschreibung: Um die Wiederverwendbarkeit der Testspezifikationen auch innerhalb einzelner Projekte zu erhöhen, sollen diese auf mehrere Zielformate abbildbar sein.

4. Schaffung der notwendigen Voraussetzung zum Testwissenstransfer zwischen den frühen Entwicklungsphasen eingebetteter reaktiver Systeme und den nachgelagerten.

5. Die Methodik soll nahtlos in den täglichen Entwicklungsprozess integrierbar sein.

Als Anwendungsgebiet der entwickelten Methodik galt das Gebiet der Fahrer-Assistenzsysteme.

Vor der Konzeption der Methodik fand eine eingehende Untersuchung der Anwendungsdomäne, nämlich der Elektronik-Entwicklung entlang der Entwicklungsprozess-Kette statt. Während dieser Untersuchung wurden primär die bereits eingesetzten Testverfahren analysiert, um anschließend ihre Relevanz für die vorliegende Arbeit zu ermitteln. Ebenfalls galt es, die grundlegenden und zum Teil gravierenden Unterschiede zwischen den frühen und den nachgelagerten Entwicklungsphasen herauszuarbeiten. Schließlich mussten Defizite bei den vorhandenen Testverfahren ermittelt werden. Basierend auf diesen sowie den definierten Zielen dieser Arbeit, wurden dann die wissenschaftlichen Fragestellungen erarbeiten.

Anschließend galt es zu untersuchen, ob es im Bereich des Testens bereits Ansätze zur Lösung dieser Fragestellungen gibt, beziehungsweise inwieweit diese zu deren Lösung unter den gegebenen Rahmenbedingungen geeignet wären. Die Untersuchung fand sowohl im Bereich des Modell-basierten Testens statt als auch im allgemeinen Testbereich. Nach der fundierten Erarbeitung des Vorgehensmodells wurde die vorgestellte Methodik auf die Domäne der Fahrer-Assistenzsysteme (FAS) angewandt. Hierbei wurde in einer Vorserien-Entwicklungsabteilung im konzeptuellen Bereich eine Metamodell-Definition für die FAS-Anwendungsdomäne vorgenommen. Im praktischen Bereich efolgte die Schaffung einer entsprechenden Infrastruktur zur Verwaltung von Testspezifikationen. Schließlich wurde untersucht, inwieweit sich die entwickelte Methodik in den täglichen Entwicklungsprozess bei einer Vorserien-Entwicklungsabteilung integrieren lässt und vor allem wie die Entwickler mit dieser zurechtkommen.

Abstract

This doctoral thesis has been written in the context of a cooperation between Friedrich-Alexander-University of Erlangen and Nuremberg and Audi AG at Ingolstadt Institute of Friedrich-Alexander-University of Erlangen and Nuremberg (INI.FAU). The main goal of this thesis was the conception of a systematical approach to the formal definition of test-specifications for the embedded reactive systems. The presented approach is based on the ideas of model-driven software development (MDSD) which aims at the creation of executable programs from the formal model descriptions via refinement and transformation to a specific format. The most important goals of the approach are as follows:

1. Standardization of the test-specification description: considering the complex organizational structure of industrial companies a minimal basis should be created which allows the exchange of whole test-specifications or its parts between different running projects or even across development stages of the embedded automotive software.

2. Format-independence of test-specifications: the created test-specifications should be independent of any specific format. This implies that they are neither stored as electronic documents nor are they described by some well-known markup languages. Their description is rather based upon a meta-model.

3. Multi-platform applicability of test-specifications: in order to step up the reusability of test-specifications within the single projects, the latter should be mappable to various target formats.

4. Creation of necessary pre-conditions for the transfer of test-knowledge between the early development stages of embedded reactive systems and the following ones.

5. The presented approach should be seamlessly integrated into the daily development process.

Due to the context in which the doctoral thesis has been written, the area of application of the latter is driver assistance systems (DAS). Before the conception of the approach an in-depth

analysis of the driver assistance systems domain has been conducted. At different development stages the currently applied test methods have been analyzed in order to determine their relevance for the thesis. Moreover, the considerable differences between the development stages have been elaborated. Finally, the deficits of currently employed test methods have been analyzed. Basing on these deficits as well as on the goals of the thesis, scientific questions have been defined. Subsequently, the research has been done in order to find out whethere there are already approaches which provide answers to the previously defined scientific questions considering also the context of the thesis. The above-mentioned research had its focus not solely on the area of model-based testing but also on the general testing techniques. After having elaborated the approach to the formal definition of test-specifications, it has been applied to the area of the driver assistance systems. To that end a suitable meta-model for DAS as well as an infrastructure for the definition, management and transformation of the test-specifications have been created. Finally an evaluation of the approach had its focus on determining how far the latter could be integrated into the daily development of the DAS. The evaluation took place at a pre-development department at Audi AG Ingolstadt.

Danksagung

Es war ein sehr schwüler Tag in Rehovot im Oktober 2009 als ich auf eigene Faust den Campus des Weizmann-Institut der Wissenschaften erkundete. Riesige Eukalipten, blühende Magnolien, schattenspendende Palmen, bunte Bouganvillen und viele Blumenarten, die man sozusagen als Polarkreis-Bewohner eigentlich gar nicht kennt. Nach vier Stunden Spaziergang setzte ich mich auf eine Bank in einer entfernten Ecke des Parks, um ein Schluck Wasser zu nehmen. Inmitten dieser Farbenpracht sah ich plötzlich vor mir einen engen Streifen Sand, auf dem eine anastatica hierochuntica, besser bekannt als Wüstenrose, wuchs. Erst dann wurde mir klar, dass noch vor dreißig oder vierzig Jahren dieses Stück Land lediglich aus glühendem Sand bestand. Es ist nur vielen Menschen zu verdanken, die es geschafft haben, an diesem Ort einen so prächtigen Garten errichtet zu haben.

Und so geschieht es mit jedem, der in ein neues Land kommt. Gleich einer Wüstenrose betreten wir voller Erinnerungen ein Territorium, auf dem es gilt, unsere Wurzeln neu wachsen zu lassen, einen Garten aufzubauen, in dem uns nicht Blumen und Bäume, sondern Menschen umgeben. Menschen, die uns helfen, die uns begleiten, zu unseren Freunden werden und uns in Erinnerung bleiben.

Ich hatte das Glück und gleichzeitig die Herausforderung, den Weg der Wüstenrose zu gehen. Ich möchte nun an dieser Stelle allen Menschen danken, die mich auf meinem langen Bildungsweg in Deutschland begleitet haben.

In erster Linie gilt mein tiefster Dank meinem Doktorvater Herrn Professor Dr.-Ing. Klaus Meyer-Wegener. Trotz vieler Ämter und Aufgaben fand er immer Zeit, mich eng fachlich und persönlich zu betreuen. Ich wünsche jedem Doktoranden, einen derart engagierten und motivierten Doktorvater zu haben.

Herrn Professor Dr.-Ing. German danke ich für die Übernahme des Koreferats. Bei Herrn Professor Dr.-Ing. Schröder-Preikschat bedanke ich mich für die Übernahme des Vorsitzes des Prüfungsausschusses. Ebenfalls danke ich Herrn Professor Dr.-Ing. Sandro Wartzack als fachfremdem Mitglied der Prüfungskommission.

Herrn Dr.-Ing. Uwe Koser danke ich dafür, dass er durch sein persönliches Engagement das INI.FAU-Programm ins Leben gerufen hat und es ermöglicht hat, jungen Menschen aus unterschiedlichsten Fachrichtungen eine Promotion in Zusammenarbeit mit AUDI AG zu machen.

Ein besonderer Dank gebührt ebenfalls Herrn Dr.-Ing. Karl-Heinz Siedersberger für eine großartige Betreuung von Seiten AUDI AG während meiner Zeit bei Ingolstadt Institute der Friedrich-Alexander-Universität Erlangen-Nürnberg (INI.FAU). Er hat mir mit viel Geduld die spannende Welt der Fahrer-Assistenzsysteme erklärt. Das einzigartige Arbeitsklima, das im AUDI-Projekthaus Fahrer-Assistenzsysteme herrschte und immernoch herrscht ist ebenfalls ihm zu verdanken. Es ist einfach vorbildlich, wieviel Bereitschaft und Motivation er hatte, sich in unterschiedlichste Denkweisen einzuarbeiten, die jeder von uns in einem interdisziplinären Team hatte.

Da ich AUDI-seits mehrere Betreuer hatte, möchte ich nun auch diesen danken. In erster Linie gilt mein Dank Frau Dipl.-Ing. Carmen Löbel, die mich schon während meiner Diplomarbeit bei Audi Electronics Venture GmbH betreut und diese Promotion aus organisatorischer Sicht erst ermöglicht hat. Leider hatte sie dann die Abteilung gewechselt, aber wir sind trotzdem in Kontakt geblieben und ich glaube, dass, neben hoher fachlicher Kompetenz, es wenige Menschen gibt, die bei über 200 km/h Geschwindigkeit während der Fahrt durch Altmühltal am Steuer eines offenen Cabrios völlig entspannt über die Freude am Autofahren erzählen können.

Herrn Dipl.-Ing. Andreas Kern danke ich ebenfalls für die Betreuung und vor allem für die Freiräume, die er mir während meiner Arbeit ließ. Es ist mit Sicherheit das Zeichen des großen Vertrauens seinerseits gewesen.

Herrn Dipl.-Inf. Sebastian Siegl fühle ich mich tief verbunden, denn wir haben miteinander nicht nur an interessanten wissenschaftlichen Publikationen in dieser Zeit gearbeitet, sondern es hat sich in den drei Jahren eine tolle Freundschaft entwickelt, die auch die Grenzen dieses Projekts überdauern wird. Es gibt wenige Menschen, die schon in der Schulzeit um Chinesisch zu lernen, dazu bereit waren, in den Ferien hunderte von Kilometern zur Sprachschule zu fahren. Es ist also die Passion für Sprachen und insbesondere für das Italienische, die uns verbindet.

Nun möchte ich jedem einzelnen Team-Kollegen aus dem AUDI-Projekthaus nicht nur für sehr hilfreiche fachliche Diskussionen danken, sondern auch für tolle Zeit und für die Möglichkeit sie, so einzigaritg wie sie sind, kennengelernt zu haben.

Herrn Dipl.-Inf. Richard Schneidt danke ich für die enorm hilfreichen praktischen programmiertechnischen Ratschläge. Wenn man einem Informatiker ein Kompliment machen möchte, dann sagt man scherzhaft er sei ein Hacker. Und das ist er!

Herrn Dipl.-Ing. Waldemar Winter danke ich für schnelle und unkomplizierte Lösung aller

organisatorischen Fragen, die während eines Projekts unvermeidlich sind. Er ist das Urgestein des Projekthauses und hat stets mit guter Laune und schönen Witzen zu einem herrlichen Arbeitsklima beigetragen.

Monsieur Dipl.-Ing. Essayed Bouzouraa danke ich für die Unterstützung bei der Realisierung des automatisierten Testablaufs und auch für die Geschichten über Tunesien, seine Bräuche, tunesische, arabische und französische Sprachen. Ich hoffe, unser Kontakt wird nicht abreißen und wir werden noch viele Sprichwörter aus dem Tunesischen und dem Russischen miteinander vergleichen.

Herrn Dipl.-Ing. Benedikt Strasser danke ich für die tolle Zusammenarbeit bei der Analyse der Testprozesse bei AUDI und für die Fähigkeit auch dann humorvoll zu bleiben, wenn es eigentlich gebrannt hat. Er hat von mir die Liebe zum roten Kaviar gelernt und wird hoffentlich ab und zu auf einen Kaviarabend vorbeikommen.

Herrn Dipl.-Ing. Michael Bär danke ich für Einblicke in die Welt der Neigetechnik sowie für die „Karl Fazer“ Pralinen, die er aus Schweden mitgebracht hat und die mich in der Implementierungsphase sehr unterstützt haben. Wie hatten viel über Gott und die Welt diskutiert und gelacht. Wir werden unsere Diskussion mit Sicherheit bei Gelegenheit weiterführen.

Herrn Dipl.-Ing. Michael Reichel danke ich ebenfalls für die Unterstützung bei der Implementierung des automatisierten Testablaufs. Ich war angenehm überrascht als er beim Reden ganz nebenbei Russische Ausdrücke verwendet hat.

Herrn Dipl.-Ing. Markus Hörwick danke ich für philosophische Diskussionen über die Werke von Milan Kundera.

Herrn Dipl.-Ing. Martin Wimmer danke ich für ein paar Geheimtipps aus der Welt des Basketballs, meiner anderen Passion, die ich nach dem Ende der Promotion unbedingt ausprobieren werde (hoffentlich ohne bleibende Schäden für den Gegner). Wir werden uns ab und zu bei den Ice-Hockey-Spielen in Nürnberg oder Ingolstadt sehen.

Herrn Dr.-Ing. Ulrich Hofmann danke ich dafür, dass er stets gezeigt hat, wie man bei Diskussionen geschickt eigenen Standpunkt verteidigt. Ich hoffe, wir werden noch viele Gespräche über die Weltordnung, Musik und Kunst beim Pizza-Essen in Erlangen führen.

Herrn Dr. rer. nat. Rolf Dubitzky danke ich für tolle Einblicke in die faszinierende Welt der Teilchenphysik und insbesondere für spannende Berichte über Teilchenbeschleuniger. Teilchenphysik ist fürwahr die Königin aller Wissenschaften!

Herrn Dipl.-Ing. Andreas Siegel danke ich für angenehme Zusammenarbeit und für tolle Reiseberichte insbesondere über die Mongolei.

Den Herren M.Sc. Manuel Siebeneicher, Dipl.-Inf. Jens Storz und Dipl.-Inf. Jan Winhuysen

danke ich dafür, dass sie bei all der Sensordaten- und Gedankenfusion, die im Projekthaus getrieben wird, immer die stoische Ruhe bewahrt haben und mit gutem informatischen Rat uns allen zur Seite standen.

Ich danke an dieser Stelle ebenfalls allen meinen und allen anderen Studenten, die mich unterstützt haben: Matthias Ißbrücker, Ke Chang, Robert Rauschecker, Peter Bergmiller, Christopher Demiral, Christoph Müller.

Auch wenn ich zugegebenermaßen Schwierigkeiten hatte, die Balance zwischen Lehrstuhl und Projekthaus beizubehalten, möchte ich an dieser Stelle allen meinen Kollegen vom Lehrstuhl sechs für Datenmanagement der FAU Erlangen-Nürnberg ganz herzlich für jegliche Unterstützung danken. Diese sind: Nadezda Jelani, Ursula Stoyan, Roswitha Braun, Ursula Büttner, Dipl.-Inf. Juliane Blechinger, Professor Dr.-Ing. Richard Lenz, Prof. em. Dr. Dr.-Ing. E.h. Hartmut Wedekind, Dipl.-Inf. Christoph Neumann, Dipl.-Inf. Robert Nagy, Dipl.-Inf. Thomas Fischer, Dipl.-Inf. Michael Daum, Dipl.-Inf. Frank Lauterwald, Dipl.-Inf. Florian Irmert.

Ganz besonders möchte ich Frau Nadezda Jelani dafür danken, dass sie mich bei meinen zahlreichen Dienstreisen stets mit Ratschlägen zur Antragsausfüllung unterstützt hat.

Frau Erika Hladky danke ich für die große Hilfe bei der Terminfindung für die Promotionsprüfung.

Außerdem gilt mein tiefster Dank Herrn Gerhard Lochner, stellvertretendem Rektor des Helene-Lange Gymnasiums in Fürth, der mir wahrlich die Tür in die akademische Welt geöffnet hat und mir eine Chance gegeben hat, mit anfangs sehr geringen Deutsch-Sprachkenntnissen am Gymansium das Abitur zu machen. Herr Lochner, ich habe all Ihre Erwartungen und Prognosen erfüllt!

Ebenso danke ich Frau Dr. Gabriella Dondolini-Scholl vom Sprachenzentrum der FAU Erlangen-Nürnberg, die mir vom ersten bis zum letzten Semester die Italienische Sprache soweit beigebracht hat, dass ich 2009 beim weltweiten universitären Schreibwettbewerb in Italienisch, das vom Italienischen Außenministerium und Accademia della Crusca veranstaltet wird, den Platz unter den zehn besten belegt habe. Wenn alle Philologen so wären wie sie! Danke ebenfalls an Dr. Davide Schenetti, der mich damals beim Wettbewerb unterstützt hat.

Schließlich, wie bereits auf der ersten Seite erwähnt, widme ich diese Arbeit meinen Eltern, die all dies ermöglicht haben. Meine besten Freunde, Trainer, Berater und Lehrer!

Благодарности

Дорогие мои родственники и друзья, спасибо Вам за все совместно проведённые годы и за то, что мы не потеряли друг друга из виду, а также за живой интерес к моим успехам и достижениям. Одновременно прошу также прощения за то, что в последние три года я так мало Вас навещал. Эту работу я посвящаю родителям и, конечно же, Вам! Мы победили!

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Testen im Bereich eingebetteter reaktiver Systeme

„Die Menschheit unterteilt sich in drei Kategorien: Die Unbeweglichen, die Beweglichen und diejenigen, die sich bewegen“ besagt ein arabisches Sprichwort. Unternimmt man den zugegebenermaßen kühnen Versuch, dieses Zitat auf die heutige Welt zu übertragen, dann gibt es eigentlich nur eine Kategorie von Menschen: Jene, die sich bewegen, denn die räumliche Mobilität hat in den letzten Jahrzehnten in unserer Gesellschaft immer mehr an Bedeutung gewonnen und sie ist inzwischen zum wichtigen Faktor für den wirtschaftlichen Fortschritt geworden. Sofort würde sich die Frage stellen, was denn mit den restlichen zwei Kategorien geschehen ist. Die erste, also die Immobilität, ist heutzutage wohl auf Probleme verkehrstechnischer Natur zurückzuführen: Man steht im Stau, es gibt einen Software-Fehler im Hochgeschwindigkeitszug oder man muss gerade im Verkehrsatlas nachblättern, weil das Navigationssystem keine korrekten Informationen anzeigt. Die zweite Kategorie, nämlich die Beweglichen, ist als eine körperliche Fähigkeit zu betrachten, die allerdings aus der heutigen Sicht eher der ersten Kategorie zuzuordnen ist, denn sie ist von wenig Nutzen, wenn unsere hochkomplexen Fortbewegungsmittel einfach sich nicht so verhalten, wie sie sich verhalten sollten. Dieses umgangssprachlich beschriebene Problem: Die Fehleranfälligkeit der heutigen eingebetteten Software birgt in sich in der Realität eine weitaus höhere Gefahr als nur Unannehmlichkeit für den Endnutzer, sie kann im schwerwiegendsten Fall zum ernsthaften Image-Schaden für ein Unternehmen und somit auch zum Wettbewerbsnachteil führen. Auch die Statistik bestätigt das gerade formulierte Problem: Die Anzahl der Rückrufaktionen etwa im Personalkraftwagen-Bereich (Abbildung 1.1) steigt kontinuierlich. Nicht zuletzt liegt dies an der Fahrzeugelektronik, die inzwischen zum großen Einsatzgebiet für die modernen eingebetteten Systeme geworden ist. Betrachtet man die Pannenstatistik, so

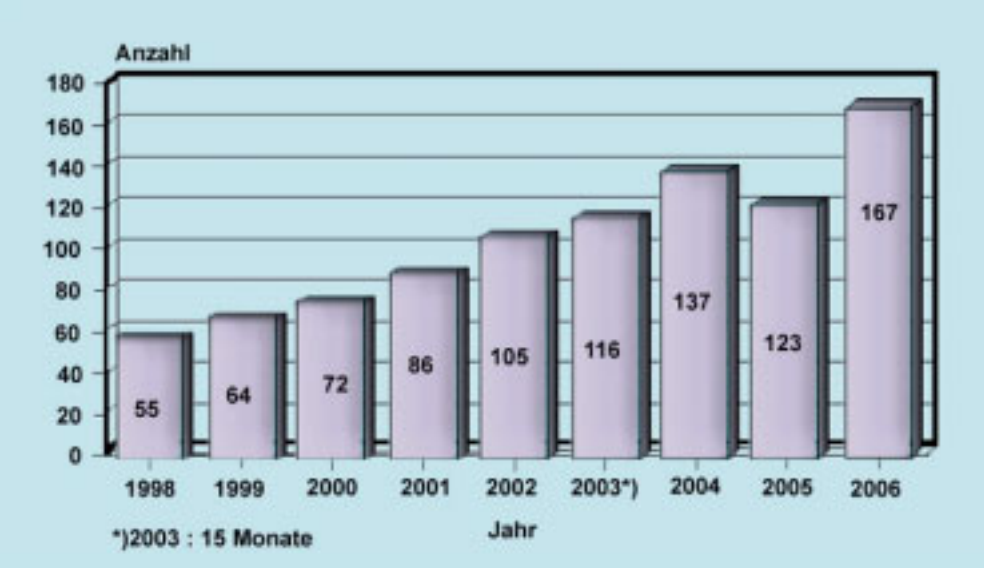


Bild 1.1: Anzahl der Rückrufaktionen [Imm07]

findet man die Ausfälle der Fahrzeugelektronik als dritthäufigste Pannenursache (Abbildung 1.2). Laut [Aut07] stieg die Fehlerhäufigkeit der Fahrzeugelektronik im Jahr 2007 um über 40 Prozent gegenüber dem Vorjahr, jeder sechste Ausfall hatte seine Ursache in gestörter Elektronik. Im vorigen Jahr war es noch jeder neunte.

Bei näherer Betrachtung lässt sich ein starker Wettbewerb auf dem Gebiet der Fahrzeugelektronik beobachten. „Die Anbieter versuchen ihre Profitabilität durch Ausbau ihrer Marktanteile zu steigern. Die Mittel dazu sind unter anderem zunehmende Variantenvielfalt und kürzere Modellzyklen. Ebenfalls steigen die Kundenerwartungen in Bezug auf die funktionale Ausstattung“ [Wol08]. Betrachtet man die Entwicklung in letzter Zeit, so ist eine steigende Anzahl an den in einem Fahrzeug verbauten elektronischen Systemen zu beobachten. Dies erfordert zunehmend deren Interaktion untereinander. In der Abbildung 1.3 ist die Vielfalt und die Menge der verbauten Systeme verdeutlicht: Adaptive Cruise Control (ACC) und Elektronisches Stabilitätsprogramm (ESP) [DJG03] sind die bekanntesten Beispiele davon. Jedes dieser Systeme besteht aus diversen Funktionen. Unter Funktion wird in diesem Kontext eine Softwarekomponente verstanden, welche die (Teil-)Funktionalität eines oder mehrerer automobiler Systeme realisiert. Es können mehrere Funktionen auf einem Steuergerät verbaut sein. Selbstverständlich kann eine Funktion auch über mehrere Steuergeräte verteilt sein. Aus der Kundensicht dienen die Funktionen dazu, unter anderem den Fahrkomfort und die Insassensicherheit zu erhöhen, indem diese nicht nur in das Fahrzeugverhalten eingreifen, sondern auch mit dem Kunden mit Hilfe akustischer,

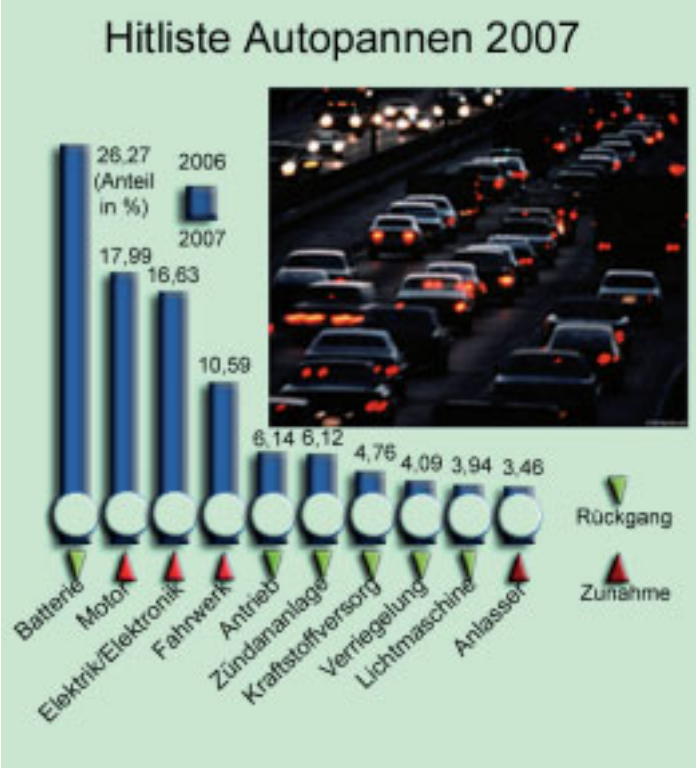


Bild 1.2: Statistik der Elektronikausfälle in den Kraftfahrzeugen [Aut07]

haptischer und visueller Signale interagieren. Zu den Funktionen gehört beispielsweise ABS (Antiblockierungssystem) [Aut10].

Ein wichtiges Gebiet der aktuellen Kraftfahrzeugelektronik sind Systeme der aktiven Sicherheit und Fahrer-Assistenzsysteme [WHW09], bei welchen die erwähnte Interaktion zwischen den einzelnen Funktionen sehr komplex ist. Dies wiederum stellt deutlich höhere Anforderungen an diverse existierende Test- und Simulationsverfahren und vor allem an deren Integration untereinander als noch vor einigen Jahren.

Testen ist zum unabdingbaren Bestandteil des automobilen Software-Entwicklungsprozesses geworden. Aus diesem Grund wird der Entwicklungsprozess als Kontext, in welchem die Testvorgänge stattfinden, im Folgenden dargestellt. Dieser ist durch zunehmende Teilung der Entwicklungsaufgaben auf mehrere beteiligte Einheiten gekennzeichnet.

Der Entwicklungsprozess beginnt meistens in der Forschungsabteilung eines Automobil-Herstellers. Dort werden Ideen geboren und erste Prototypenteile entwickelt. Im nächsten Schritt wird die prototypisch entwickelte Software an eine Vorentwicklungsabteilung übergeben, deren Ziel es ist, die Machbarkeit der von der Forschung entwickelten Idee zu zeigen sowie weitere Prototypen

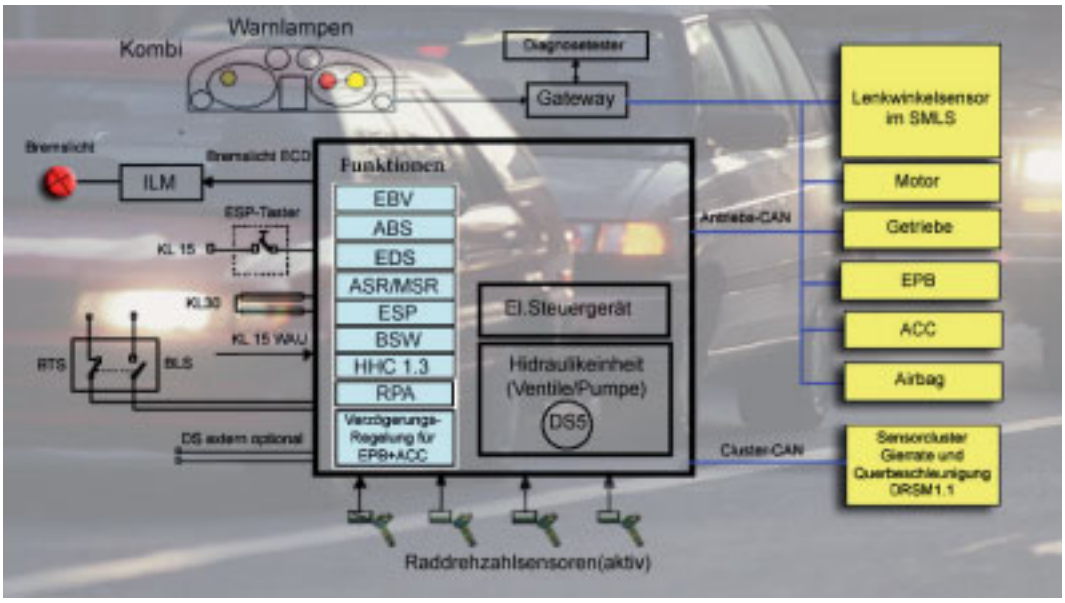


Bild 1.3: Funktionen im modernen Fahrzeug [Far05]

der zu entwickelnden Funktion zu realisieren und ein entsprechendes Lastenheft zu erzeugen. Sobald der Funktionszustand einen bestimmten Reifegrad erreicht, wird die entwickelte Software samt Lastenheft über die Grenzen des Unternehmens hinaus an ein Zulieferer-Unternehmen übergeben. Dieses entwickelt die vorliegende Funktion im Hinblick auf bestimmte in der Serienproduktion eingesetzte Steuergeräte, ergänzt das Lastenheft und übergibt schließlich eine fertig implementierte Funktion an die Serien-Entwicklungsabteilung (Synonym: Original Equipment Manufacturer oder OEM). Charakteristisch für den Entwicklungsprozess ist, dass die OEMs mit mehreren Lieferanten zusammenarbeiten. Oftmals ist jeder einzelne Lieferant für die Implementierung nur einer bestimmten Funktion zuständig. Der OEM beschäftigt sich hauptsächlich mit der Integration der von den Lieferanten implementierten Funktionen und der Gesamtfahrzeugerprobung.

Dieser in der Praxis beobachtete Prozess findet ebenfalls eine Bestätigung in der Literatur über das automobile Software-Engineering (Abbildung 1.4). In jeder Entwicklungsphase kommen diverse Entwicklungsmethoden zum Einsatz, die durch entsprechende Werkzeuge unterstützt werden. Eine große Verbreitung hat in den letzten Jahren die modellbasierte Entwicklung gefunden, bei welcher die zu entwickelnde Funktion erst als meist graphisches Modell mit entsprechender Parametrisierung vorliegt. Anschließend wird durch den Einsatz von Generatoren der Code für das entsprechende Ziel-Steuergerät produziert. Nichtsdestotrotz kann die modellbasierte Entwicklung nicht komplett die manuelle Codeentwicklung ersetzen, da bestimmte syntaktische Kosntrukte, wie

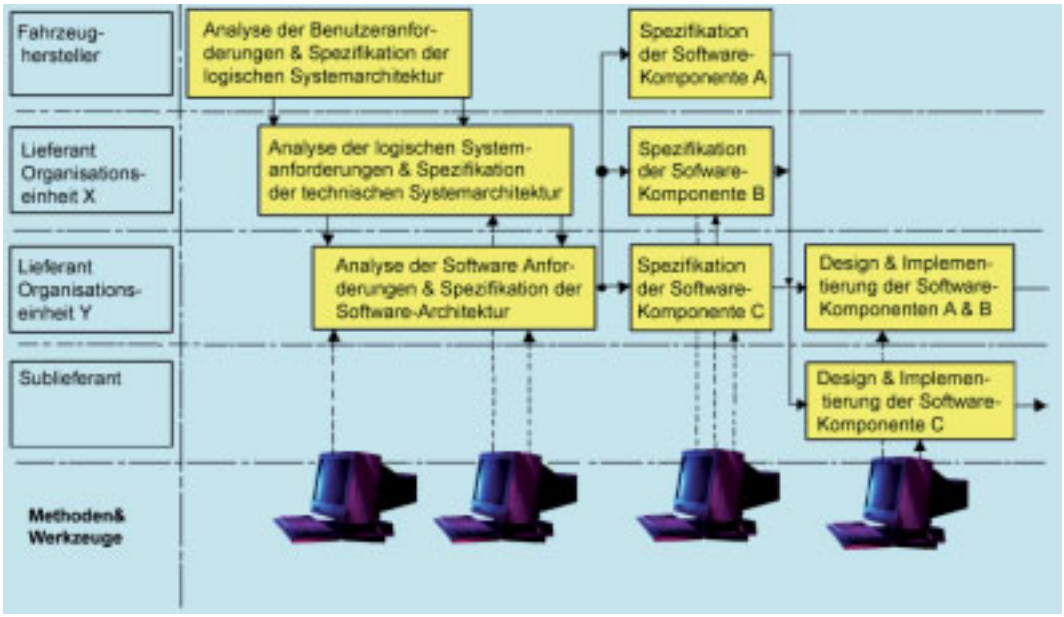


Bild 1.4: Automobiler Software-Entwicklungsprozess [SZ06]

Bildung von Schleifen oder Bedingungen, in den graphischen Modellen nicht explizit vorhanden sind und nur umständlich nachgebildet werden können. Sowohl Zulieferer-Unternehmen als auch Entwicklungsabteilungen entwickeln daher ihren Code teilweise noch manuell.

Im Folgenden wird nun auf die grundsätzlichen Testverfahren in dem dargestellten Entwicklungsprozess eingegangen. In jeder der vorgestellten Entwicklungsphasen existieren bereits etablierte Testverfahren. Die Mehrheit dieser Verfahren verläuft automatisiert, also ohne den menschlichen Eingriff während der Ausführung. Da es sich um das Testen reaktiver Systeme mit komplexer Umweltinteraktion handelt, müssen die entwickelten Verfahren diese berücksichtigen. In der Fachliteratur wird der Begriff „Regelkreis (engl. loop)“ [SZ06] für die in der Abbildung 1.5 dargestellten Testverfahren eingeführt. Bei genauer Betrachtung ist bei den dort dargestellten Verfahren eine Rückschleife sowohl von der zu testenden Funktion zur Umwelt als auch vom Fahrer zur Umwelt zu erkennen. Bei den im Regelkreis durchgeführten Testvorgängen wird, je nachdem in welcher Form die Funktion vorliegt, grundsätzlich zwischen drei Testverfahren unterschieden: „Model-in-the- Loop“(MiL), „Software-in-the-Loop“(SiL) und „Hardware-in-the-Loop“ (HiL) [SZ06]. Wichtig anzumerken ist, dass die erwähnten Testverfahren allesamt im Labor angewandt werden. Daher muss die komplette Umgebung simuliert werden, also der Verkehrsfluss, die Dynamik des Eigenfahrzeugs (EGO), das Fahrerverhalten sowie das Verhalten der Sensoreinheiten des Eigenfahrzeugs. Im Eigenfahrzeug wird sowohl das zu testende System oder die zu

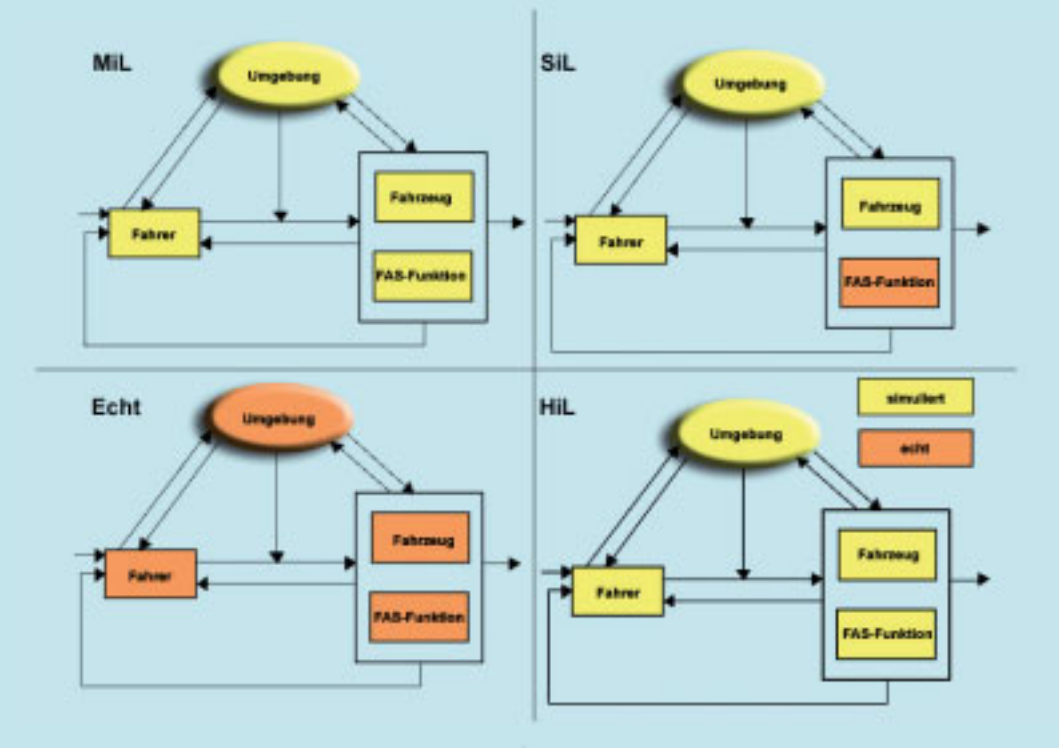


Bild 1.5: Testverfahren im automobilen Software-Entwicklungsprozess [MDBS08]

testende Funktion installiert als auch die zugehörige Sensorik verbaut. Bei der Model-in-the-Loop Simulation liegt die zu testende Funktion als Modell vor, bei SiL als ein Softwaremodul, während sie bei HiL bereits auf eines oder mehrere Steuergeräte portiert worden ist, die wiederum in einem Prüfstand integriert sind. Die genaue Betrachtung der hier eingeführten Testtechniken findet in Kapitel 2 statt. Selbstverständlich finden auch Tests im realen Fahrzeug statt, indem Testfahrten beispielsweise auf einem Prüfgelände durchgeführt werden.

Aufgrund der bereits erwähnten steigenden Komplexität der reaktiven Systeme hat das Testen eine immer größere Bedeutung, denn „manche von ihnen greifen ohne explizite Anforderung des Fahrers in die Fahrdynamik eines Fahrzeugs ein und müssen somit einen erhöhten Grad an Zuverlässigkeit aufweisen. Bei autonom intervenierenden Assistenzsystemen versuchen, beispielsweise, die Systeme „aktiver Sicherheit“, die Unfallfolgen zu mindern oder gänzlich zu vermeiden“ [Boc08]. Gerade das Testen solcher Systeme unter realen Bedingungen stellt sich als sehr aufwändig dar und unter Umständen sogar lebensgefährlich. Der Aufwand hat vor allem mit der Zeit und Kosten für den Aufbau und Vorbereitung entsprechender Teststreckenabschnitte auf einem Prüfgelände zu tun. An dieser Stelle sind Simulationsumgebungen gefordert und

teilweise schon im Entstehen, welche anhand einer großen Menge von so genannten Szenarien automatisiert den kompletten Test im Sinne eines SiL-Tests simulieren und bestimmte Funktionsausgangsgrößen auswerten (Black-Box Testing). Ein Szenario, in diesem konkreten Fall, ist die Beschreibung einer Abfolge von Fahrmanövern des Eigenfahrzeugs sowie der am Verkehrfluss beteiligten Fremdfahrzeuge in einem bestimmten Zeitabschnitt. Ein Szenario beschreibt ebenfalls die Randbedingungen, unter denen die Manöver durchgeführt werden wie zum Beispiel Straßenart, Staßenbelag, Beleuchtung oder Lichtverhältnisse.

In diesem Kontext sind auch neue Testmethodiken entstanden, welche die Vorteile der realen Testfahrten mit der virtuellen Simulation verbinden. Zu diesen gehört beispielsweise der Vehicle-in-the-Loop-Ansatz (ViL) [Boc08]. Dabei wird das reale Versuchsfahrzeug in eine Verkehrssimulation eingebunden. „Der virtuelle Fremdverkehr wird dem Fahrer durch ein optisches am Kopf befestigtes, durchsichtiges Display während der Fahrt realitätsnah eingeblendet“ [Boc08]. Somit wird es möglich, ähnlich wie beim SiL mit der virtuellen Verkehrsflusssimulation, Tests in den frühen Entwicklungsphasen von Fahrer-Assistenzsystemen (FAS) durchzuführen, ohne dass dabei eine Gefährdung auftritt. Was die Problematik bei den einzelnen oben eingeführten Testverfahren betrifft, so lässt sich folgendes beobachten:

- Steigende Komplexität stellt besonders hohe Anforderungen an die Testautomatisierung. Dadurch, dass die HiL-Prüfstände nicht unbegrenzt zur Verfügung stehen, müssen zunehmend bei der automatischen oder manuellen Generierung von Testfällen auch zeitliche Aspekte der Ausführung berücksichtigt werden neben den bereits etablierten Überdeckungskriterien. Hierbei gibt es Bestrebungen, die Testsuits parallel an einem HiL-Stand auszuführen, um die Rechnerauslastung zu erhöhen. Nach wie vor ist das Gebiet des modellbasierten Testens im Bereich reaktiver Systeme aktuell. Hier gilt es zunehmend die Interaktion der Umwelt, insbesondere auch des Fahrers mit der zu testenden Funktion zu modellieren und anschließend ausgehend davon Testfälle abzuleiten. Aus wirtschaftlicher Sicht möchte man dadurch einen sehr hohen manuellen Aufwand zur Erstellung und Parametrisierung von Testfällen vermeiden [Sch08].

- Im Bereich des Testens mit realen Messdaten existieren bereits zahlreiche proprietäre und kommerzielle Lösungen zur automatisierten Ausführung des Testablaufs [Ise08]. Herausforderungen, die in diesem Gebiet anstehen, betreffen, angesichts der Größe der zu verwaltenden Datenmengen, das Testdatenmanagement. Ebenfalls sind in diesem Bereich auf einer großen Menge an Messdaten detaillierte Suchvorgänge durchzuführen, um die Menge der für den Testablauf notwendigen Testdaten minimal zu halten. Angesichts strenger gesetzlicher Vorlagen für FAS müssen sämtliche Testdaten, insbesondere aber auch die

realen Messdaten über mehrere Jahre hinweg aufbewahrt werden, was bei deren Größe zu massiven Speicherproblemen führen kann. Im FAS-Bereich kann die Größe einer Messung beispielsweise zwischen 70 und 100 Gigabyte pro Stunde Fahrt betragen.

Wie bereits erwähnt, finden Testvorgänge auf allen Entwicklungsstufen statt. In der Vorentwicklung beziehungsweise in den frühen Entwicklungsphasen wird viel Wert auf Black-Box Methodik [Lig09] auf Modulebene Wert gelegt. Der Entwicklungs- und Qualitätssicherungsprozess beim Zulieferer (Synonym zu Lieferant) muss bestimmte Standards erfüllen wie beispielsweise Automotive SPICE [VDA10a]. Aus diesem Grund müssen die Lieferanten Tests auf allen möglichen Abstraktionsebenen der automobilen Software durchführen, angefangen bei Unit-Tests bis hin zum Komponententest. Hierbei wird nicht nur die Black-Box Methodik herangezogen, sondern verstärkt auch die White-Box Methodik, wobei in den Standards festgelegt ist, welche Überdeckungskriterien beispielsweise eine Assistenzfunktion erfüllen muss, damit sie von dem OEM übernommen wird.

1.2 Zielsetzung

Aus den bisherigen Ausführungen wird die Komplexität des automobilen Software-Entwicklungsprozesses ersichtlich. Aus Sicht des Testens führt dies zur folgenden Problematik. Betrachtet man einzelne Entwicklungsphasen wie zum Beispiel die Vorentwicklung, so stellt man folgende hierarchische Struktur fest (Abbildung 1.6). Mitarbeiter jeder Abteilung sind an diversen Entwicklungsprojekten beteiligt. Jedes dieser Projekte hat mit der Entwicklung einer bestimmten Teil-Funktionalität eines FAS zu tun, wie beispielsweise Umfeldwahrnehmung, Sensordaten-Fusion oder Situationsanalyse. Innerhalb eines jeden Projekts können ebenfalls Subprojekte ablaufen, die meistens mit den Varianten einer Teil-Funktionalität zu tun haben. So kann beispielsweise ein Umfeldwahrnehmungsmodul sowohl für eine Lane-Departure-Warning (LDW) [Fed10] Funktion als auch für den Spur-Wechsel-Assistent (SWA) entwickelt werden [DJG03]. Innerhalb eines jeden Subprojekts werden die bereits erwähnten Testverfahren angewendet.

Die Untersuchung hat gezeigt, dass die Testspezifikationen innerhalb der einzelnen Projekte und für einzelne Testverfahren im höchsten Maß uneinheitlich sind. Dies betrifft primär die Spezifikation der Testdaten, aber auch solche Aspekte wie die Definition von Soll-Verhalten, die oftmals nur im Kopf eines Entwicklers stattfindet und nicht formal spezifiziert wird. Dies führt dazu, dass zum einen der Austausch und somit auch die Wiederverwendung der Testspezifikationen zwischen den Projekten unterschiedlicher Abteilungen schwer möglich sind. Zum anderen findet aus diesem Grund eine mehrfache Definition von Testspezifikationen statt.

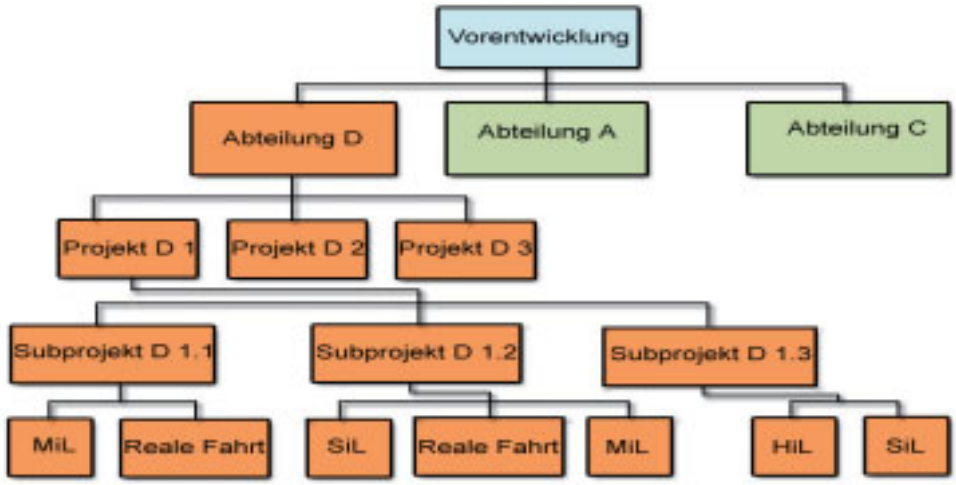


Bild 1.6: Struktur einer Entwicklungsphase am Beispiel der Vorentwicklung

Ein anderer Aspekt betrifft die zumindest teilweise Wiederverwendbarkeit von Testspezifikationen für verschiedene Testverfahren innerhalb eines Subprojekts. In der aktuellen Situation wird für jedes einzelne Testverfahren eine separate Testspezifikation definiert, was einen hohen Zeitaufwand mit sich bringt und zum anderen beispielsweise die Vergleichbarkeit der Testergebnisse erschwert, die durch die Anwendbarkeit unterschiedlicher Testmethodiken erzielt wurden. Es ist also wünschenswert, dass die projektspezifischen Testspezifikationen „aus einer Hand“ oder aus einer einheitlichen Darstellung automatisch erzeugt werden. In der Abbildung 1.7 ist beispielhaft die Spezifikation eines Testfalls für die reale Testfahrt dargestellt. Wie man sehen kann, sind Kataloge, in welchen solche Testfälle verwaltet werden, bestenfalls tabellenartig oder als strukturierter Text darstellbar. Dort werden in der Spalte „Testschritt“ einzelne Testaktionen, also Fahrmanöver, beschrieben. Es werden ebenfalls zugehörige Parameter definiert (z.B. Geschwindigkeit von Eigen- und Fremdfahrzeugen, diverse Abstände usw.). In der Spalte „Ergebnis (Soll)“ wird das erwartete Ergebnis spezifiziert, während in die Spalte „Ergebnis (Ist)“ das tatsächlich erzielte Testergebnis hineingeschrieben wird. Die Spezifikation von Soll-Werten ist also in natürlicher Sprache gegeben. Außerdem ist die Beschreibung von Fahrszenen ebenfalls in natürlicher Sprache, was dazu führen kann, dass die Menge der dort erwähnten Parameter vom Testfall zu Testfall variiert, weil man beispielsweise vergessen hat, bestimmte Parameter zu erwähnen. Die Formalisierung ist somit nicht gegeben. Ebenfalls kann eine solche Beschreibung zu Unklarheiten führen, wie zum Beispiel unpräzise Formulierung dessen, was mit dem Eigenfahrzeug (EGO) nach den durchgeführten Fahrmanövern geschehen soll. Es kann also dazu kommen, dass die Ingenieure und Testfahrer

Szenario 4 ISO15623_6.5.2_1 - Objekttrennung Objekt hinter Objekt 1
Test-Status: draft
Testziel: Nachweis der Fähigkeit zur Objekttrennung
Testdurchführung: Folgefahrt hinter zwei Objekten mit konstanter Geschwindigkeit.
Aufzeichnung:

Testschritt	Ergebnis (Soll)	Ergebnis (Ist)	Bewertung
Das Testobjekt „klein" (Testobjekt A) fährt mit konstanter Geschwindigkeit $v_{target\_klein} == v_{target\_groß} == 36km/h$ in einem Abstand von T=0,4s (d = 4m) hinter dem Testobjekt „groß" (Testobjekt B). Das Ego-Fahrzeug fährt mit konstanter Geschwindigkeit $v_{ego} = v_{target} = 36km/h$ in einem Abstand T=1.5s (d = 15m) zum Testobjekt „groß" bzw. mit einem Abstand T=1.1s (d=11m) hinter Testobjekt „klein"	Die Testobjekte werden während der Folgefahrt nicht verloren.		
Das EGO-Fahrzeug beschleunigt hinter den Objekten und nähert sich dem Testobjekt „klein" bis auf einen Abstand d_1 an bzw. bis zur Ausgabe einer Warnung und schert nach links aus.	Auslösung einer Warnung.		

Bild 1.7: Beispielhafte tabellenartige Testfallbeschreibung für reale Testfahrten [IAV05]

außerhalb der Entwicklungsabteilung die Spezifikation nicht eindeutig verstehen können.

Der dritte Aspekt betrifft die Formatgebundenheit der Testspezifikationen. Betrachtet man beispielsweise den Bereich der virtuellen Simulation (SiL) in der Vorentwicklung, so existiert dort eine XML-basierte Fahrszenen-Beschreibungsnotation. Dieses Format ist proprietär. Dies führt dazu, dass zum Beispiel ein Zulieferer, der eine andere Umgebung als die in der Vorentwicklung eingesetzte zur virtuellen Simulation verwendet oder möglicherweise gar keine, damit eine komplett neue Testspezifikation für eigene Zwecke definiert. Somit kann es passieren, dass die in der Vorentwicklung definierten Testfälle vom Lieferanten aus den benannten Gründen unberücksichtigt werden. Das führt zum Verlust von Testwissen im gesamten Prozess und möglicherweise auch zur Minderung der Software-Qualität.

Letztlich muss der Aspekt der automatisierten Testausführung betrachtet werden. Dadurch, dass bei bestimmten Testverfahren die Testspezifikationen nicht formal definiert sind, entsteht

die Notwendigkeit, diese in eine formale Darstellung zu überführen, bevor ein automatisiert gesteuerter Testvorgang gestartet werden kann. Während des letzteren wird die formale Testspezifikation von einer dedizierten Automatisierungskomponente ausgeführt. Dieser Aufwand ist zum größten Teil manuell und er besteht beispielsweise in der Programmierung und Parametrisierung entsprechender Test-Skripte.

In dieser Arbeit soll eine systematische Methodik (ein Vorgehensmodell) zur formalen Definition von Testspezifikationen für eingebettete reaktive Systeme vorgestellt werden. Die Methodik basiert auf dem Grundgedanken der Modell-getriebenen Software-Entwicklung (MDSD) [SVEH07], bei der es darum geht, aus formalen und plattformunabhängigen Modellen mittels Verfeinerungsschritten und der automatisierten Transformation in ein bestimmtes Zielformat lauffähige Applikationen zu erzeugen. Mit dem vorgestellten Ansatz soll folgendes erreicht werden:

- Vereinheitlichung der Testspezifikationsbeschreibung: Unter der Berücksichtigung der komplexen Organisationsstruktur eines Unternehmens soll eine minimale gemeinsame Basis für alle laufenden Projekte und eingesetzten Testverfahren geschaffen werden, die es erlaubt komplette Testspezifikationen oder ihre Teile untereinander, also Projektgrenzen übergreifend innerhalb einer Entwicklungsphase oder gar über die Entwicklungsphasen hinweg, auszutauschen. Dies soll den Grad der Wiederverwendbarkeit bereits im Rahmen anderer Projekte erzeugter Testspezifikationen erhöhen und somit letztendlich zu Zeiteinsparungen bei deren Definition führen.

- Formatunabhängigkeit der Testspezifikationsbeschreibung: Die erzeugten Testspezifikationen sollen formatunabhängig sein. Dies bedeutet, dass sie weder als elektronische Dokumente noch als sonstige gängige Formate wie zum Beispiel XML [BPSM$^+$10] verwaltet werden. Vielmehr geschieht die Verwaltung von Testspezifikationen in einem (Meta)Modell. Damit wird vermieden, dass die einmal als formatspezifisch definierten Testspezifikationen zum späteren Zeitpunkt manuell in ein anderes Format umgewandelt werden müssen, weil beispielsweise die Testautomatisierungsumgebung ausgetauscht wurde. Die Formatunabhängigkeit bietet ebenfalls mehr Möglichkeiten, den Verständlichkeitsgrad einer Testspezifikation für Dritte zu erhöhen. Gleichzeitig sei an dieser Stelle betont, dass trotz der Formatunabhängigkeit die Testspezifikationen formal sind. Damit wird eine automatisierte Umwandlung dieser in bestimmte Zielformate möglich, die es erlauben, beispielsweise einen automatisierten Testvorgang zu starten. Im gewissen Sinne versucht man damit auch „die semantische Lücke“ [Mü07] zwischen dem Entwickler und Tester in den fortgeschrittenen Entwicklungsphasen eingebetteter reaktiver Software zu schließen, indem

Teile solcher formaler Testspezifikationen als Kommunikationsgrundlage zwischen diesen dienen können. Die Lücke ensteht dadurch, dass der Tester zwar ein breites Fachwissen über diverse Testverfahren besitzt, dafür sich aber weniger mit den Anforderungen an das zu testende System (engl. System under Test: SUT) im Vergleich zum Entwickler auskennt. Dies kann dazu führen, dass die erzeugten Testspezifikationen nicht alle Funktionsbereiche des SUT abdecken.

Die Formatunabhängigkeit wird außerdem als ein wichtiger Beitrag zur Vereinheitlichung der in einem Industrieunternehmen stattfindenden Testvorgänge gesehen.

- Mehrplattform-Fähigkeit der Testspezifikationsbeschreibung: Um die Wiederverwendbarkeit der Testspezifikationen auch innerhalb einzelner Projekte zu erhöhen, sollen diese auf mehrere Zielformate abbildbar sein. Dies wird durch die Transformation realisiert. Die Transformation läuft gewöhnlich regelbasiert ab und verlangt daher bestimmte vorbereitende Schritte, die durch den Domäneningenieur eingeleitet werden. Der Domainingenieur ist hauptsächlich mit der Formalisierung der Testdomäne beschäftigt. Er stellt bereit und pflegt außerdem die komplette Infrastruktur, die zur Definition, Verwaltung und Transformation von Testspezifikationen notwendig ist. Der Entwicklungsingenieur ist hingegen unmittelbar mit der Entwicklung des eingebetteten Systems beschäftigt. Im Weiteren werden die Begriffe „Entwicklungsingenieur“, „Entwickler“ und „Domänenexperte“ als Synonym verwendet.

- Schaffung der notwendigen Voraussetzung zum Testwissenstransfer zwischen den frühen Entwicklungsphasen eingebetteter reaktiver Systeme und den nachgelagerten. Die technische Infrastruktur, welche als Folge der Methodikanwendung entsteht, soll für die nachgelagerten Entwicklungsphasen zugreifbar sein. In diesen Phasen können ganze Testspezifikationen oder deren relevante Teile in die für die dort eingesetzten Testverfahren spezifischen Formate umgewandelt werden. Somit bleibt das Testwissen nicht nur im Unternehmen vorhanden, sondern es wird auch von einer Entwicklungsphase in die nächste übergeben, was zweifelsohne zur Steigerung der Software-Qualität führt.

- Die Methodik soll nahtlos in den täglichen Entwicklungsprozess integrierbar sein.

Als Anwendungsgebiet der entwickelten Methodik soll das Gebiet der Fahrer-Assistenzsysteme gelten.

1.3 Vorgehensweise

Vor der Konzeption der Methodik muss eine eingehende Untersuchung der Anwendungsdomäne, nämlich der Elektronik-Entwicklung entlang der Entwicklungsprozess-Kette durchgeführt werden. Zunächst soll aber die Funktionsweise der SiL, MiL, HiL Techniken sowie der realen Testfahrten näher beleuchtet werden. Während dieser Untersuchung sollen primär die bereits eingesetzten Testverfahren analysiert werden, um anschließend ihre Relevanz für die vorliegende Arbeit zu ermitteln. Ebenfalls gilt es, die grundlegenden und zum Teil gravierenden Unterschiede zwischen den frühen und den nachgelagerten Entwicklungsphasen herauszuarbeiten. Schließlich müssen Defizite bei den vorhandenen Testverfahren ermittelt werden. Dies geschieht im Kapitel 2. Im Kapitel 3 geht es darum, basierend auf den im vorhergehnden Kapitel ermittelten Defiziten sowie den im Kapitel 1 definierten Zielen dieser Arbeit, die wissenschaftlichen Fragestellungen zu erarbeiten. Anschließend gilt es zu untersuchen, ob es im Bereich des Testens bereits Ansätze zur Lösung dieser Fragestellungen gibt, beziehungsweise inwieweit diese zu deren Lösung unter den gegebenen Rahmenbedingungen geeignet wären. Die Untersuchung soll sowohl im Bereich des Modell-basierten Testens stattfinden als auch im allgemeinen Testbereich. Anschließend wird im Kapitel 4 die Methodik vorgestellt. Dabei wird das entwickelte Vorgehensmodell erläutert. Im Kapitel 5 gilt es, die im Kapitel 4 vorgestellte Methodik auf die Testvorgänge in einer Vorserien-Entwicklungsabteilung anzuwenden. Hierbei soll im konzeptuellen Bereich eine Metamodell-Definition für die FAS-Anwendungsdomäne geschehen. Im praktischen Bereich muss eine entsprechende Infrastruktur zur Verwaltung von Testspezifikationen geschaffen werden. Anschließend muss im Kapitel 6 untersucht werden, inwieweit sich die entwickelte Methodik in den täglichen Entwicklungsprozess integrieren lässt und vor allem wie die Entwickler mit dieser zurechtkommen. Dies soll ebenfalls am Beispiel einer Vorserien-Entwicklungsabteilung geschehen. Schließlich muss das Erreichte zusammengefasst werden, um anschließend weitere Entwicklungsrichtung aufzuzeigen. Dazu dient das Kapitel 7.

Kapitel 2

Bestandsaufnahme in der Anwendungsdomäne

In diesem Kapitel werden die in der automobilen Software-Entwicklung verwendeten Testverfahren eingehend beleuchtet. Testverfahren ist ein Verfahren, das dazu eingesetzt wird, Software-Fehler in dem zu testenden System zu finden. Die Begriffe „Testverfahren“ und „Testtechnik“ werden in dieser Arbeit synonym verwendet. Ein Testvorgang ist die konkrete Anwendung eines bestimmten Testverfahrens. Zu Beginn wird eine kurze Einführung in die Fahrzeugelektronik mit dem Schwerpunkt Steuergeräte-Software gegeben. Anschließend wird der beobachtete Ist-Testprozess im automobilen Bereich betrachtet. In den darauffolgenden Unterabschnitten werden die einzelnen Testtechniken nach folgender Struktur eingeführt. Als erstes wird immer das entsprechende Testverfahren dargestellt. Daraufhin folgen ein oder mehrere konkrete Beispiele aus der Anwendungsdomäne. Schließlich wird das Verfahren bewertet.

2.1 Fahrzeugelektronik

Um die in diesem Kapitel vorgestellten Verfahren verstehen zu können, ist ein Grundverständnis der Software-Architektur im modernen Fahrzeug notwendig. Zu Beginn wird der Begriff „Funktion“ eingeführt. Abweichend von der in der Informatik üblichen Definition für die (programmiersprachliche) Funktion wird die automobile Funktion als eine komplexe, ausführbare Software-Komponente definiert, die eine bestimmte Funktionalität im Fahrzeug realisiert. Diese kann auf mehrere Steuergeräte im Fahrzeug verteilt sein und mit anderen Funktionen über Bus-Systeme interagieren. Um Missverständnisse zu vermeiden, wird im weiteren Verlauf für die klassische Definition der Begriff „programmiersprachliche Funktion“ verwendet, während für die

hier eingeführte Definition „Funktion“ oder „Fahrzeugfunktion“ als synonym gelten. Ein Fahrzeug ist in mehrere Subsysteme aufgeteilt (Abb. 2.1). Jedes dieser Subsysteme wird durch mehrere

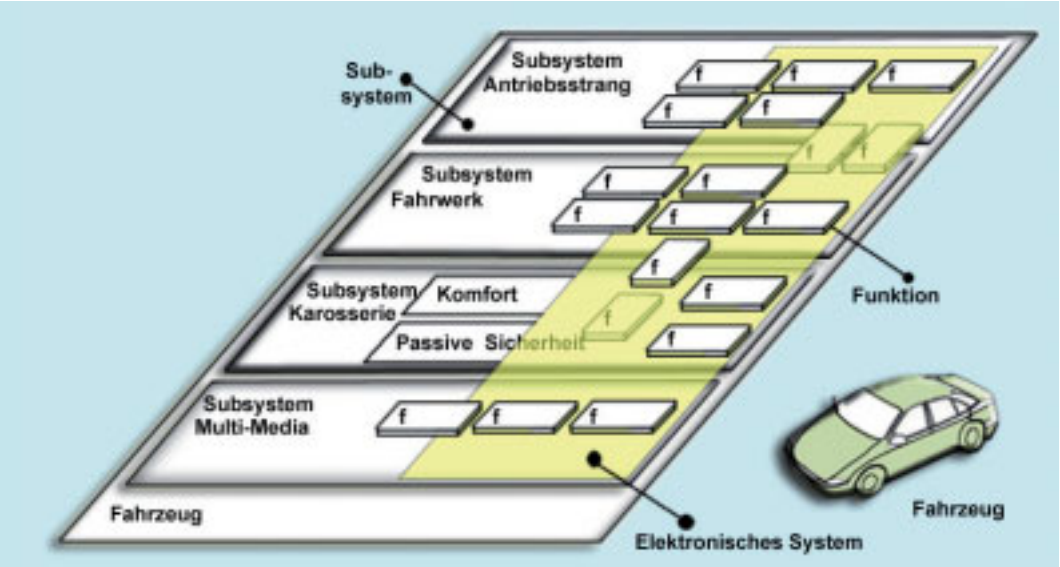


Bild 2.1: Subsysteme im Fahrzeug [SZ06]

Funktionen realisiert. Diese Sichtweise wird als logische Systemarchitektur bezeichnet [SZ06], da auf diesem Abstraktionsniveau noch nicht feststeht, wie die Funktionen auf einzelne Steuergeräte verteilt sind. Als „technische Systemarchitektur“ [SZ06] wird in der Literatur hingegen die bereits erwähnte technische Realisierung der Verteilung von Fahrzeugfunktionen auf verschiedene Steuergeräte bezeichnet. Da es sich bei vielen Kraftfahrzeugsystemen um Steuerungs- und Regelungssysteme handelt, soll im Weiteren deren logische Architektur vorgestellt werden (Abb. 2.2). Von entscheidender Bedeutung ist bei den Steuerungssystemen der Einfluss beziehungsweise die

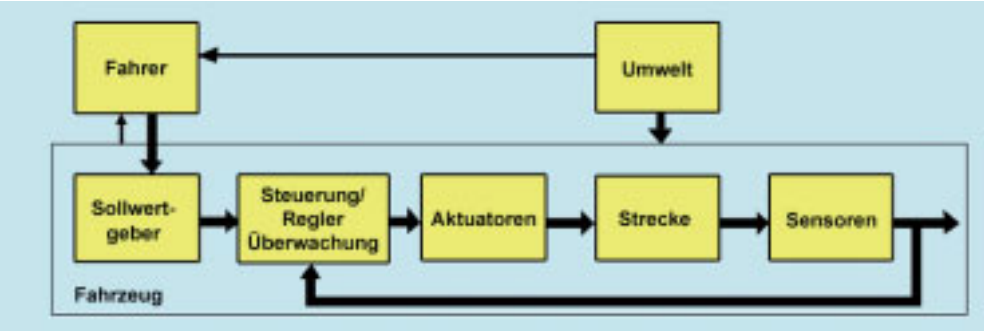


Bild 2.2: Logische Architektur der Steuerungs-und Regelungssysteme [SZ06]

Interaktion des Systems mit der Umwelt. Unter Umwelt werden in diesem Kontext sowohl die von einer höheren Gewalt ausgelösten Ereignisse wie beispielsweise Niederschlag verstanden als auch von den anderen Objekten ausgelöste Aktionen wie beispielsweise bestimmte Fahrmanöver der Testfahrzeuge.

Ein anderer wichtiger Aspekt ist die Interaktion des Menschen mit dem Fahrzeug (engl. Human-Machine-Interaction oder auch Human-Computer-Interaction [HBC+96]). Durch visuelle, akustische und haptische Sinneskanäle nimmt der Mensch seine Umgebung wahr und reagiert darauf, indem er lenkend in das Maschinenverhalten eingreift. Über die benannten Wahrnehmungskanäle empfindet er nicht nur die Umweltereignisse, sondern ebenfalls das Maschinenverhalten. Es ist also ein Kreislauf zwischen Mensch, Maschine und Umgebung zu beobachten.

Zur Umgebungswahrnehmung im Fahrzeug werden Sensoren eingesetzt [Har09]. Diese sind Einheiten, die die zu messende physikalische Größe in elektrische Signale umwandeln. Im Bereich der FAS werden Sensoren beispielsweise zur Objektdetektion verwendet (so genannte Radar- und Lasersensoren). Sensoren können aber auch Eigenwerte des Fahrzeugs messen wie beispielsweise Raddrehzahl oder die Fahrzeuggeschwindigkeit.

Die Aktoren [Jan04] sind Einheiten, die elektrische Signale in nichtelektrische umwandeln, also in mechanische Arbeit zum Beispiel. Im FAS-Bereich werden Aktoren dazu eingesetzt, um beispielsweise aktiv in die Lenkung eines Fahrzeugs einzugreifen.

Schließlich gilt es den Kommunikationsaspekt zwischen den oben definierten, im Fahrzeug verbauten Komponenten zu betrachten. Die Kommunikation findet über Bus-Systeme statt. Das seit 1983 eingesetzte Bus-System ist der von Bosch für die Vernetzung von Steuergeräten in Automobilen entwickelte Controller-Area-Network-Datenbus (CAN-Datenbus) [Bos10]. Der

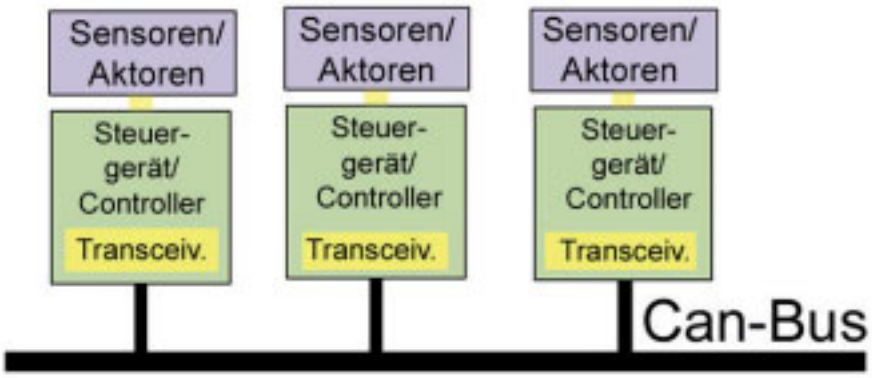


Bild 2.3: Vernetzung der Steuergeräte über CAN-Bus

CAN-Datenbus (Abb. 2.3) sorgt für einen digitalen Austausch zwischen Sensoren, Aktoren sowie Steuergeräten und gewährleistet, dass mehrere Steuergeräte die Informationen eines Sensors verarbeiten und ihre Aktoren entsprechend ansteuern können. Das Steuergerät hat also die Aufgabe, die vom Fahrer eingestellten beziehungsweise festkodierten Soll-Werte mit den aktuellen Ist-Werten des Fahrzeugumfelds zu vergleichen und den physikalischen Prozess entsprechend nachzuregeln.

Abschließend sind in der Abbildung 2.4 beispielhaft drei Systeme aus dem Bereich Fahrerassistenz dargestellt (ESP, ABS und ACC), deren Darstellung an der in der Abbildung 2.2 eingeführten logischen Architektur orientiert ist.

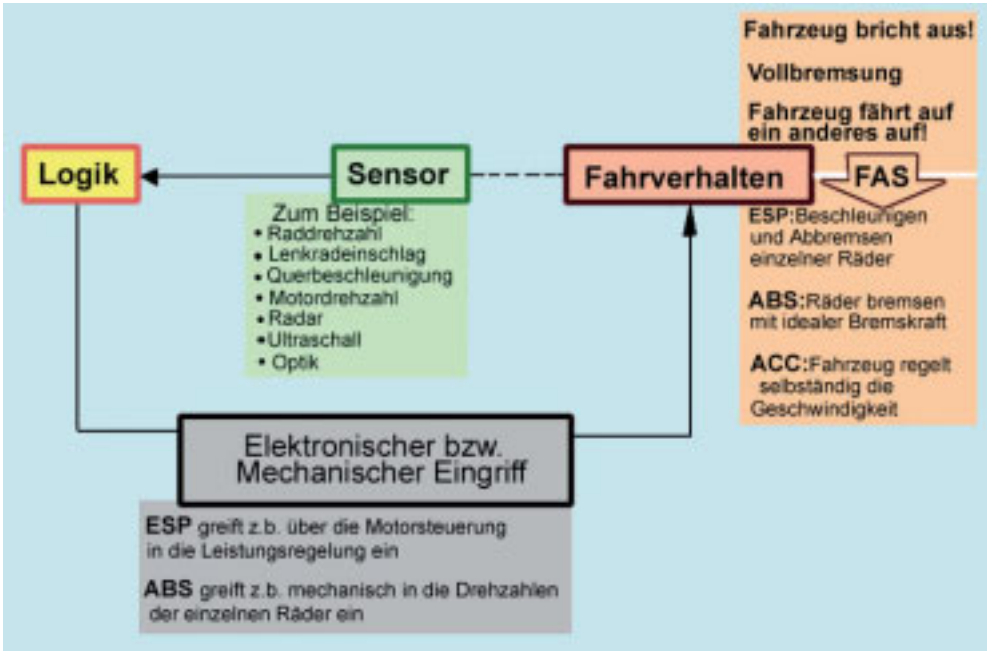


Bild 2.4: Funktionsweise ABS, ESP, ACC

Während solche Systeme wie ABS und ESP ausreichend bekannt sind, ist Adaptive Cruise Control noch nicht sehr lange auf dem Markt vorhanden. Aus diesem Grund wird im Folgenden seine Funktionsweise erklärt. Das Adaptive Cruise Control stellt eine Weiterentwicklung des Tempomats dar. Der Fahrer stellt die gewünschte Geschwindigkeit am Tempomat ein. Ein im Fahrzeug eingebauter Fernbereichsradar registriert dabei das Vorhandensein eines vorausfahrenden Fahrzeugs und zeigt dies am Kombibrett dem Fahrer an. Der Fahrer wählt daraufhin mittels eines Bedienhebels die gewünschte Distanz, welche vom vorausfahrenden Fahrzgeug eingehalten werden soll. Das Assistenzsystem hält den vom Fahrer gewünschten Abstand ein, indem es sich der Geschwindigkeit des vorausfahrenden Fahrzeugs durch direkten Eingriff in das Bremssystem sowie die Motorsteuerung anpasst. Das ACC-System ist für den Einsatz auf den Autobahnen

konzipiert, bei denen ein sehr dichter Verkehr vorkommt, welcher dem Fahrer eine erhöhte Konzentration abverlangt.

2.2 Der Ist-Testprozess

Um die Einsatzbereiche der in der Einleitung erwähnten Testverfahren strukturiert darzustellen, soll deren Vorkommen entlang des bei AUDI AG untersuchten und für diese Arbeit gültigen Ist-Prozesses angeordnet werden (Abb. 2.5). In den frühen Entwicklungsphasen (Vorentwicklung)

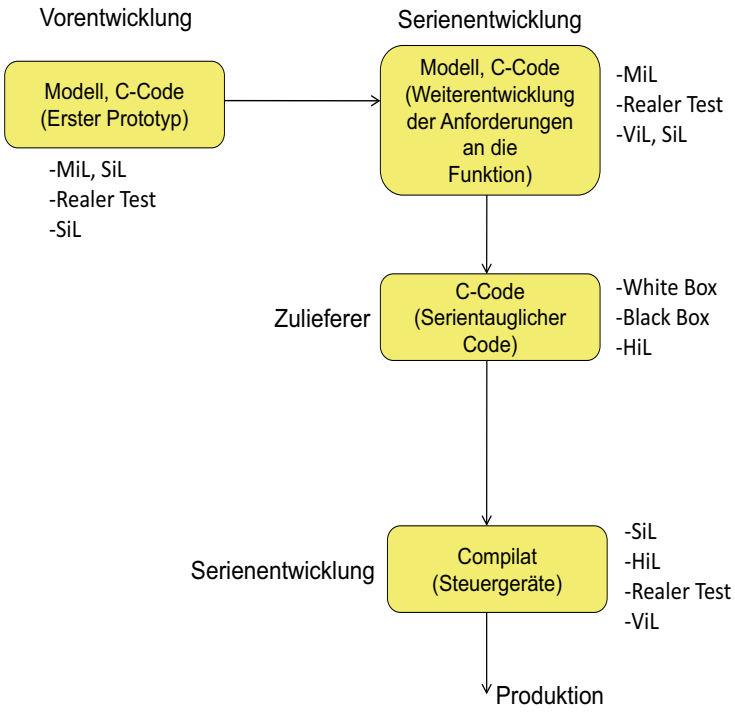


Bild 2.5: Der Ist-Testprozess

wird weniger Wert auf White-Box-Tests im Sinne von [Lig09] als auf die so genannte Black-Box-Methodik gelegt. In dieser Phase sind solche Verfahren angesiedelt wie Model-in-the-Loop und Rapid-Prototyping beziehungsweise Test im realen Fahrzeug (synonym zum Begriff „realer Test“). Ginge es beispielsweise ganz nach dem modellbasierten Entwicklungsprozess von automobilen Steuergeräten [Lam06], dann würde aus dem modellierten Systemverhalten über einen Transformationsschritt mit geringen Anpassungen ein auf der Serien-Zielarchitektur einsetzbarer Code erzeugt. Dies ist oftmals nicht der Fall. Die Modelle werden zweifelsohne dazu eingesetzt, um

bestimmte Grundfunktionalitäten und „erste zündende Ideen“ zu überprüfen oder um auf dem prototypischen Steuergerät mittels Rapid-Prototyping [Geb00] erste Testfahrten durchzuführen.

Im nächsten Entwicklungsschritt innerhalb der frühen Entwicklungsphase ist allerdings der Sachverhalt zu beobachten, dass die Funktion in C-Code mittels eigens dazu umgesetzter Frameworks entwickelt wird. An dieser Stelle haben sich solche Verfahren wie SiL mit Hilfe der virtuellen Simulation, aber auch mit Hilfe automatisierter Labor-Testabläufe mit großen Mengen an vorher aufgezeichneten Messdaten etabliert. Weiterhin werden reale Testfahrten und neuartige Testtechniken wie ViL eingesetzt. Neben dem prototypischen Code wird von der Vorentwicklung an die Serienentwicklung vor allem ein Anforderungsdokument übergeben.

Die Serienentwicklung macht eine Weiterentwicklung der von der Vorentwicklung erhaltenen Funktionalität, sie wendet solche Testverfahren wie MiL, realer Test, SiL sowie ViL an und ergänzt am Ende das von der Vorentwicklung erhaltene Anforderungsdokument mit eigenen Anforderungen.

Die Zulieferer, die den prototypischen Stand der Funktion von der Serie samt dem ergänzten Anforderungsdokument bekommen, werden von den Automobilherstellern zunehmend verpflichtet, ihre Software-Entwicklungsprozesse nach bestimmten Standards durchzuführen. Der Vorteil solcher Vorgehensweise ist, dass sich die Prozessqualität klar bemessen lässt. Konkret bedeutet dies, dass man für eine verwendete Vorgehensweise zu einer Einschätzung kommen möchte, ob diese dem derzeitigen Stand der Technik entspricht. Der gängigste Standard ist das bereits erwähnte Automobile Software Process Improvement and Capability Determination (Automotive SPICE) [VDA10a]. Dieser besteht aus zwei Dimensionen: Prozessdimension und Reifegraddimension (Abb. 2.6). Alle Prozesse in der Prozessdimension entsprechen der ISO 12207 Norm, welche um die für das automobile Umfeld spezifischen Ergänzungen erweitert ist. Die Reifegrad-Dimension beinhaltet sechs Reifegrade, die in der ISO 15504 festgelegt sind [VDA10a]:

- „Level 0: Incomplete process. The process is not implemented, or fails to achieve its process purpose. At this level, there is little or no evidence of any systematic achievement of the process purpose.
- Level 1: The implemented process achieves its process purpose.
- Level 2: Managed process. The previously described Performed process is now implemented in a managed fashion (planned, monitored and adjusted) and its work products are appropriately established, controlled and maintained.
- Level 3: Established process. The previously described Managed process is now implemented using a defined process that is capable of achieving its process outcomes.

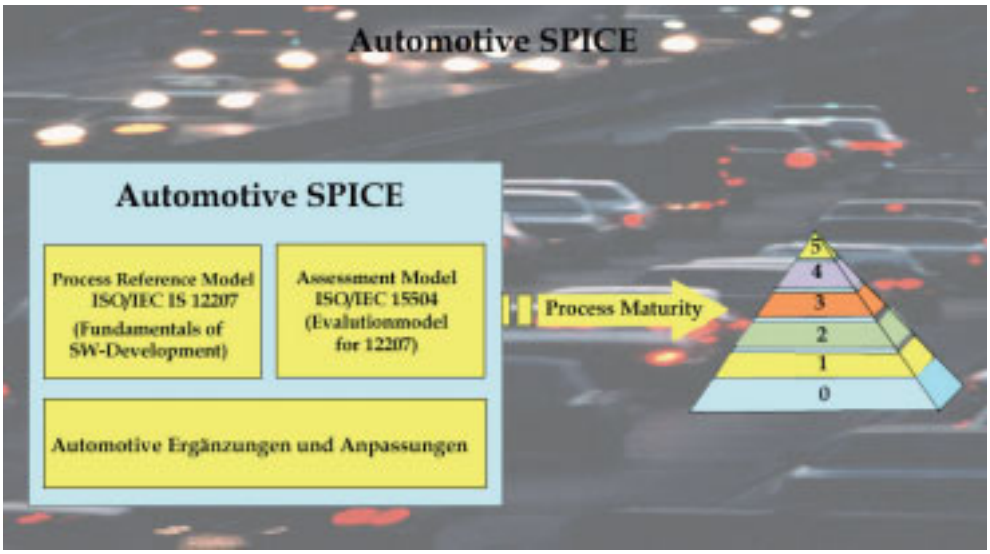


Bild 2.6: Automotive SPICE: Grundsätzliche Bestandteile [VDA10b]

- Level 4: Predictable process. The previously described Established process now operates within defined limits to achieve its process outcomes.
- Ebene 5: Optimizing process. The previously described Predictable process is continuously improved to meet relevant current and projected business goals.“

Was die Testaktivitäten betrifft, so beinhaltet vor allem die Prozessdimension mehrere Prozessgruppen, innerhalb von welchen auch ein Prozess namens „Software Testing“ durch solche Größen wie Prozessname, Prozessziel und Prozessausgang beschrieben wird (Abb. 2.7).

Der Zulieferer (Synonym zu Lieferant) nimmt das bereits erwähnte Lastenheft von der Serienentwicklung entgegen und stimmt es mit der Serien-Abteilung ab, indem er Kommentare an die Anforderungen bezüglich deren Erfüllbarkeit schreibt. Letztendlich entsteht daraus ein Pflichtenheft, anhand welchem der Lieferant die Funktion entwickelt. Nach dem SPICE-Level 2 dürfen der Tester und der Entwickler nicht dieselbe Person sein. Somit muss sich beim Lieferanten eine eigene Entwicklungsabteilung mit der Zusammenstellung des Pflichtenhefts befassen, während die Testvorgänge von dedizierten Test-Teams durchgeführt werden. Am Ende der Entwicklungsphase wird kein Code, sondern ein Kompilat an die Serien-Entwicklungsabteilung übergeben, das von dieser auf die Serien-Steuergeräte überspielt werden kann. An dieser Stelle wird ein gravierender Unterschied zur Vorentwicklung deutlich. In dieser gibt es keinen klaren Unterschied zwischen Testingenieur und Entwicklungsingenieur: Es sind oft eine und dieselbe Person.

Was den Umfang der beim Lieferanten durchgeführten Testvorgänge betrifft, so ist dieser

3.3.8 ENG.8 Software testing

Process ID	ENG.8
Process Name	Software testing
Process Purpose	The purpose of the Software testing process is to confirm that the integrated software meets the defined software requirements.
Process Outcomes	As a result of successful implementation of this process: 1) a strategy is developed to test the integrated software according to the priorities and categorization of the software requirements; 2) a test specification software test for the integrated software is developed that demonstrates compliance to the software requirements; 3) the integrated software is verified; 4) test results are recorded; 5) consistency and bilateral traceability are established between software requirements and test specification software test including test cases;and 6) a regression test strategy is developed and applied for re-testing the integrated software when a change in software items occur. NOTE 1: The test specification for software testing includes the test design specification, test procedure specification and test case specifications. NOTE 2: The verification is performed according to the test cases NOTE 3: The test results for software testing include the test logs, test incident reports and test summary reports.

Bild 2.7: Automotive SPICE: Prozess „Software Testing“ [VDA10a]

deutlich höher als noch in der Vorentwicklungsphase. Nach entsprechendem SPICE-Level müssen die Zulieferer zusätzlich zu funktionsorientierten und diversifizierenden Tests (auch bekannt als Black-Box-Testen) [Lig09] ebenfalls strukturorientierte Testmethodiken [Lig09] anwenden (so genanntes White-Box-Testen). Darüber hinaus werden beispielsweise bei kontrollflussorientierten Testmethoden bestimmte Kriterien festgelegt, die ein Testergebnis nach der Auswertung aufweisen muss, damit die Funktion von der Serienentwicklung übernommen wird. Ebenfalls unterscheidet sich die Granularität, auf welcher die Testvorgänge beim Lieferanten stattfinden, von der in der Vorentwicklung. Während in der Vorentwicklung hauptsächlich auf Komponenten- und Funktionsebene getestet wird, so fängt man beim Zulieferer auf der untersten Ebene an, also beispielsweise mit dem Testen einzelner programmiersprachlicher Funktionen (so genannte Unit-Tests [Ham05]). In bestimmten Fällen können die Lieferanten die Funktion bereits am so genannten Einzel-HiL testen, also nicht im Verbund mit anderen Funktionen.

Schließlich erhält die Serien-Testabteilung das fertige Kompilat einer Funktion von dem Lieferanten und führt HiL-Tests durch. Diese finden auf der Systemebene statt. Eine wichtige Aufgabe der Serien-Testabteilung ist es auch, Integrationstests durchzuführen: Es werden also an den HiL-Prüfständen mehrere von unterschiedlichen Lieferanten übergebene Funktionen im Verbund getestet. Die HiL-Testverfahren finden in enger Abstimmung zwischen der Testabteilung und der Serien-Entwicklungabteilung statt. Es ist also auch hier eine klare Rollentrennung in Entwicklungsteams und Test-Teams zu beobachten.

Nicht zu unterschätzen sind ebenfalls so genannte Erprobungsfahrten, die neben HiL-Methodik auch in der Serienentwicklung eine breite Anwendung finden.

2.3 Model-in-the-Loop Testverfahren

In diesem Abschnitt wird die Methodik „Model-in-the-Loop“ (MiL) vorgestellt. Da es sich um den Einsatz der Methode im Bereich der Software-Entwicklung für mechatronische Systeme handelt, muss zunächst der mechatronische modellbasierte Entwicklungskreislauf [Far05] vorgestellt werden (Abb. 2.8). Am Anfang des mechatronischen Entwicklungskreislaufs steht die Idee eines Entwicklers. Anschließend wird diese in einem Modell verfeinert. Der Vorgang, der zur Entstehung eines Modells führt, wird als Modellbildung bezeichnet. Bei der Modellbildung geht es darum, die Details eines bestimmten zu entwickelnden Systems zu einem möglicherweise idealisierten Modell zu abstrahieren. Durch die Idealisierung wird die Aussagekraft des Modells geschwächt. In der Entwicklungsphase, in welcher dieses Verfahren eingesetzt wird, ist ein sehr hoher Detaillierungsgrad noch nicht strikt gefordert. Anschließend folgen die Schritte der Parameteridentifikation für das Modell, deren Analyse, Synthese und Implementierung. Aus der Testsicht ist der Schritt „Experiment“ interessant, denn in diesem wird das eigentliche Testverfahren (MiL) eingesetzt. Dabei werden entweder künstlich generierte oder echte Stimuli in das im ersten Schritt erzeugte Modell eingespeist und am Modell-Ausgang im Sinne eines Blackbox-Tests ausgewertet. Durch den Rapid-Prototyping-Ansatz besteht ebenfalls die Möglichkeit, den vom Modell abgeleiteten Code mit bestimmten Anpassungen auf einer prototypischen Steuereinheit bei der echten Testfahrt zu überprüfen. Wichtig zu erwähnen ist, dass mit dem „Experiment“-Schritt der Entwicklungsvorgang nicht zu Ende ist. Die während des Tests ermittelten Fehler werden in der nächsten Iteration des Schritts „Modellbildung“ berücksichtigt. Man versucht durch den Einsatz der MiL-Methodik die Fehler sehr früh im gesamten Entwicklungsprozess zu entdecken und deren Detektion ins Labor, also in den „off-line“-Bereich zu verlagern, um hohe Kosten der Fehlersuche im Fahrzeug selbst (zum Beispiel während einer Testfahrt) möglichst zu vermeiden.

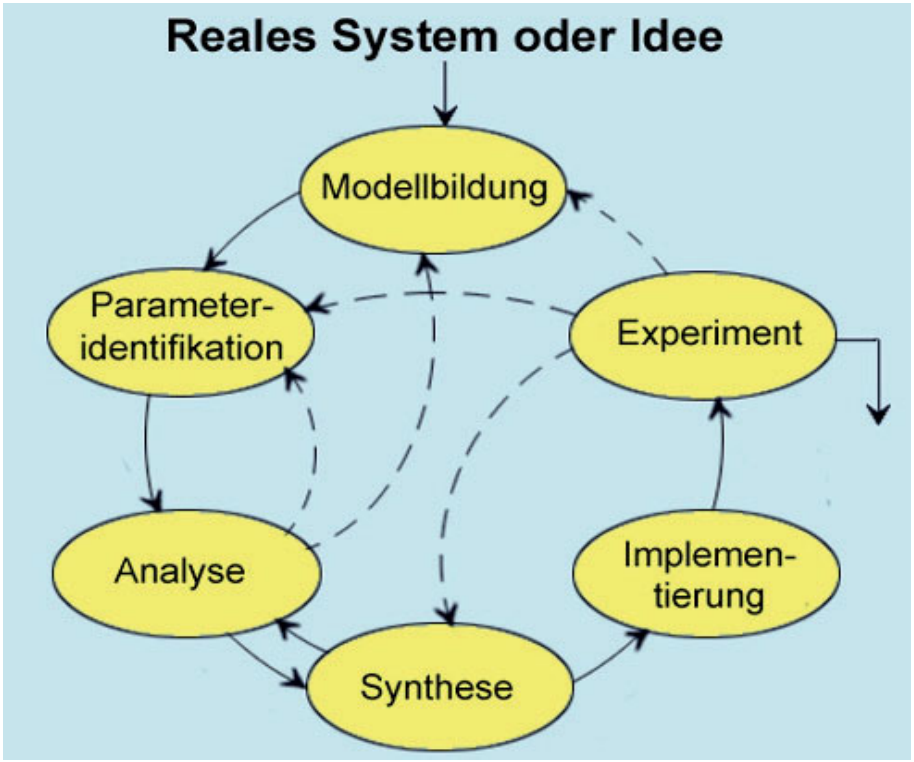


Bild 2.8: Mechatronischer Entwicklungskreislauf [Far05]

2.3.1 Verfahrensbeschreibung

Die MiL-Methodik basiert auf der Annahme, dass in den frühen Entwicklungsphasen noch keine Zielhardware für die Funktion vorliegt. Da die Testläufe vielfach im Labor verlaufen, liegen auch keine vom Sensor gemessenen Daten sowie kein Fahrer- und Fahrzeugverhalten vor. Diese müssen dementsprechend simuliert werden. Dazu werden ebenfalls Modelle erzeugt, welche beispielsweise das Sensorverhalten oder die Fahrzeugdynamik [MW05] simulieren. Bei den MiL-Testverfahren handelt es sich um rückgekoppelte Vorgänge, bei denen die von der Funktion erzeugten Signale zurück in die simulierte Umgebung fließen. In der Abbildung 2.9 ist ein beispielhafter MiL-Testablauf dargestellt. Das mittels Simulink [Mat10b] erstellte Modell einer ACC-Funktion hat mehrere Eingänge. Ein Teil dieser Eingänge bekommt die Signale von einem künstlichen Testdatengenerator. Beispielsweise werden Werte für solche Parameter erzeugt wie Lenkradwinkel oder Position des Bremspedals. Ebenfalls bekommt das ACC-Modell von dem Fahrzeug- und Fahrbahnmodell Werte für diverse Parameter wie zum Beispiel seitliche Beschleunigung oder die Rädergeschwindigkeit. Die Funktion erzeugt in diesem Beispiel am Ausgang folgende Signale „Bremsmoment“ und „Gasmoment“, die im Block „Test Output“

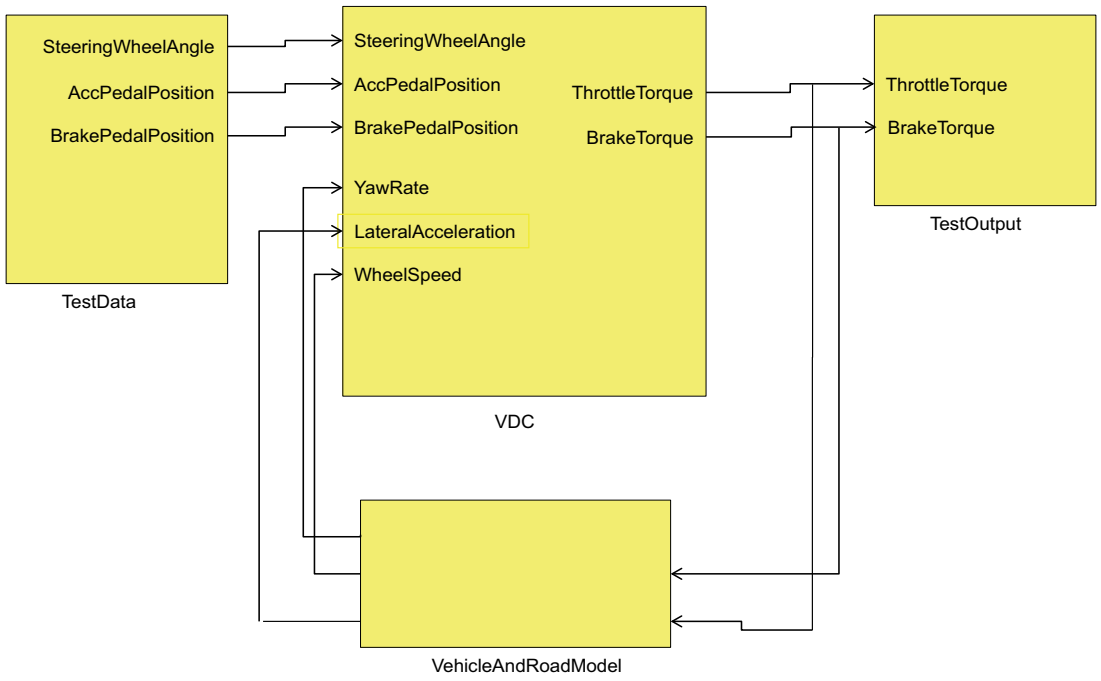


Bild 2.9: Beispielhafter MiL-Testablauf [LBEO04]

gegen bestimmte Zielgrößen ausgewertet werden können. Gleichzeitig handelt es sich um einen geschlossenen Kreis, sodass die am Ausgang erzeugten Signale wiederum in das Fahrzeug- und das Fahrbahnmodell einfließen. Was in diesem Beispiel nicht abgebildet ist, sehr wohl aber auch berücksichtigt werden kann, ist zum Beispiel ein Sensormodell, durch welches die vom Testdatengenerator erzeugten Daten „geschleust“ werden, bevor sie in das Funktionsmodell einfließen. Ein Sensormodell kann beispielsweise dazu dienen, die durch den Testgenerator erzeugten idealisierten Daten so zu transformieren, dass sie denen entsprechen, die von einem echten Sensor detektiert wurden. Dies bedeutet vor allem, dass sie fehlerhaft oder ungenau sein können.

2.3.2 Einsatz in der Anwendungsdomäne

In den frühen Entwicklungsphasen wird das kommerzielle Tool „EXACT“ [Ext10] eingesetzt. Die gesamte Vorgehensweise zur Erstellung, Ausführung und Auswertung der Testfälle ist in der

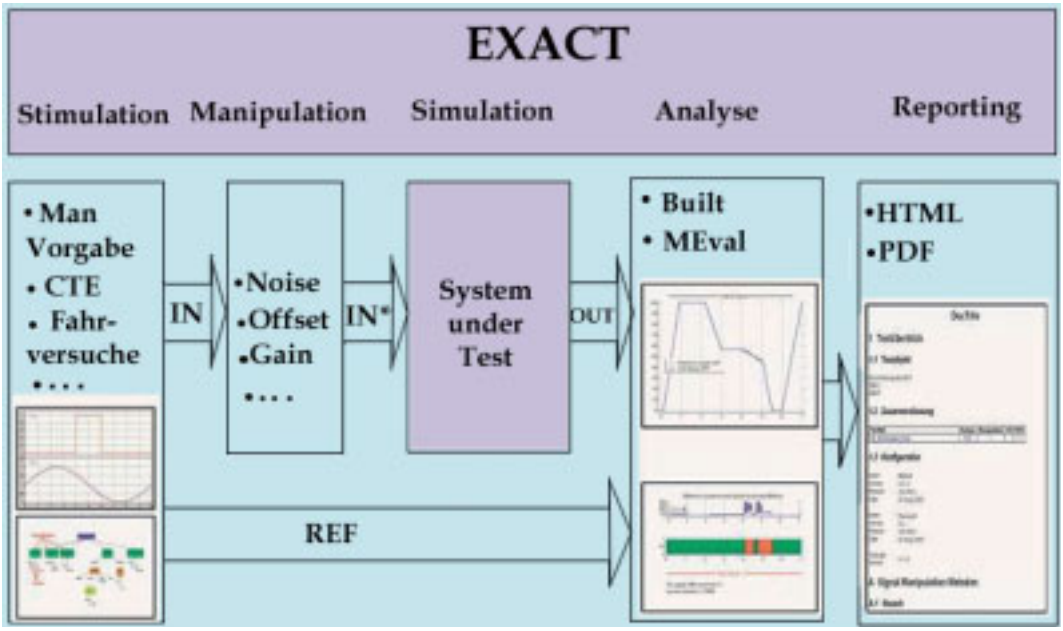


Bild 2.10: Vorgehensweise zur MiL-Testen mit EXACT [Ext08]

Abbildung 2.10 dargestellt. Die Modelle werden dabei in Matlab erstellt. In einem ersten Schritt geht es darum, die einzelnen Stimuli zu bestimmen. Diese können aus diversen Quellen kommen (Fahrversuche oder per Hand definierte Signalverläufe). Außerdem besteht die Möglichkeit, die definierten Signale durch bestimmte Methoden (z.B. Rauschen) zu manipulieren. Die Referenzwerte, also die erwarteten Test-Ergebnisse, werden ähnlich wie die Stimuli definiert. Pro Signal können sogenannte Analysen definiert werden. Diese dienen dazu die Testergebnisse auszuwerten, sie mit den Referenzwerten zu vergleichen und die Ergebnisse im Testreport abzulegen. Schließlich muss die Form des Testreports bestimmt werden. Es geht also darum zu bestimmen, welche Informationen im Report erscheinen sollen und in welcher Form (z.B. HTML [W3C10a] oder PDF [Ado10]).

Interessant ist es in diesem Zusammenhang, die gerade beschriebene Vorgehensweise anhand eines konkreten Beispiels aus dem Bereich der Reglerentwicklung zu betrachten. Hier beginnen die Vorbereitungen zur automatisierten Testausführung mit der Definition von Anforderungen. Diese werden in Word-Dokumenten tabellarisch zusammengefasst. Jede Anforderung ist mit einem eindeutigen Schlüssel versehen. Im nächsten Schritt geht es darum die Testfälle zu spezifizieren. Diese werden mithilfe des Classification Tree Editors (CTE) [GG93] dargestellt (Abb. 2.11) [ATS10]. Die grundsätzliche Idee der Klassifikationsbaum-Methode ist es, zuerst den Eingabedatenraum des Testobjekts nach verschiedenen testrelevanten Gesichtspunkten jeweils getrennt

voneinander in Äquivalenzklassen [Lig09] zu zerlegen, um dann durch die Kombination geeigneter Klassen zu Testfällen zu kommen. Wichtigste Informationsquelle ist dabei die funktionale Spezifikation, in der das geforderte Verhalten des Testobjekts beschrieben ist [We94]. In der Abbildung 2.11 ist ein Beispiel für den Klassifikationsbaum dargestellt. ABS-Funktion wird durch Kategorien Umgebung, Fahrer und Fahrzeugdynamik beschrieben. Für die jeweilige Kategorie werden mehrere Äquivalenzklassen bestimmt: Zum Beispiel die Klassen „Fahrbahn“ und „Straßenverlauf“ für die Kategorie „Umgebung“. Im unteren Bildteil werden dann die Testfälle erstellt, indem die durch den schwarzen Kreis gekennzeichneten Repräsentanten diverser Klassen gewählt werden. Die Klassifikationsbäume sind besonders dann nützlich, wenn viele verschiedene Varian-

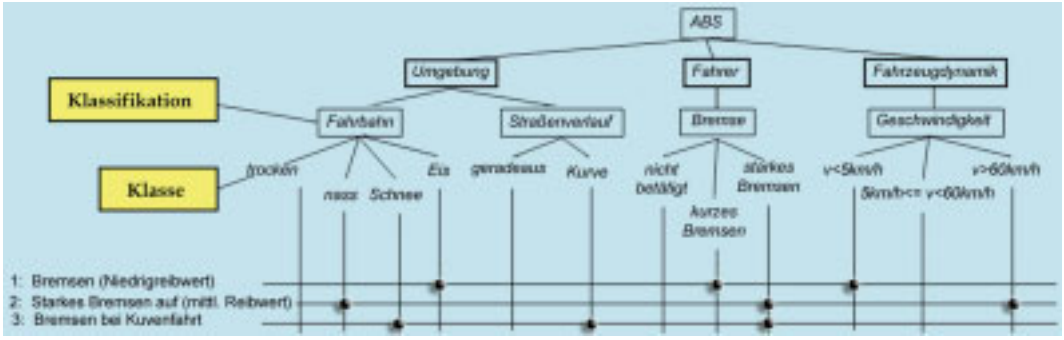


Bild 2.11: Klassifikationsbaum-Methode [ATS10]

ten der Eingangssignale des zu testenden Systems in unterschiedlichen Kombinationen getestet werden müssen. Ein Testfall wird durch die Auswahl bestimmter Werte für die angegebenen Signale bestimmt. Dabei darf aus jedem Intervall genau ein Wert für das Signal gewählt werden. Die Wertebereiche dürfen aber nicht nur durch Intervalle angegeben werden, sondern auch durch diskrete Werte. Die Testfälle werden von oben nach unten angeordnet. Der wesentliche Nachteil dieser Methode ist ihre schwere Handhabung mit steigender Anzahl der Eingangssignale.

Für jeden Testfall ist es möglich, in einem Textfeld bestimmte Metainformationen über beispielsweise den Autor oder den Anforderungsschlüssel zum entsprechenden Word-Dokument abzulegen.

Im nächsten Schritt geht es darum, die Soll-Werte (synonym zu Referenzwerte) für die definierten Testfälle zu erzeugen. Dazu existiert ein Referenzmodell, welches für die im vorigen Schritt definierten Testfälle entsprechende Referenzwerte erzeugt. Deshalb muss vor dem eigentlichen automatisierten Testlauf noch ein Lauf stattfinden, bei dem die Soll-Werte erzeugt werden. Im nächsten Schritt findet die Zuweisung beziehungsweise die Auswahl von Testfällen sowie Referenzdaten für den durchzuführenden Testlauf. Dabei müssen für jeden Testfall entsprechende

Dateien mit den erzeugten Referenzwerten ausgewählt werden. Ebenfalls müssen die Auswertungsmethoden für jedes Signal festgelegt werden. Intern wird in einem Zwischenschritt eine Umwandlung der Testfälle in das Matlab-Format [Mat10a] durchgeführt. Während des Testlaufs wird eine XML-Datei erzeugt, welche anschließend in EXCEL [Mic10b] importiert werden kann. In dieser ist auf eine kompakte Art und Weise die Zuordnung von Testfall zur Anforderung dargestellt sowie in Kurzform das Testergebnis (OK oder nicht OK). Abschließend wird noch ein mehrseitiger Testreport generiert.

2.3.3 Bewertung des MiL auf Relevanz

Das MiL-Verfahren ist eine ausgereifte Methodik, die in der Ideenfindungsphase und vor allem bei der ersten Überprüfung dieser Ideen dem Entwickler eine solide Unterstüzung bietet. Aus der Sicht dieser Arbeit sind allerdings zwei wichtige Aspekte zu betrachten. Zum einen handelt es sich um eine Tatsache, dass bei den entwickelten Modellen die Transformation meist in nur ein bestimmtes Zielformat möglich ist. Die Möglichkeit die Transformation in ein anderes Zielformat durchzuführen als das von dem Hersteller zur Verfügung gestellte, ist entweder nicht gegeben oder sehr aufwändig. Zum anderen liegen die erzeugten ausführbaren Testspezifikationen nur in einem plattformspezifischen Format vor (XML, Matlab-Files). Dabei lässt sich ihre Semantik für Dritte nur schwer erschließen. Offensichtlich wird aus den obigen Ausführungen, dass einzelne Bestandteile der Testspezifikation wie z.B. Testdaten und das Soll-Verhalten zunächst in unterschiedlichen Formaten vorliegen (Soll-Verhalten als elektronisches Dokument und die Testdaten als M-Files. M-File ist hierbei ein proprietäres Matlab-Format). Erst in einem nachgelagerten Schritt werden die maschinenlesbaren Soll-Werte generiert. Offensichtlich ist ebenfalls die beobachtete Lücke zwischen MiL und dem nachgelagerten SiL. Dadurch, dass der aus einem Modell erzeugte Code nur teilweise übernommen wird und teilweise per Hand mit einem dedizierten Framework entwickelt wird, müssen bestimmte Teile der ausführbaren Testspezifikation entweder neu definiert werden oder per Hand aus den bereits erstellten MiL-Testspezifikationen übertragen werden. Hierbei ergibt sich folgende Problematik:

- Neue Generierung von Testdaten: Betrachtet man beispielsweise den FAS-Bereich, so müssen für virtuelle SiL-Simulationen die Fahrszenen komplett neu mit Hilfe eines graphischen Editors definiert werden, denn sie können kaum aus den Klassifikationsbäumen übernommen werden. Mit den letzteren lässt sich beispielsweise die Interaktion des Fahrzeugs mit seiner Umwelt, also zum Beispiel mit mehreren anderen Verkehrsteilnehmern, angesichts der hohen Anzahl an Äquivalenzklassen nicht auf eine übersichtliche und gut lesbare Art und Weise definieren.

- Neue Parametrisierung der Funktion: Dadurch dass die Entwicklungsumgebungen für die manuelle Programmierung von Funktionen meist eigene Konfigurationsformate haben, ist mit einem manuellen Aufwand für die Übertragung (oder gar Neuerzeugung falls nötig) der Funktionsparameter aus dem MiL-Format zu rechnen.

- Programmierung und Parametrisierung einer Komponente, die den automatisierten Testablauf steuert: Im Gegenteil zu MiL-Testverfahren, bei denen die Hersteller meistens eine integrierte Lösung anbieten, besteht bei den SiL-Verfahren, wie beispielsweise bei der virtuellen Simulation, eine Tool-Landschaft, die erst durch einen Skript oder eine dedizierte Komponente zu einem automatisierten Testablauf gebracht werden kann. Oftmals sind solche Teile der Testspezifikation wie Soll-Werte in solchen Skripten festkodiert oder bestenfalls durch proprietäre Dateiformate einstellbar. Es liegt also keine formatunabhängige Beschreibung von diesen auf einer höheren semantischen Ebene vor.

Zusammenfassend lässt sich feststellen, dass die MiL-Methodik insofern für diese Arbeit von Bedeutung ist, als sie, beziehungsweise die von ihr verwendeten Formate für die Definition von ausführbaren Testspezifikationen als Zielformate dienen sollen, in die eine auf einer höheren semantischen Ebene definierte Testspezifikation automatisch transformiert wird. Dadurch soll der Übergang von der modellbasierten Entwicklung in die Code-basierte erleichtert werden. Insbesondere betrifft dies die automatisierten Testabläufe.

2.4 SiL-Testverfahren

In diesem Abschnitt wird das Verfahren Software-in-the-Loop vorgestellt mit besonderem Schwerpunkt auf der virtuellen Simulation.

2.4.1 Beschreibung des Verfahrens

Der wesentliche Unterschied zwischen der MiL-Methodik und der SiL-Methodik besteht darin, dass bei der letzteren die Funktion als ein bereits prototypisch im Code implementiertes Software-Modul vorliegt. In einer prototypischen Implementierung ist der Detaillierungsgrad im Vergleich zum Modell wesentlich höher. Zudem werden bereits solche Design-Informationen berücksichtigt wie zum Beispiel Komma-Arithmetik: In einem Modell wird oftmals mit Gleitkommata gearbeitet, in der Implementierung dagegen mit Fixkommata [Lam03]. Die Implementierung der letzteren erfolgt Zielprozessor-abhängig. Ein anderer Unterschied besteht darin, dass bei der virtuellen Simulation die Umgebung beziehungsweise die Interaktion des SUT mit der Umgebung

eine wesentlich höhere Rolle spielt als noch beim MiL. So wird beim SiL mittels virtueller Simulation oft der komplette Verkehrsfluss simuliert samt graphischer Visualisierung. Da die virtuelle Simulation für reaktive Systeme ein relativ junges Gebiet ist, gibt es noch nicht so viele integrierte Lösungen, die beispielsweise ähnlich dem Matlab/Simulink die Definition, Ausführung und Laufzeitüberwachung der Testspezifikationen „aus einer Hand“ erlauben. Vielmehr existiert auf diesem Gebiet, wie bereits erwähnt, eine heterogene Tool-Landschaft mit unterschiedlichsten Beschreibungsformaten für die Daten.

Im Folgenden wird nun die Vorgehensweise vorgestellt, die zur Erstellung einer ausführbaren Testspezifikation beim SiL mittels virtueller Simulation notwendig ist. In der Abbildung 2.12 ist diese zunächst schematisch dargestellt.

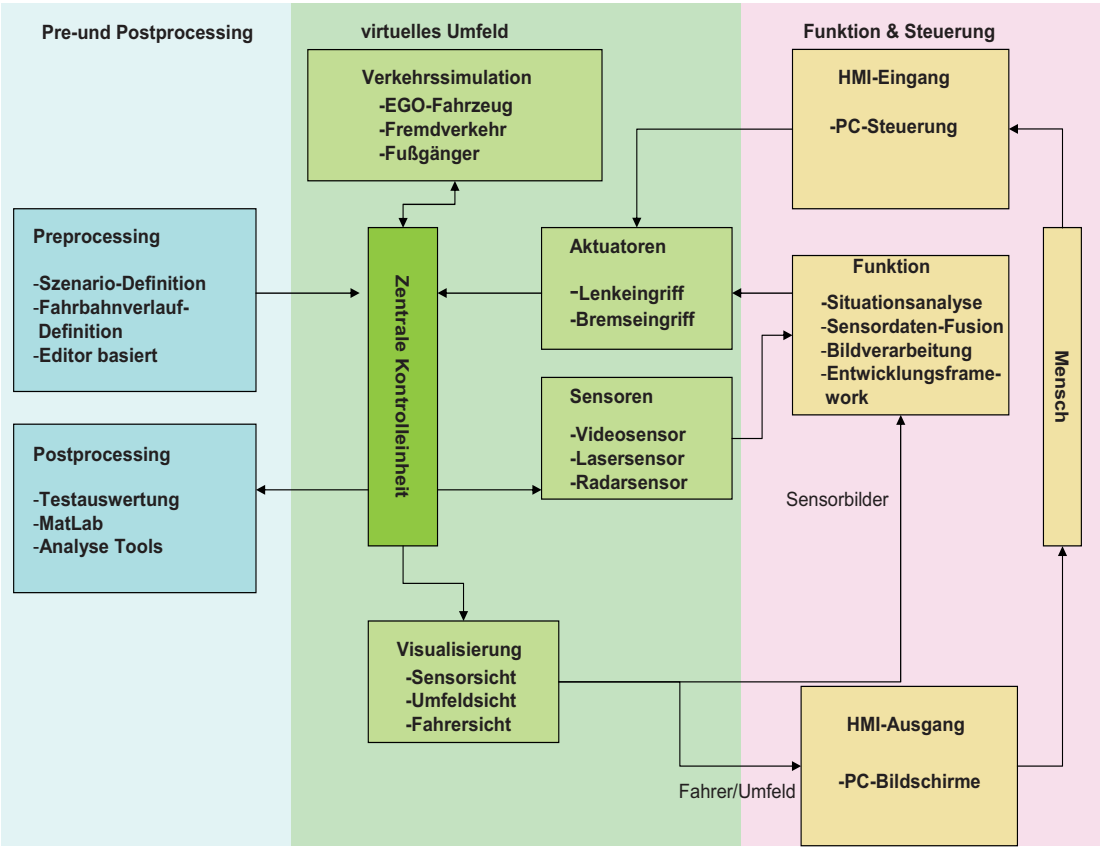


Bild 2.12: SiL-Architektur

Preprocessing

In diesem Schritt geht es darum, die Fahrszenarien und die zugehörigen Fahrbahnverläufe zu definieren. Die Fahrszenarien werden in diesem Fall dazu verwendet, die eigentlichen Verkehrsdaten in einem nachgelagerten Schritt zu erzeugen. Die Fahrszenarien beinhalten eine Abfolge von Fahrmanövern, die sowohl vom EGO als auch von Fremdfahrzeugen ausgeführt werden. Die Fahrszenarien können beispielsweise in einer XML-Datei verwaltet werden. Das Ergebnis der Straßenverlaufsdefinition ist eine XML-Datei in OpenDrive-Format [Vir10a]. Falls die Postprocessing-Phase ohne zeitliche Unterbrechung nach der Simulation erfolgt, müssen bereits im Vorfeld die Soll-Werte im Format des entsprechenden Auswertungstool definiert werden. Nachdem nun der Preprocessingschritt abgeschlossen ist, beginnt der eigentliche Simulationsvorgang. Dieser wird vom Entwickler gestartet und besteht prinzipiell aus drei wichtigen Bereichen: Virtuelles Umfeld, Funktion und Vorgangssteuerung.

Virtuelles Umfeld

Die wichtigste Komponente im virtuellen Umfeld ist die Verkehrssimulation, welche die Daten über das EGO-Verhalten, das Fremdverkehrverhalten und über das Fußgängerverhalten liefert. Die produzierten Daten umfassen sowohl die Positionen der Verkehrsteilnehmer als auch die Videodaten, die zur Visualisierung des Verkehrsflusses notwendig sind. Über eine Kontrolleinheit bekommen die Komponenten „Visualisierung“ und „Sensoren“ (Abb. 2.12) die oben beschriebenen Informationen von der Verkehrssimulation. Die Visualisierung kann aus verschiedenen Perspektiven geschehen wie zum Beispiel aus der Fahrersicht oder aus der Umfeldsicht. Die Sensoren werden wiederum durch Sensormodelle implementiert und sie können beispielsweise einen Radar- oder einen Lasersensor repräsentieren. Aktoren sind beim SiL mittels virtueller Simulation ebenfalls durch Modelle ersetzt und sorgen dafür, dass die von der Funktion produzierten Signale in beispielsweise Lenk- oder Bremseingriffe umgesetzt werden. Ebenfalls können Aktoren zur Warnung des Fahrers eingesetzt werden, indem akustische oder optische Signale erzeugt werden. Die Verkehrssimulation und die Visualisierung sind meistens in einer Umgebung integriert. Was die Aktoren und die Sensoren betrifft, so können diese in einer separaten Umgebung implementiert sein. Diese Umgebung kann zum Beispiel eine Entwicklungsumgebung sein, in der die Funktion manuell programmiert wird.

Funktion

An der Simulation ist normalerweise eine Entwicklungsumgebung beteiligt, in welcher die zu testende Funktion läuft. Die Funktion empfängt die Daten von den Sensoren sowie von der Visualisierung über gängige Netzprotokolle wie Ethernet [Rec08]. Bevor allerdings die Daten von der Funktion bearbeitet werden, sind noch einige verarbeitende Schritte notwendig wie zum Beispiel die Sensordatenfusion und die Situationsanalyse. Die genaue Reihenfolge der Datenverarbeitungsschritte ist in der Abbildung 2.13 zu sehen. Bei der Sensordatenaufbereitung

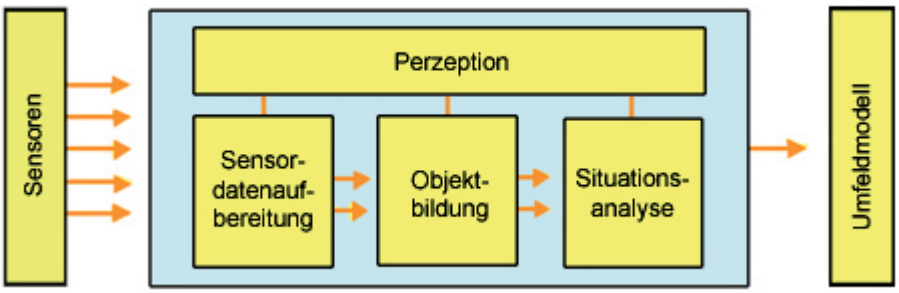


Bild 2.13: Multisensorielle Umgebungserfassung [Bun10]

müssen drei wichtige Aufgaben erledigt werden. Zum einen müssen die Sensoren bezüglich des festgelegten Koordinatensystems ortskalibriert werden. Da sich die Objekte im Verkehrsumfeld oft mit hoher Geschwindigkeit bewegen, ist die Festlegung einer gemeinsamen Zeitbasis eine fast unverzichtbare Voraussetzung für einen multisensoriellen Ansatz. Stereokameras werden z.B. synchron betrieben, um beide Messungen zum gleichen Zeitpunkt zu erhalten. Alternativ können auch asynchrone Systeme verwendet werden, wenn die Messungen mit einem Zeitstempel versehen werden (Ereignissteuerung), der von einer gemeinsamen Systemuhr (Master-Clock) geliefert wird. Schließlich werden in den Sensormodellen alle bekannten Eigenschaften von den Sensoren abgelegt und bei der Sensordatenfusion explizit verwendet. Nur dann ist es möglich, Eigenschaften der einzelnen Sensoren (wie Reichweite, Öffnungswinkel) auch bei Änderungen (z.B. anderes Kameraobjektiv) effizient zu berücksichtigen [Bun10]. Im nächsten Schritt findet die so genannte Objektbildung statt, bei welcher beispielsweise mehrere Reflexionspunkte eines Radars zu einem Objekt zusammengefasst werden. Im Bereich der Bildverarbeitung existiert eine Reihe von Algorithmen, die darüber entscheiden, ob eine Menge von benachbarten Pixeln ein Objekt bildet oder nicht. Die letzte wichtige Aufgabe in der multisensoriellen Umgebungserfassung ist die Situationsanalyse, bei welcher es darum geht, die detektierten Objekte in Beziehung zu

einander zu setzen. Konkret geht es um die Erkennung bestimmter Situationen wie zum Beispiel „Gassenfahrt“ oder „Naheinscherer“. Für FAS ist die Situationsanalyse insofern von Bedeutung, als diese dazu eingesetzt wird, eine Risikobewertung vorzunehmen, die aus der aktuellen Situation erschlossen wird, und ausgehend davon eigene Aktion zu planen beziehungsweise den Fahrer über den entsprechenden Aktor zu warnen, dass das System abgeschaltet wird, da dessen Einsatz bei der ermittelten Situation unmöglich oder risikobehaftet ist.

Steuerung

Im Bereich „virtuelles Umfeld“ ist eine zentrale Kontrolleinheit (Abb. 2.12) für die Steuerung und zeitliche Koordination der Simulation zuständig. Konkret bedeutet dies, dass alle Datenströme und Kommandos über diese laufen. Darüberhinaus sind zwischen der Steuerungskomponente und den restlichen Komponenten im Bereich „virtuelles Umfeld“ Schnittstellen definiert. Die Kontrolleinheit ist imstande mit unterschiedlichen Kommunikationsprotokollen umzugehen. Es bestehen ebenfalls zwei weitere Möglichkeiten den Ablauf der Simulation von außen zu steuern. Zum einen können von der Funktion zur Laufzeit so genannte „Trigger“ abgesetzt werden, die den Ablauf der Simulation verändern. So kann auf diese Art und Weise die Geschwindigkeit bestimmter Verkehrsteilnehmer verändert werden. Gleichzeitig kann der Entwickler über eine HMI-Komponente Einfluss auf die Simulation zur Laufzeit nehmen.

In Bezug auf die automatisierten Testabläufe ist ebenfalls eine externe Ausführungskomponente notwendig, die das zyklische Abspielen und Auswerten mehrerer Fahrszenarien erlaubt. Konkrete Aufgaben dieser Komponente bestehen darin, den Ablauf der Entwicklungsumgebung und des virtuellen Umfelds zu überwachen, das Abspielen der einzelnen Fahrszenarien zu beenden, die Postprocessingphase einzuleiten und den Durchlauf mit der nächsten Fahrszene zu starten.

Postprocessing

Im Bereich „Postprocessing“ findet die Auswertung der während der Simulation aufgezeichneten Funktionssignale statt. Dazu kann sowohl der klassische Soll-Ist-Vergleich im Sinne eines Black-Box-Tests verwendet werden als auch eine einfache Visualisierung der Signalverläufe, die vom Entwickler visuell ausgewertet wird. Die Auswertung kann sowohl in einer separaten Umgebung geschehen wie zum Beispiel Matlab, aber auch in die Entwicklungsumgebung integriert sein.

2.4.2 Einsatz in der Anwendungsdomäne

Aktuell wird SiL im Bereich der Vorentwicklung mittels virtueller Simulation aktiv zur Aufnahme von Verkehrsdaten verwendet. Simuliert werden dabei bestimmte Fahrmanöver, welche für eine reale Messfahrt mit Risiken verbunden wären, wie beispielsweise Naheinscherer. Dazu wurde folgende Architektur realisiert. Im Bereich „virtuelles Umfeld“ wird die von der Firma Vires enwickelte Umgebung „Virtual Test Drive“ [Vir10b] eingesetzt. Diese sendet zur Laufzeit über Ethernet fest definierte Daten an den Windows-PC, auf welchem das Automotive Data and Time Triggered Framework [Löb08] (ADTF) läuft. Einzelne Funktionen werden in ADTF in Form von Plug-ins in C-Code entwickelt. In der hier vorgestellten Konstellation läuft in ADTF lediglich ein Plug-In, das die vom „Virtual Test Drive“ empfangenen Verkehrsdaten in Binärformat aufzeichnet. Der Aufzeichnungsvorgang wird für diverse Parametervariationen eines bestimmten Grundszenarios zyklisch wiederholt. Dazu ist eine Ablauf-Automatisierungskomponente (Testautomat) notwendig, welche den Ablauf von ADTF steuert und überwacht. Der Entwickler muss in der Preprocessing-Phase folgendes definieren:

- Fahrszenenbeschreibung als XML-Export aus der dedizierten graphischen Oberfläche.
- Konfiguration von ADTF in Form einer XML-Datei.
- Konfiguration des Testautomaten in Form einer XML-Datei.
- Parameterintervalle für die zu variierenden Parameter als Einträge in der XML-Datei für die Konfiguration des Testautomaten.

Die Postprocessing-Phase als solche wird in dieser Variante erst später ausgeführt, da die erzeugten Verkehrssimulationsdaten in eine Szenariendatenbank [Ent06] eingestellt werden, aus welcher sie später automatisiert zum so genannten „open-loop“ Testvorgang ausgelesen werden. Ersichtlich wird aus diesem Anwendungsbeispiel, dass zum Zustandebringen nicht einmal des kompletten Testablaufs, sondern lediglich der automatisierten Testdatenaufnahme die Beschreibung der Daten in drei diversen plattformspezifischen Formaten notwendig ist. Möchte man einen kompletten Testablauf in der Zukunft realisieren, dann kommt eventuell noch ein zusätzliches Format für die Definition von Soll-Werten hinzu. Aus technischer Sicht fehlt aktuell zu einem Testlauf noch eine Rückschleife von der Entwicklungsumgebung (ADTF) zurück zum „Virtual Test Drive“.

2.4.3 Bewertung der Methode

SiL mittels virtueller Simulation für reaktive Systeme wie FAS ist, wie bereits dargestellt, ein relatives junges Gebiet, bei dem noch nicht viele Standardisierungsversuche unternommen worden

sind. So existiert zwar ein Bestreben, die Fahrbahnbeschreibung mittels OpenDrive [Vir10a] zu standardisieren, für einige andere Bereiche sind aber noch keine einheitlichen Beschreibungsformate vorhanden. Dies gilt zum Beispiel für die Fahrszenen-Beschreibungen, welche je nach dem Hersteller zwar mit einer graphischen Oberfläche erstellt werden, dafür aber in proprietären Dateiformaten abgespeichert sind. Hinzu kommt die Tatsache, dass, wie bereits erwähnt, an SiL-Abläufen mittels virtueller Simulation mehrere Anwendungen teilnehmen, die auf unterschiedlichste Art und Weise parametrisiert werden müssen.

In der Praxis ist es oft so, dass bestimmte auf dem Prüfgelände eingefahrene Fahrszenen, zu einem späteren Zeitpunkt mit geringen Veränderungen mittels SiL simuliert werden sollen. Das ist vor allem dann der Fall, wenn die vorgenommenen Veränderungen für die Ausführung des Szenarios in der realen Welt ein gewisses Gefährdungspotential in sich bergen. Verringert man zum Beispiel die Distanz des EGO-Fahrzeugs zum Vorderfahrzeug bei einem Auffahrmanöver, so kann dies auf dem Prüfgelände im schlimmsten Fall zu einer Kollision führen. Bislang musste jedes Szenario, das in der realen Welt eingefahren wurde, in einem Katalog nachgesehen werden. Dann musste es komplett neu mit Hilfe eines graphischen Editors erstellt werden. Bei einem Szenario hält sich der zeitliche Aufwand noch in Grenzen, bei mehreren eingefahrenen Szenarien wird er zum gewichtigen Zeitfaktor. Eine einheitliche Beschreibung von Testdaten würde also eine große Abhilfe schaffen.

In der umgekehrten Richtung möchte man natürlich wissen, inwieweit die mit SiL durchgeführten Tests denen mit realem Fahrzeug entsprechen oder, verallgemeinert, mit realen Messdaten. Es stellt sich also die Frage „Sind die mit SiL erreichten Testergebnisse genauso gut wie die mit den realen Tests?“. Denn man muss sich selbstverständlich im klaren sein, dass die Simulationen zwar enorm viel an Zeit und finanziellen Mitteln sparen, sie nutzen aber wenig, wenn sie wichtige Aspekte der Realität nicht ausreichend präzise abbilden.

In Bezug auf diese Arbeit würde eine einheitliche Testspezifikationsbeschreibung den manuellen Aufwand zur Erstellung bestimmter Teile dieser, also beispielsweise die Fahrszenen, verringern. Ebenfalls wäre somit „die Umwandlung“ der Testspezifikation aus der virtuellen Welt in die reale und umgekehrt wesentlich erleichtert und deshalb auch eine solide Basis zur Vergleichbarkeit von Testergebnissen gelegt.

2.5 HiL-Testverfahren

In diesem Abschnitt wird die Hardware-in-the-Loop Testmethodik [Gom01] beschrieben. Im Vergleich zu MiL und SiL ist diese die wohl am weitesten entwickelte und die am methodischsten eingesetzte Testart.

2.5.1 Beschreibung des Verfahrens

Wie bereits beschrieben, wird HiL beim System- und Integrationstest sowohl vom Zulieferer als auch vom OEM eingesetzt. Da es sich um Serienentwicklung handelt, liegt bei dieser Testart die Funktion bereits als ein für den Einsatz auf einer bestimmten Zielhardware optimiertes Software-Modul vor. Der beispielhafte Prüfstandaufbau ist in der Abbildung 2.14 dargestellt. Der Prüfling, also die Steuereinheit(en) mit der entsprechenden Funktion, ist zusammen mit

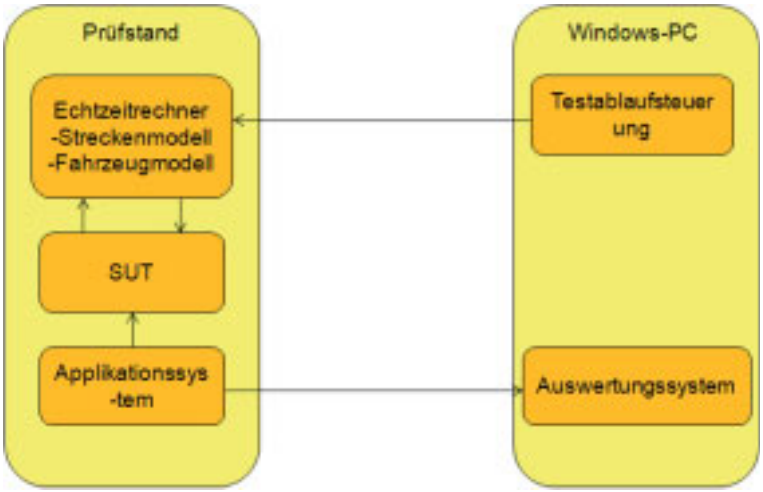


Bild 2.14: Beispielhafter Aufbau eines HiL-Prüfstands

dem Applikationssystem und einem Echtzeitrechner in einem Prüfstand (auch als HiL-Schrank bezeichnet) integriert. Auf dem Echtzeitrechner laufen Modelle, die beispielsweise die Fahrzeugdynamik oder die Strecke simulieren. Der Datenaustausch zwischen dem SUT und dem Echtzeitrechner ist rückgekoppelt, d.h. die von der Funktion produzierten Signale werden von den auf dem Echtzeitrechner laufenden Modellen verarbeitet. Der gesamte Vorgang wird von einem separaten Rechner, meistens einem Windows-PC, gesteuert. Alle von dem Prüfling produzierten Signale werden von dessen Applikationssystem an ein Auswertungssystem weitergeleitet, das sich ebenfalls auf dem Windows-PC befindet. Ein wichtiger Unterschied zu MiL und SiL besteht darin, dass an der Vorbereitung des Testvorgangs mehrere Personen beteiligt sind. Dazu ist die Einführung eines Prozesses notwendig, der Planung und vor allem Koordination aller Aktivitäten ermöglicht, die zur Ausführung eines automatisierten Testvorgangs notwendig sind. Im Folgenden wird nun ein solcher Prozess (Abb. 2.15) beispielhaft vorgestellt. Hierbei handelt es sich um

„Extended Automation Method“ (EXAM [Mic10a]), die von Volkswagen AG im Bereich HiL-Test standardmäßig eingesetzt wird. Es wird zwischen den folgenden Rollen unterschieden:

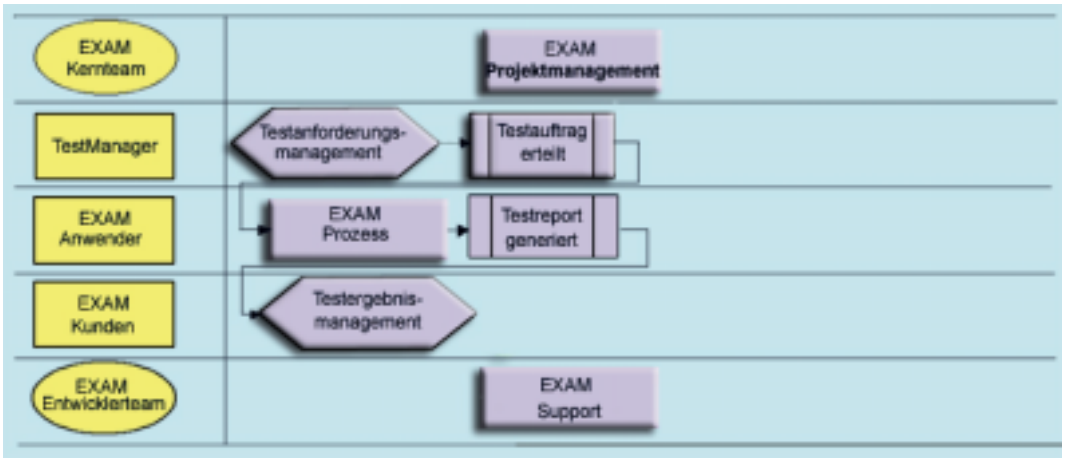


Bild 2.15: Prozess zur Vorbereitung und Koordination von HiL-Testabläufen [Aud07]

- EXAM-Kernteam: Leitung des EXAM-Projektes.
- Testmanager: Ist verantwortlich für ein bestimmtes Testprojekt und dessen Durchführung. Er koordiniert, welche Tests wann und wo durchgeführt werden, und beobachtet die zeitliche Entwicklung der Testergebnisse.
- EXAM-Anwender: Erstellt und pflegt die Testfälle.
- EXAM-Kunden: Fachabteilungen aus der Serien-Entwicklung.
- EXAM-Entwicklerteam: Besteht aus den EXAM-Projektleitern, den EXAM-Tool- und Bibliotheksentwicklern, sowie den Support-Mitarbeitern. Es beschäftigt sich mit der Entwicklung und Pflege von systeminternen Bibliotheken. Es ist ebenfalls mit der Entwicklung der nächsten Version von EXAM beauftragt.

Der Spezifikationsprozess (Bestandteil des EXAM-Prozesses) ist iterativ und verläuft zwischen der mit dem Test beauftragten Abteilung (Testmanager, EXAM-Anwender) und der Fachabteilung (EXAM-Kunde), die sich letztendlich mit der Entwicklung der zu testenden Funktion beschäftigt. Als Test-Spezifikationsgrundlage dient eine von IBM entwickelte Plattform zur Erfassung und Verwaltung von Anforderungen namens DOORS [Int10]. Die Testspezifikationen werden in strukturiertem Text erfasst. Dabei wird folgende Struktur verwendet: alle Aktivitäten, die zum

Herstellen eines bestimmten SUT-Zustands dienen, werden im Abschnitt „Condition“ hergestellt. Sobald das zu testende System in dem gewünschten Zustand ist, wird im Abschnitt „Action“ der eigentliche Testfall beschrieben. Dabei ist wichtig zu erwähnen, dass alle Parameter beziehungsweise auch deren Belegungen in dem Testfall angegeben werden müssen. Ein Testfall, der die gleiche Struktur, aber eine andere Parameterbelegung hat, wird als ein neuer Testfall betrachtet und im DOORS-Dokument abgelegt. Schließlich wird im Abschnitt „Ergebnis“ das gewünschte Soll-Ergebnis spezifiziert samt den Funktionssignaturen, mit denen die Auswertung geschehen soll. Sobald ein bestimmter Stand bei der Testfallspezifikation erreicht worden ist, besteht die Möglichkeit den Testfall in die Modellierungsumgebung EXAM zu exportieren (Hier verwendet jede der untersuchten Abteilungen ein proprietäres Tool). Dabei wird nur ein zunächst leeres Element „Testcase“ angelegt mit der entsprechenden textuellen Beschreibung, welche aus DOORS exportiert wird. Der Testentwickler (EXAM-Anwender) analysiert diese exportierte Testfallbeschreibung und erzeugt daraus innerhalb des Elementen „Testcase“ ein UML-Sequenzdiagramm (Abb. 2.16 [Obj09]), welches den strukturellen Testablauf darstellt. Ebenfalls besteht die Möglichkeit, den beschriebenen Testfall in ein Wiki im Engineering Portal von Audi einzutragen. Das Wiki bietet eine schnelle Möglichkeit zum Nachschlagen von Testfallbeschreibungen für alle Testprozessbeteiligten. Ein großer Vorteil der EXAM-Methodik ist, dass die entwickelten Testfälle unabhängig von einem bestimmten Prüfstand sind. Sie werden erst über einen komplexen Transformationsschritt in die Formate umgewandelt, welche von dem jeweiligen Prüfstand akzeptiert werden. Dazu müssen so genannte Abbildungsklassen definiert werden, die für jeden im Sequenzdiagramm definierten Parameternamen dessen Prüfstand-spezifischen Namen spezifizieren.

2.5.2 Einsatz in der Anwendungsdomäne

In der vorliegenden Arbeit wurden drei wichtige Themenbereiche untersucht, bei denen HiL-Tests zum Einsatz kommen: Antriebs-, Komfort-, Infotainmentbereich. Dabei wurden innerhalb eines jeden Themenbereichs folgende Gebiete untersucht:

- Antrieb: Diverse Funktionen im Verbund (ACC, Lane Departure Warning, Spurwechsel-Assistent, ESP), Getriebe, Motor, Bordnetz. Einzelne Funktionen (zum Beispiel ACC).
- Komfort: Schlafenlegen/Aufwachen von Steuergeräten, Batterie-Energiemanagement, Ruhestrom-Messung, Komponentenschutz, Wischen, Diagnose von Steuergeräten.
- Infotainment: Multimedia Interface (MMI), Soundsystem, Bluetooth-Autotelefon, Tuner (DAB, FM, DVBT)

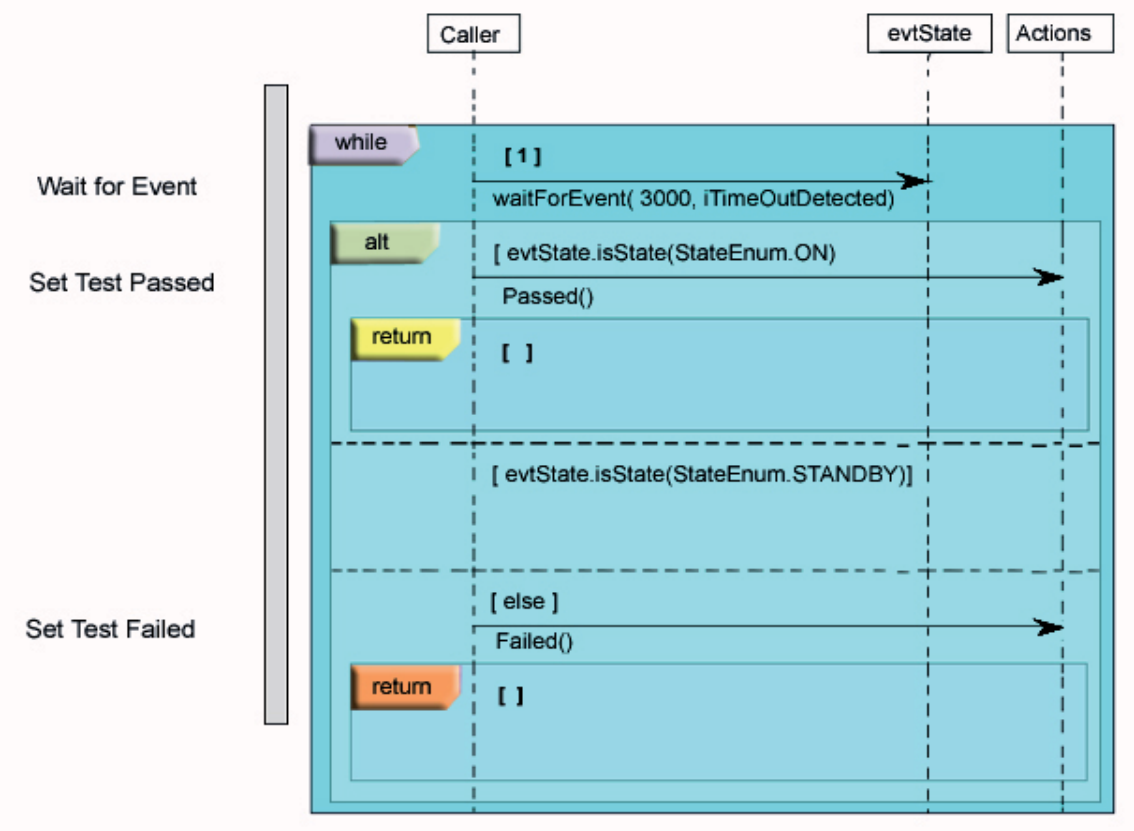


Bild 2.16: Beispielhafte Testfall-Spezifikation mit EXAM [Aud07]

Da als Anwendungsgebiet für diese Arbeit der Bereich der Fahrer-Assistenzsysteme gilt, wird in diesem Abschnitt ein Beispiel aus dem Antriebsbereich gebracht. Dieser demonstriert die Anwendung des Verfahrens auf eine einzelne Funktion (ACC).

ACC-Einzel-HiL

Die Einzel-HiL-Testläufe werden oft von den Zulieferern ausgeführt. Die Testspezifikation wird in Zusammenarbeit von Testfall-Programmierer und dem Entwicklungsingenieur in Form des strukturieren Textes in DOORS abgelegt. Deren Struktur entspricht dabei der im Abschnitt 2.5.1 beschriebenen. Jeder erzeugten Testfallspezifikation werden ein Ausführungsort sowie eine SynchronisationsID zugewiesen. Diese SychnronisationsID wird vom proprietären Synchronisationswerkzeug verwendet, um die Testfallspezifikationen nach EXAM zu exportieren. Jede Testspezifikation enthält außerdem einen Verweis auf die entsprechenden Anforderungen aus einem separaten ACC-Lastenheft, das von der OEM kommt und ebenfalls in DOORS verwaltet

wird. Sehr wichtig zu erwähnen ist, dass das Synchronisationswerkzeug keineswegs komplette Sequenzdiagramme in EXAM erzeugt, sondern lediglich eine Ordnerstruktur sowie leere TestCase-Elemente anlegt, in denen die Informationen aus der Testfallspezifikation in der Spalte „Beschreibung“ abgelegt werden. Diese dienen als Ausgangsbasis für den Testfallprogrammierer, der im nächsten Schritt, basierend auf diesen Informationen *manuell* ein UML-Sequenzdiagramm erzeugt.

Der grundsätzliche Prüfstand-Aufbau für ACC-Einzel-HiL ist in der Abbildung 2.17 dargestellt. Über einen Bedien-PC kann der HiL-Schrank manuell mit ControlDesk von dSpace

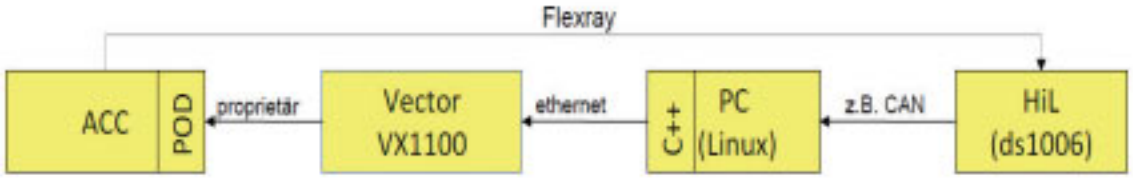


Bild 2.17: Prüfstand-Aufbau für ACC-Einzel-HiL

[dSp10b] oder automatisiert mit EXAM gesteuert werden. Der Testingenieur kann so (bisher nur) ein Zielobjekt über die Zielobjektsimulation (Linux-PC), in Form einer Objektliste über die Vector Box VX1100 und einem Plug-on Device (POD) [Alb10] vorgeben. Auf dem Echtzeitrechner (HIL, ds1006) läuft das Fahrdynamikmodell Automotive Simulation Models (ASM) [dSp10a] von dSpace. Beim ACC-HiL Test geht es verstärkt darum, das Steuergerät in seiner Umgebung zu testen. Da man im Labor keine Sensoreinheiten einsetzen kann, wird ein so genanntes „Software-Bypassing“ angewandt (Abb. 2.18). Dabei werden die vom Sensor detektierten Objektlisten entweder statisch definiert oder sie werden durch die Simulation dynamisch erzeugt und in die entsprechenden Sensormodelle eingespeist. Gerade das virtuelle Umfeld bietet auf diesem Gebiet neue Möglichkeiten, da die Objektlisten von detektierten Objekten nicht mehr statisch vor dem Testlauf festgelegt, sondern dynamisch zur Laufzeit erzeugt werden. Dazu ist, genauso wie beim SiL, eine präzise Beschreibung von der Interaktion des SUT mit seiner Umwelt notwendig, es muss also im FAS-Fall eine Fahrszenario-Beschreibung vor dem eigentlichen Testlauf erstellt werden.

2.5.3 Bewertung des Verfahrens

Wie eingangs erwähnt, ist das HiL-Verfahren weitestgehend im VW-Konzern etabliert: Es existiert im Vergleich zu MiL und SiL ein ausgereifter Prozess zur Planung, Vorbereitung und Ausführung

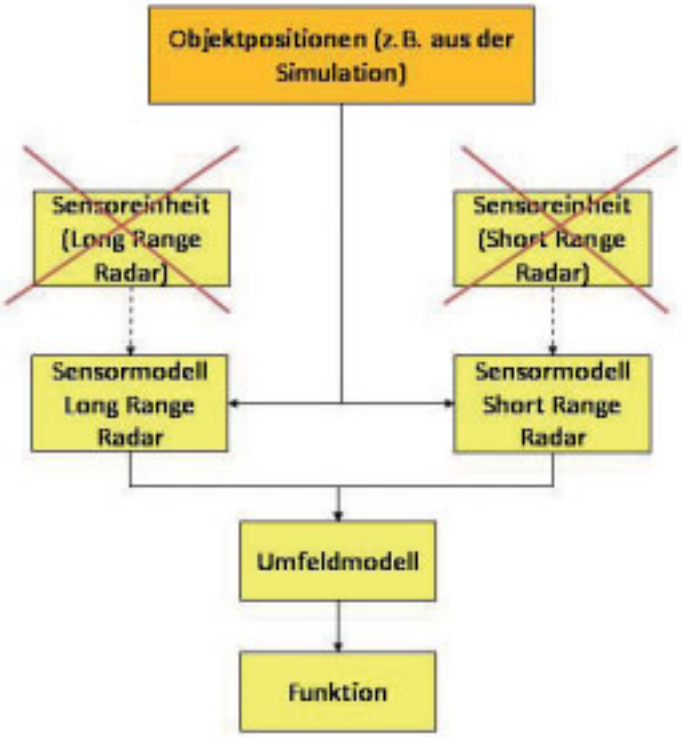


Bild 2.18: Software-Bypassing Prinzip

von Testaktivitäten. Die Software zur Erstellung der Testfälle und deren Ausführung am Prüfstand wird ständig weiterentwickelt und neue Technologien wie Verwendung von virtuellem Umfeld zu einer realitätsnahen Umfeldgenerierung werden eingeführt. In Bezug auf diese Arbeit lässt sich jedoch folgendes feststellen:

- Nicht formale Testspezifikationsbeschreibung: Dadurch, dass Testspezifikationen zunächst lediglich als semi-strukturierter Text vorliegen, geschieht deren Transformation in formale testartspezifische Notation manuell durch den Testfall-Programmierer.

- keine Trennung zwischen abstrakter und konkreter Testspezifikationsbeschreibung: In manchen untersuchten Bereichen werden Testspezifikationen in Tabellendokumenten für eine bestimmte Testumgebung (wie z.B. EXAM) spezifiziert. Solche Dokumente werden oft zusammen mit den restlichen für einen automatisierten Testablauf notwendigen Daten für ein bestimmtes Testprojekt verwaltet. Deshalb erweist sich deren Wiederverwendung in anderen verwandten Projekten als problematisch. Selbst wenn die Testfallbeschreibung für EXAM

als Prüfstand-unabhängig gilt, so wurde auf der Testspezifikationsebene in DOORS festgestellt, dass viele Testfälle immer wieder dieselbe Stuktur aufweisen, was beispielsweise die Ausführungsreihenfolge bestimmter Testaktivitäten betrifft. Allerdings wurde festgstellt, dass dadurch, dass die Parameterwerte gleich in die Testspezifikation hineingeschrieben werden, viele Testfallbeschreibungen sich wiederholen, da für den Testfall-Programmierer bei dem bestehenden Grad an Nicht-Formalisierung kaum die Möglichkeit besteht, nach dem Vorkommen der Testfälle mit ähnlicher Struktur zu suchen. Dies führt zu Redundanzen bei der Beschreibung, da möglicherweise bei den neu definierten Testfällen der Großteil der Parameter manuell mit Werten belegt werden muss, die schon bei einem ähnlichen Testfall eingegeben wurden, und mit nur wenigen neuen Parameterbelegungen.

- Keine klare Trennung zwischen den einzelnen Bestandteilen der Testspezifikation: Es wurde beobachtet, dass beispielsweise Testfallbeschreibungen und die Angabe von den Soll-Werten miteinander vermischt werden, indem die letzteren direkt in die Testfallbeschreibungen integriert werden. Bei den Soll-Wert Angaben werden außerdem direkt plattformspezifische, programmiersprachliche Auswertungsfunktionen, beziehungsweise deren Signaturen, hineingeschrieben.

2.6 Erprobungsfahrten

2.6.1 Verfahrensbeschreibung

In den frühen Entwicklungsphasen liegt die Funktion, wie bereits erwähnt, als ein von der Serien-Steuereinheit unabhängiges Software-Modul vor. Um im Fahrzeug lauffähig zu sein, wird sie samt der Entwicklungsumgebung auf einem Fahrzeug-Rechner installiert. Solche Fahrzeugrechner sind meistens konventionelle PC's, welche im Heckraum des Versuchsfahrzeugs untergebracht werden. Gewöhnlich werden mehrere Rechner installiert, auf welchen verschiedene Module einer Funktion ablaufen. So kann beispielsweise die Analyse der Umgebungswahrnehmung auf einem Rechner ablaufen und die Situationsanalyse auf einem anderen. Die PC's sind ebenfalls an die entsprechenden CAN-Busse angeschlossen. Die Testfahrt wird meistens von zwei Personen durchgeführt: Der Testfahrer und der Entwicklungsingenieur. Der Entwicklungsingenieur beobachtet das Verhalten des SUT auf einem eigens im Fahrzeug installiertem Bildschirm, während der Fahrer eine bestimmte Reihenfolge der Fahrmanöver abfährt. Die Fahrmanöver werden samt Referenzwerten meistens in den entsprechenden Katalogen verwaltet. Diese sind in den jeweiligen Entwicklungsabteilungen als elektronische Dokumente vorhanden. Bei bestimmten Funktionen

kann es ebenfalls sein, dass man keine vorher definierte Fahrmanöver fährt, sondern einfach eine Erprobung im Straßenverkehr durchführt. Beispielsweise kann es sich um eine Erprobung im Stauverkehr auf der Autobahn handeln oder aber um diese im städtischen Verkehr. In diesem Fall befinden sich die Referenzwerte im Bewusstsein des jeweiligen Entwicklers beziehungsweise des Testfahrers und sind daher sehr subjektiv. Eine andere Kategorie von Tests im Fahrzeug ist eine, bei der die Tests auf einem Prüfgelände ausgeführt werden. Dort werden unter anderem sicherheitskritische Testabläufe ausgeführt, bei denen die Ausführung im Straßenverkehr ein gewisses Gefährdungspotenzial für die Verkehrsteilnehmer implizieren würde. Dazu gehört beispielsweise ein dichtes Auffahren auf Fußgänger im Bereich Fussgängerschutz, die durch Testpuppen simuliert werden.

2.6.2 Bewertung der Methode

Die Auswertung der Testergebnisse nimmt der Entwickler zum Beispiel während der Fahrt vor, sie ist daher subjektiv. Ein wichtiger Unterschied zur Serienentwicklung besteht darin, dass dort der Softwarestand der Funktion direkt auf die beteiligten Steuereinheiten überspielt (umgangssprachlich: „geflasht“) wird statt auf einen PC wie in der Vorentwicklung. Erprobungsfahrten sind ein gewichtiger Kostenfaktor, sie sind aber unabdingbar.

2.7 „Open Loop“ Testverfahren

Als „Open Loop“ Testverfahren werden in dieser Arbeit solche bezeichnet, bei denen keine Rückkopplung zwischen der Funktion und der Umgebung besteht. Die „Open Loop“ Tests können automatisiert sowohl mit realen als auch mit virtuellen vorher aufgezeichneten Testdaten ablaufen. Im FAS-Bereich werden die Testdaten oft mit Fahrszenarien gleichgesetzt. Eine wichtige Eigenschaft von den Open-Loop Testverfahren ist, dass sie nicht als eine eigenständige Kategorie von Testverfahren angesehen werden kann. Vielmehr können „Open Loop“ Testverfahren sowohl bei SiL als auch, zumindest theoretisch, bei HiL sowie bei MiL durchgeführt werden. Im Weiteren werden beispielhaft einige SiL „Open Loop“ Testverfahren näher betrachtet.

2.7.1 Verfahrensbeschreibung

Die Daten werden meistens durch ein dediziertes Programm aufgezeichnet, das je nach Konfiguration entweder nur CAN-Verkehr aufzeichnet oder aber auch sämtliche im Fahrzeug stattfindende

Kommunikation inklusive die von Sensoren gemessenen Signale (z.B. Videobilder und Radarreflexe). Aufgrund der Tatsache, dass die Messdaten während der Aufzeichnung nicht komprimiert werden dürfen, entstehen bei Aufzeichnungen enorme Mengen an Daten. Bei einer Serien-Dauererprobung können beispielsweise bis zu 500 Terabyte an Messsignalen aufgenommen werden.

Noch vor einigen Jahren bestand das Problem der Verwaltung von Messdaten: Viele Entwickler legten diese einfach auf eigenen Entwicklungsrechnern ab und sie beschrieben die Messdaten lediglich durch den Dateinamen. Dies führte dazu, dass nach einer bestimmten Zeitperiode diese Testdaten selbst für den Entwicklungsingenieur nicht mehr verwendbar waren, da er anhand des Dateinamens nicht mehr feststellen konnte, um welche aufgenommene Fahrszene es sich handelt. Inzwischen wurden erste zentrale Datenverwaltungssysteme in Form von Datenbanken geschaffen [Ent06] [Aud10]. Diese erlauben es, die Messdaten zentral abzulegen und durch eine minimale Menge an Information in Form von Attributen zu beschreiben. Solche Datenbanken dienen oftmals als Ausgangsbasis für automatisierte „Open Loop“ Testabläufe im FAS-Bereich. Grundsätzlich unterscheidet man zwischen zwei Arten von „Open Loop“ Testverfahren.

Während der Testausführung werden die aufgezeichneten Fahrszenarien im ersten Schritt von einer Datenbank heruntergeladen (Abbildung 2.19). Anschließend werden sie sukzessive von der entsprechenden Ausführungsumgebung abgespielt. Diese filtert die Daten entsprechend der Konfiguration und leitet den Datenstrom an das SUT, das meistens als ein Plug-in der Entwicklungsumgebung realisiert ist. Im nächsten Schritt wird die eigentliche Berechnung durchgeführt. Deren Ergebnisse werden entweder in einer oder mehreren separaten Dateien abgespeichert oder es wird sofort eine Auswertung im Sinne eines Soll/Ist Vergleichs durchgeführt. Die Evaluierung kann entweder in der Ausführungsumgebung oder in einer separaten Anwendung ausgeführt werden. Was die Verwaltung von Soll-Werten (synonym Referenzwerte) bei dieser Testart betrifft, so können diese ebenfalls in einer Datenbank verwaltet werden. Dies ist dann der Fall, wenn die Referenzdaten per Hand nach der Messung erzeugt wurden (z.B. Markierung bestimmter Regionen im Videobild). Falls diese aber während der Messung automatisch aufgenommen wurden, dann werden sie nicht aus der Messdatei extrahiert, um in der Referenzdatenbank verwaltet zu werden, sondern sie werden dort belassen und erst zur Laufzeit extrahiert und als solche verwendet. Selbstverständlich sind Referenz-Datensätze bestimmten Fahrszenarien zugeordnet.

Die zweite Testart, welche im Rahmen der „Open Loop Testverfahren“ stattfindet, wird als „Klassifikator-Training“ bezeichnet (Abb. 2.20). Das Klassifikator-Training ist ein Verfahren, das aus dem Gebiet „Maschinelles Lernen“ kommt. Er wird überwiegend zur Bestimmung „optimaler“ Parametrierungssätze für Bild-basierte Klassifikatoren eingesetzt. Diese erkennen bestimmte

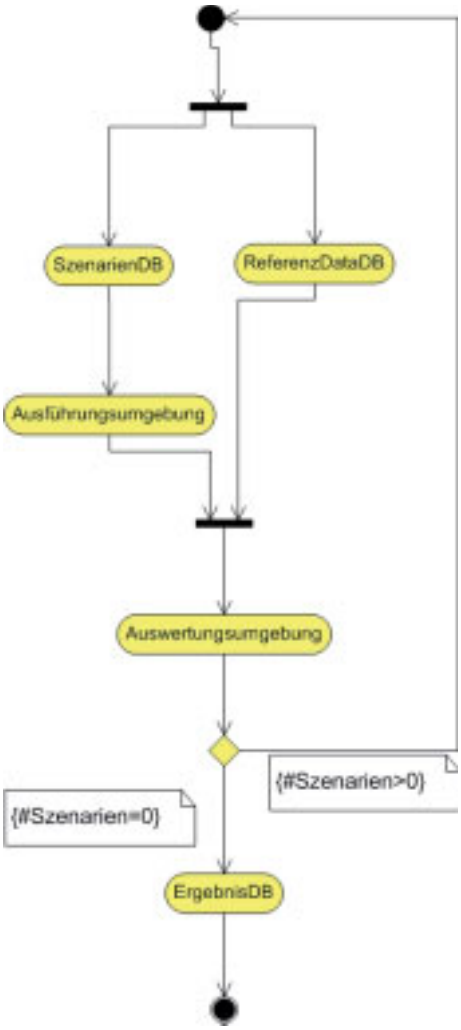


Bild 2.19: SiL open loop [EMWL08]

Objekte im Bild wie zum Beispiel Fussgänger oder Verkehrsschilder. Klassifikatoren sind oft als neuronale Netze oder als Vektor-Support-Maschinen [SC08] implementiert. Der Vorgang gliedert sich in die Trainingsphase und die Testphase (Abb.2.20). Zunächst werden aus dem heruntergeladenen Szenario bestimmte Eigenschaften extrahiert. Basierend auf diesen Eigenschaften wird der Trainingsvorgang mit einem bestimmten Parametersatz gestartet. Anschließend geschieht die Bewertung der Klassifikator-Ergebnisse anhand eines Vergleichs der von diesem gelieferten Ergebnisse mit bestimmten Soll-Werten. Danach wird anhand der Auswertungsergebnisse entschieden, ob der Parametersatz „gut genug“ war und der Trainingsvorgang beendet wird oder ob dieser fortgesetzt werden soll, indem der gesamte Ablauf für dieselbe Eigenschaftsmenge, aber mit einem neu kalkulierten Parametersatz fortgesetzt wird. Nachdem der Trainingsvorgang abgeschlossen ist,

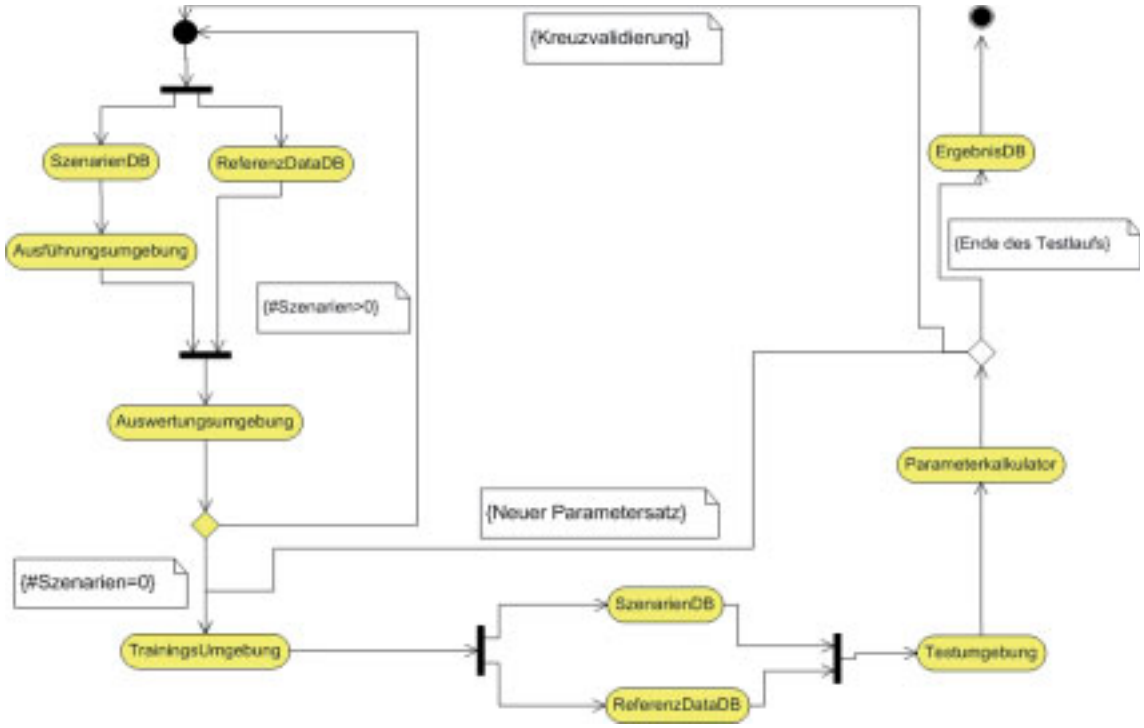


Bild 2.20: Klassifikator-Training

muss der Klassifikator mit dem ermittelten Parametersatz auf einer Fahrszenarien-Testmenge getestet werden. Dazu werden aus einer mit der Trainingsmenge disjunkten Fahrszenarien-Menge die Eigenschaften extrahiert. Anschließend bekommt der Klassifikator die Eigenschaften als Eingang, er verarbeitet sie und produziert einen Ausgang, der dann mit den zugehörigen Referenzwerten verglichen wird.

2.7.2 Einsatz in der Domäne

Als Beispiel wird im Folgenden ein Bild-basierter Symmetrieoperator aus der Vorentwicklung von FAS betrachtet. Der Symmetrieoperator befasst sich mit der Ermittlung von Symmetriepunkten diverser Objekte (z.B. Verkehrsschilder, Brücken, PKW) in den Bildern. Zur Berechnung von Letzteren benötigt der Algorithmus als Eingabedaten die Typinformationen des Objektes im Bild, in welchem die Symmetrie ermittelt werden soll. Außerdem wird die Distanz bis zu diesem Objekt benötigt. Da bei der Objektdetektion in der Bildverarbeitung mehrere unterschiedliche Operatoren verwendet werden, ist es notwendig für jeden Operator den Typ anzugeben. Vor dem eigentlichen Testvorgang markiert der Entwickler die Symmetriepunkte (Pixelpositionen) in den ausgewählten

Bildern. Diese Werte werden, sobald sie im Bild mit der Maus angeklickt wurden, in eine ASCII-Datei [Net10] herausgeschrieben und dienen als Referenzdaten. Anschließend wird der eigentliche Testvorgang mit den während einer Messfahrt aufgenommenen Messdaten gestartet. An dessen Ende liefert der Algorithmus pro Frame eine Menge ermittelter Symmetriewerte (Pixelpositionen), die in eine ASCII-Datei herausgeschrieben werden. Diese werden anschließend manuell vom Entwickler mit den Referenzdaten verglichen. Auch wenn die Soll-Werte bei diesem Algorithmus klar definiert sind, so geschieht die Testergebnisauswertung immer noch auf die manuelle Art und Weise, indem der Entwickler einen Ist/Soll Vergleich zwischen den vom Algorithmus gelieferten Symmetriewerten und den Referenzdaten ausführt, was den Testumfang stark einschränkt. Zu bedenken ist auch, dass mit der existierenden Testmethode keine Funktionsparameter-Optimierung möglich ist. Außerdem ist der Symmetrieoperator nur einer von vielen anderen Operatoren (es gibt beispielsweise diverse Kantenoperatoren), die zur Objektdetektion verwendet werden. Es fehlt also eine automatisierte Auswertung/Vergleich der Test-Ergebnisse mehrerer Operatoren.

2.7.3 Bewertung der Methode

Im Bereich der automatisierten „Open Loop“ Testabläufe ist die Vereinheitlichung und Schaffung gemeinsamer Standards zum Austausch von Anforderungsdokumenten und Testspezifikationen im Vergleich zu allen bisher untersuchten Entwicklungsphasen im automobilen Software-Entwicklungsprozess, beziehungsweise den dort untersuchten Testmethoden und Testprozessen, am wenigsten fortgeschritten. Dies liegt nicht nur daran, dass es eine Fülle an proprietären Skripten zur Steuerung automatisierter Testabläufe und der Auswertung der Testergebnisse gibt, sondern vielmehr daran, dass jede Abteilung auf eigene Art und Weise, meistens aber nicht formal, wichtige Bestandteile der Testspezifikation definiert. Wie bereits erwähnt, werden die Fahrszenarien in umfangreichen (70-100 Seiten) Fahrszenarien-Katalogen spezifiziert. Diese sind meistens Word-Dokumente, in denen mit Piktogrammen, Tabellen und kurzen Textbeschreibungen die einzufahrenden Fahrszenen erklärt sind. Eine sinnvolle zentralisierte Ablage und Verwaltung für diese ist nicht gegeben und in der Form kaum möglich. Somit müssen oftmals für jede neu zu entwickelnde Funktion komplett neue Fahrszenenkataloge geschrieben werden, wobei viele von den bereits im Rahmen der Entwicklung einer anderen Funktion mit geringen Anpassungen hätten übernommen werden können. Der Grad der Wiederverwendbarkeit der Testdatenbeschreibungen und im Allgemeinen der Testspezifikationsbeschreibungen ist folglich nicht sonderlich hoch.

Eine andere Frage, die entsteht, ist, wie man auf den großen Testdaten-Mengen in akzeptabler Zeit detaillierte Suchvorgänge ausführen kann. Der heutige Stand erlaubt es, grundlegende Suche nach solchen in einer relationalen Datenbank [Ent06] abgelegten Metainformationen

wie beispielsweise Wetterverhältnisse, Fahrbahnzustände, Fahrzeugkonfiguration durchzuführen. Eine Suche nach bestimmten Fahrszenen, also einer Abfolge von Fahrmanövern, ist allerdings nicht gegeben. Nun würde sich sofort anbieten, diese Suche auf die Messdaten-Ebene zu verlagern. Dies erscheint jedoch selbst bei dem heutigen fortgeschrittenen Entwicklungsstand relationaler Datenbanken und ständig sinkenden Speicherkosten als völlig unrealistisch, da allein schon das direkte Verwalten der Messdaten von 10-20 Szenarien in einer relationalen Datenbank zu inakzeptablen Anfragezeiten führen würde. Man könnte zwar alternativ die Datenbank umgehen und die Suche direkt in einer Messdatei durch beispielsweise einen Skript realisieren. Dies würde jedoch auch zu längeren (meistens über Nacht) Bearbeitungszeiten führen. Eine lebensfähige Alternative scheint daher, die Metabeschreibung von Testdaten so zu erweitern, dass eine detallierte Suche nach diesen möglich wird.

Abschließend werden die Defizite einzelner untersuchter Testverfahren zusammengefasst:

- Alle untersuchten Testverfahren verwenden proprietäre und zielplattformabhängige Notationen zur Beschreibung von Testspezifikationen. Die Einheitlichkeit der Test-Spezifikationsbeschreibung ist nicht gegeben.
- Die komplexe Struktur des Software-Entwicklungsprozesses wird in den aktuellen Testspezifikationsbeschreibungen nicht genügend berücksichtigt.
- Aktuell eingesetzte Modell-basierte Ansätze (MiL) erlauben die Transformation der Testspezifikation in ein einziges Zielformat.
- Die Lesbarkeit und somit auch der Verständlichkeitsgrad untersuchter plattformspezifischer Notationen ist gering.
- Der Übergang zwischen MiL, SiL und HiL ist immernoch lückenhaft, also nicht durchgängig.
- Der Übergang von realen Testfahrten zu „virtuellen“ ist zurzeit mit einem hohen manuellen Aufwand verbunden.
- Einige der Testspezifikationen besitzen einen zu niedrigen Grad an Formalität, um beispielsweise in den nachgelagerten Software-Entwicklungsphasen wiederverwendet zu werden.

Kapitel 3

Stand der Wissenschaft

Aus den im Kapitel „Bestandsaufnahme in der Anwendungsdomäne“ festgestellten Defiziten ergeben sich folgende wissenschaftliche Fragestellungen, auf welche in diesem Kapitel Antworten gegeben werden:

- Welche wissenschaftliche Ansätze gibt es zur formalen Definition von Testspezifikationen? Inwieweit berücksichtigen sie die komplexe Organisationsstruktur eines Industrieunternehmens?
- Mit welchen Ansätzen lässt sich die Formatunabhängigkeit der formalen Beschreibung von Testspezifikationen errreichen?
- Wie lassen sich formatunabhängige Spezifikationsbeschreibungen in unterschiedliche formatabhängige umwandeln?
- Neben der Formalisierung der eigentlichen Testspezifikationsbeschreibung muss ebenfalls untersucht werden, welche Prozess-basierten Ansätze existieren, um die Verwendung einer formalen Testspezifikationsbeschreibung in den täglichen Gebrauch in einem Industrieunternehmen abteilungsübergreifend zu integrieren.
- Inwieweit sind alle in diesem Kapitel untersuchten Methoden dazu geeignet, die im vorigen Kapitel beschriebenen Defizite zu beheben?

Bevor mit den Untersuchungen zu den bestehenden wissenschaftlichen Ansätzen zur formalen Definition von Testspezifikationen begonnen wird, muss zweierlei erarbeitet werden. Zum einen muss genau definiert werden, woraus eine Testspezifikation besteht. Dies geschieht im Abschnitt 3.1. Im nächsten Schritt muss mit Hilfe oben definierter wissenschaftlicher Fragestellungen ein

Kriterienkatalog (Abschnitt 3.2) erarbeitet werden. Anhand dieses Katalogs werden die in diesem Kapitel untersuchten Ansätze auf ihre Eignung zum Erreichen der im Kapitel 1 definierten Ziele bewertet (Abschnitt 3.3). Wie schon angedeutet, müssen nicht nur die formalen Methoden zur Definition von Testspezifikationen untersucht werden, sondern auch Prozessmodellierungsmethoden, welche eine Einführung und sinnvolle Verwendung dieser im alltäglichen Leben eines Industrieunternehmens ermöglichen. Dies ist Gegenstand des Abschnitts 3.3.2.

3.1 Begriffsklärung: Testspezifikation

In der Literatur wurden sehr viele Definitionen der Testpezifikation gefunden, die auch äußerst unterschiedlich verwendet werden. Für die Bewertung der hier untersuchten Verfahren muss jedoch eine gemeinsame Begriffsbasis geschaffen werden. Deshalb wird basierend auf den in Kapitel 2 untersuchten Verfahren folgende Definition des Begriffs „Testspezifikation“ vorgenommen. Eine Testspezifikation ist hier eine Sammlung von Artefakten, die dazu notwendig sind, einen manuellen oder automatisierten Testvorgang vorzubereiten, auszuführen, auszuwerten und schließlich zu dokumentieren. Je nach Intention können die Artefakte formell oder informell definiert sein. In der Abbildung 3.1 sind einzelne Artefakte aufgelistet. Es muss gleich erwähnt werden, dass Anforderungen an das zu testende System kein unmittelbarer Bestandteil einer Testspezifikation sind. Sie bilden aber den Ausgangspunkt eines jeden Testvorgangs sowie aller Aktivitäten, die mit ihm verbunden sind. Mit Hilfe von Testfällen wird nachgewiesen, dass bestimmte vorher schriftlich festgehaltene Bedingungen im Sinne der Anforderungsdefinition erfüllt oder nicht erfüllt werden. Jeder Testfall sollte also auf die entsprechende Anforderung verweisen, die er testet. Trotz zahlreicher wissenschaftlicher Versuche, die Anforderungsspezifikationen zu formalisieren, um daraus direkt Testfälle abzuleiten, wird in der Praxis heutzutage meistens noch immer mit strukturierten Texten gearbeitet (siehe Kapitel 2). Im Folgenden wird nun jedes der dargestellten Artefakte genauer beschrieben.

3.1.1 Testdaten

Unter Testdaten versteht man die Eingangsdaten, welche direkt in das SUT eingespeist werden, um anschließend dessen Verhalten zu beobachten und auszuwerten. Es wurde die in Abbildung 3.2 dargestellte und für diese Arbeit relevante Klassifikation von Testdaten vorgenommen. Grundsätzlich wird zwischen zwei Arten von Testdaten unterschieden: Manuelle und automatisch erzeugte. Bei der manuellen Definition handelt es sich um Definition von diskreten Größen, also numerische oder textuelle Werte. Die manuellen Testdaten können unter Verwendung bestimmter Verfahren

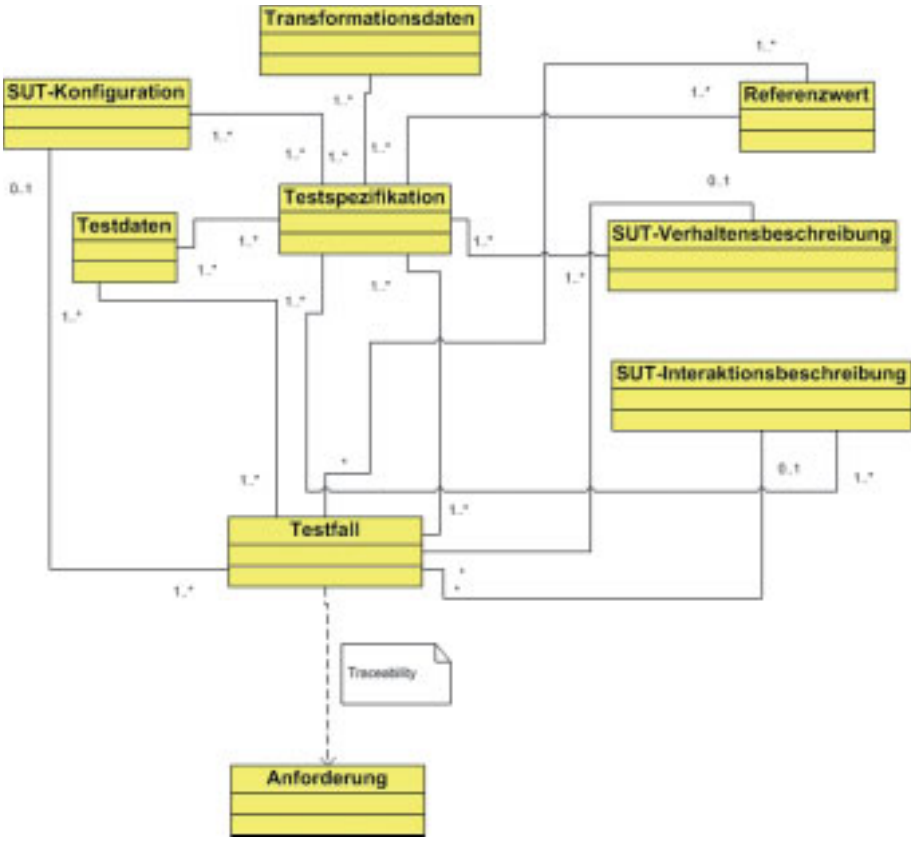


Bild 3.1: Bestandteile einer Testspezifikation

spezifiziert werden wie beispielsweise durch die im Kapitel 2 erwähnten CTE oder UML. Andererseits können die Testdaten auch ohne Anwendung spezifischer Methoden textuell definiert werden. Dazu gehört die Definition der Testdaten in Tabellen, Skripten oder auch als strukturierter Text. Die automatisch erzeugten Testdaten können zum einen aus den Verhaltensbeschreibungen oder aus den Interaktionsbeschreibungen des SUT mit der Umwelt abgeleitet werden (siehe 3.2). Zum anderen existieren seit geraumer Zeit so genannte Generatoren, welche die benötigten Test-Datensätze automatisch erzeugen. Dabei kann es sich sowohl um diskrete als auch kontinuierliche Werte (z.B. Signalverläufe) handeln. Eine andere Kategorie bilden die Testdaten, welche durch diverse Messungen entstanden sind.

3.1.2 SUT-Konfiguration

Unter SUT-Konfiguration wird eine Menge an Parametern verstanden, die spezifiziert werden muss, um einen Black-Box [SL05] Testvorgang durchführen zu können. Die Parameter betreffen

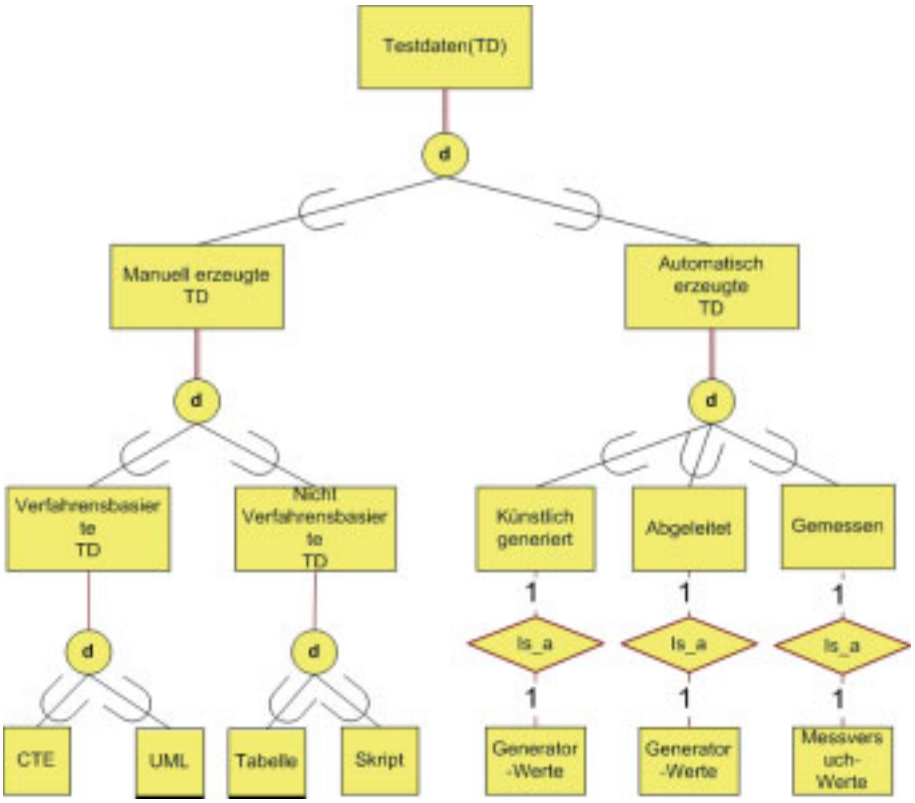


Bild 3.2: Klassifikation der Testdaten

diverse Systemeigenschaften und können für denselben Testdatensatz variiert werden. Dies führt dazu, dass ein Testvorgang mehrmals mit unterschiedlichen SUT-Parametern ausgeführt werden kann. Die Parametervariation spielt vor allem bei reaktiven Systemen eine große Rolle (Kapitel 2).

3.1.3 Transformationsdaten

Transformationsdaten gehören zwar nicht unmittelbar zu einer Testspezifikation, sie sind aber dann von Bedeutung, wenn diese auf eine bestimmte Zielplattform abgebildet wird. Meistens wird in den Transformationsdaten zusätzlich spezifiziert, wie die einzelnen an einem konkreten Testablauf beteiligten Anwendungen parametrisiert sind. Dazu können folgende Angaben gehören:

- Zugangsdaten zu Datenbanken, in welchen die Test- oder Referenzdaten verwaltet werden.
- Parametrisierung für die Entwicklungsumgebung, unter welcher das SUT läuft.

- Parametrisierung für die Testausführungsumgebung, welche den kompletten automatisierten Testvorgang steuert.
- Parametrisierung für die Werkzeuge, welche an der Testauswertung beteiligt sind.

Außerdem muss bei der Transformation spezifiziert werden, wie die einzelnen Bestandteile der kompletten Test-Spezifikation im Zielformat auszusehen haben.

3.1.4 Referenzwert

Referenzwert oder auch SOLL-Wert dienen dazu, um speziell bei den Black-Box Testfällen das gewünschte Verhalten des Systems zu spezifizieren. Referenzwerte werden meistens aus den Anforderungen an das SUT abgeleitet. Die Soll-Werte dienen als Grundlage der Testauswertung, nach welcher festgelegt wird, ob der Test bestanden wurde oder nicht. Die Soll-Werte lassen sich auf eine ähnliche Art und Weise wie die Testdaten klassifizieren. Zum einen exisitieren auch hier manuell erzeugte Referenzwerte. Diese können als numerische oder textuelle Konstanten definiert werden. Bei manchen Verfahren werden Referenzwerte in den Verhaltensmodellen abgelegt. Somit kann man nach jedem automatisch erzeugten und in das SUT eingegebenen Testdatensatz dessen Reaktion überprüfen, indem der erzeugte Ausgang mit dem Referenz-Datensatz verglichen wird. In bestimmten Fällen definieren die Verhaltens- und Interaktionsmodelle ebenfalls das Soll-Verhalten des SUT. Diese Beobachtung ist allerdings nicht für alle möglichen Testbereiche gültig und vor allem nicht für sämtliche Typen von reaktiven Systemen, auch wenn bei klassischen modell-getriebenen Testvefahren strikt gefordert ist, dass die Modelle eben das Soll-Verhalten definieren (Siehe Kapitel 4). Zum anderen können die Referenzwerte auch automatisch generiert werden. Dies geschieht beispielsweise während eines Messversuchs.

3.1.5 Verhaltens- und Interaktionsbeschreibung des SUT mit der Umwelt

Diese Artefakte sind nicht obligatorisch und sie werden lediglich bei spezifischen Testmethodiken wie dem modell-getriebenen Testen [Bak05] (siehe Unterabschnitt „Modell-basierte Verfahren“ für ausführlichere Darstellung) angewandt. Das Verhaltensmodell des SUT wird redundant zur bereits bestehenden Code-Implementierung des SUT erzeugt. Diese Redundanz ist erwünscht, denn man möchte letztendlich das im Modell beschriebene Verhalten am eigentlichen SUT überprüfen. Würde man das SUT-Code zu 100 Prozent aus dem Verhaltensmodell ableiten, wie dies beispielsweise bei Model-Driven Architecture (MDA) [MSUW04] der Fall ist, dann würde das modell-basierte Testen sich erübrigen, denn der Code würde eindeutig das im Modell beschriebene

Systemverhalten nachbilden. Auf Modellen können bestimmte Überdeckungskriterien definiert werden, ähnlich wie sie bei White-Box Tests üblich sind [Lig09]. Gemäß diesen Kriterien werden im nächsten Schritt automatisch Testdaten (engl. traces) erzeugt. Im Gegensatz zu Verhaltensmodellen wird bei Interaktionsmodellen nicht das interne Verhalten des SUT modelliert, sondern das externe, also wie das SUT mit bestimmten Objekten aus seiner Umwelt interagiert. Solche Modelle können auf verschiedenen Abstraktionsebenen definiert werden: Es können sowohl eine einzige Systembenutzung als auch alle möglichen oder sinnvollen Systembenutzungen modelliert werden. Im nächsten Schritt können aus solchen Modellen Testdaten abgeleitet oder generiert werden.

3.1.6 Testfall

Testfall wird durch Testdaten und Referenzwerte definiert. Oftmals müssen zusätzlich zu den Testdaten noch Angaben dazu gemacht werden, wie man das SUT in einen bestimmten Zustand bringt, damit der eigentliche Testfall überhaupt stattfinden kann. Ebenfall ist es oft notwendig zu spezifizieren, was mit dem SUT nach der Beendigung des Testfalls geschehen soll. Solche Angaben gehören ebenfalls zu einem Testfall. Ebenfalls können die Testfälle in einen Testlauf organisiert werden. Ein Testlauf ist eine geordnete Menge von Testfällen.

3.1.7 Anforderungen an das zu testende System

Laut [Poh08] ist eine Anforderung:

- „Eine Bedingung oder Eigenschaft, die ein System oder eine Person benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen.
- Eine Bedingung oder Eigenschaft, die ein System oder eine Systemkomponente aufweisen muss, um einen Vertrag zu erfüllen oder einem Standard, einer Spezifikation oder einem anderen formell aufgelegten Dokument zu genügen.
- Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft wie in den obigen zwei Punkten definiert“.

3.2 Kriterienkatalog

Die Kriterien zur Bewertung der untersuchten Methoden dienen dazu, eine möglichst objektive Aussage darüber zu treffen, inwieweit diese zur Lösung der in dieser Arbeit beschriebenen

Probleme geeignet sind. In der Literatur [FrvL03] existieren diverse Vorschläge dazu, wie man Kriterien sinnvoll nach Gruppen strukturieren kann. So wird es beispielsweise nach formalen, anwender-, und anwendungsbezogenen Kriterien unterschieden. Diese in [FL03] vorgeschlagene Klassifikation erscheint ebenfalls für die vorliegende Arbeit als sinnvoll. Die Bewertungskriterien selbst werden zum einen aus den wissenschaftlichen Zielen der Arbeit abgeleitet. Zum anderen werden sie aus [FL03] wegen Allgemeingültigkeit mit gewissen Anpassungen übernommen. Im Weiteren werden nun die einzelnen Kriterien genauer erläutert.

3.2.1 Formale Kriterien

Die Testspezifikationen lassen sich maschinell verarbeiten nur dann, wenn sie formal definiert sind. Deshalb muss eine Reihe von Kriterien definiert werden, anhand welcher sich die Formalität der untersuchten Beschreibungsmethode überprüfen lässt.

- ***Korrektheit:*** Die Definition einer Beschreibungsmethode genügt dem Kriterium der Korrektheit, wenn sie die eindeutige Identifikation (zumindestens syntaktisch) unzulässiger Testspezifikationen gewährleistet. Auf Basis einer korrekt definierten Beschreibungsmethode lassen sich somit Algorithmen zur Validierung entwickeln, die eine hohe Qualität der später erzeugten Testspezifikationen garantieren.

- ***Vollständigkeit:*** bedeutet in diesem Kontext, dass alle Konstrukte der Beschreibungsmethode hinreichend definiert sind. Die Vollständigkeit der Methodendefinition ist für einen sinnvollen Einsatz unabdingbar, da sonst keine objektiven Beschreibungsregeln ableitbar sind. Außerdem müssen diese Konstrukte eindeutig definiert sein.

- ***Einheitlichkeit:*** Eine Beschreibungsmethode, die ähnliche Sachverhalte nicht mit ähnlichen Prinzipien (oder Konstrukten) abbildet und so Redundanzen innerhalb der Methodenspezifikation schafft, kann u.U. an Prägnanz/Verständlichkeit gewinnen. Die Gefahr ist jedoch, dass die resultierenden Testspezifikationen trotz ähnlicher semantischer Inhalte sich signifikant unterscheiden und damit an Vergleichbarkeit verlieren. Daher muss eine Beschreibungsmethode möglichst klar, d.h. so einfach wie möglich, und redundanzfrei spezifiziert sein.

- ***Wiederverwendbarkeit:*** Bedeutet in diesem Kontext, dass die Beschreibungsmethode es ermöglichen soll, die Testspezifikationen unabhängig von einem bestimmten Zielformat zu definieren. Dies führt dazu, dass in der plattformunabhängigen Beschreibung die Details der

Zielplattform „abstrahiert werden“, was zu einem erhöhten Grad an Wiederverwendbarkeit führt.

- ***Wartbarkeit/Anpassbarkeit:*** Die Wartbarkeit beziehungsweise die Anpassbarkeit sind wichtige Aspekte im Kontext dieser Arbeit. Denn angesichts komplexer Unternehmensstruktur muss es möglich sein, die Beschreibungsmethode ohne einen beträchtlichen Aufwand um weitere Elemente (z.B. Attribute, Beschreibungskonstrukte usw.) zu erweitern. Nach [Mü07] ist die Komplexität der Beschreibungsmethode ein weiterer zu berücksichtegender Aspekt der Warbarkeit, da bei hoher Komplexität die Änderungen an der Testspezifikation schlecht zu erkennen sind.

3.2.2 Anwenderbezogene Kriterien

Durch diese Kriterien soll eine Beurteilung möglich sein, inwieweit die betrachtete Beschreibungsmethode zum eingenständigen Einsatz für die Entwickler geeignet ist.

- ***Einfachheit:*** Eine Beschreibungsmethode ist dann einfach, wenn Sie in relativ kurzer Zeit zu erlernen ist. Einflussfaktoren für die Einfachheit sind dementsprechend die Anzahl der Konstrukte, Regeln oder die Syntax der Beschreibungsmethode. Allerdings gilt nicht generell, dass eine Beschreibungsmethode mit wenigen Elementen einfacher ist als eine Beschreibungsmethode mit mehr, da sonst die Komplexität spätestens beim Beschreibungsvorgang entsteht. Eine Beschreibungsmethode muss also einen gelungenen Kompromiss zwischen Generizität (komplexe Modellierung) und Spezifität (komplexe Erlernung) darstellen [Mü07].

- ***Anwendbarkeit:*** Eine Beschreibungsmethode muss auf den Einsatzbereich ausgerichtet sein und die notwendigen Konstrukte zur Abbildung der relevanten Sachverhalte in der Testspezifikation ermöglichen. So ist beispielsweise eine Methode zur Beschreibung statischer Strukturen nur schlecht für die Spezifikation dynamischer Abläufe geeignet. Klar im Vorteil sind hier Methoden, die für eine Anwendungsdomäne durch eigene Definitionen erweitert werden können.

- ***Verständlichkeit:*** Formale Testspezifikationen sollen als Kommunikationsgrundlage zwischen den unterschiedlichen am Testvorgang beteiligten Personengruppen dienen. Sie soll ebenfalls als Kommunikationsgrundlage zwischen den einzelnen Entwicklern agieren. Es ist daher für alle folgenden Arbeitsschritte entscheidend, dass der Entwickler auch versteht, was in der Testspezifikation abgebildet wurde. Dies lässt sich durch eine Beschreibungsmethode

unterstützen, welche die Begriffe der Fachdomäne aufgreift und so für den Entwickler leichter verständlich wird. Nach [FL03] ist eine verständliche Beschreibungsmethode dadurch gekennzeichnet, dass ihre Konzepte und Symbole direkt mit Begriffen korrespondieren, die dem Anwender vertraut sind.

- ***Anschaulichkeit:*** Anschauliche Testspezifikationen sind wichtig, um ein schnelles Verständnis des Inhalts durch den Anwender zu ermöglichen. Es soll daher für Bestandteile der Testspezifikation, welche dynamische Verhaltensaspekte repräsentieren, wie Testdaten-, Verhaltens-, und Interaktionsbeschreibung möglichst eine graphische Notation bevorzugt werden. Die entscheidenden Zusammenhänge, welche in den einzelnen Bestandteilen der Testspezifikation abgebildet werden, sollen auf den ersten Blick erkennbar sein. Anschaulichkeit wird ebenfalls dadurch verbessert, dass Abteilungs-, Projekt-, und Testartbezogene Beschreibungskonstrukte dem entsprechenden Entwickler angezeigt werden („Sichtenkonzept“) und nicht alle möglichen Informationen.

3.2.3 Anwendungsbezogene Kriterien

Eine Beschreibungsmethode soll nicht nur alle relevanten Aspekte der zu testenden Domäne abbilden können, sondern sie dient auch als Basis für weitere Anwendungen (Datenbanken, Transformatoren etc.)

- ***Ausdrucksmächtigkeit:*** Laut [Mü07] muss eine Beschreibungsmethode alle relevanten Elemente der abzubildenden Sachverhalte mit adäquater Genauigkeit darstellen können. Die notwendige Genauigkeit ist stark vom späteren Einsatzzweck abhängig und ist immer als Kompromiss zwischen Detaillierungsgrad und Einfachheit zu sehen. Generell ist es besser, wenn eine Beschreibungsmethode eine hohe Ausdrucksmächtigkeit besitzt, um die Möglichkeit zu haben, eine Testspezifikation iterativ anzureichern. Eine Beschreibungsmethode ist umso ausdrucksmächtiger, je besser sie auf die Anwendung zugeschnitten ist.

- ***Operationalisierbarkeit:*** Testspezifikationen dienen oft nicht nur zur Kommunikation oder Visualisierung, sondern können auch als Basis für weitere Aktivitäten wie zum Beispiel Zusammenstellung von SUT-Benutzungsprofilen, Ausführung von automatisierten Testabläufen oder Transformation in bestimmte Zielformate. Eine Beschreibungsmethode, die ausreichend formal ist, um diese Formen der Nutzung zu unterstützen, gilt als operationalisierbar. Das Kriterium ist für diese Arbeit wichtig, da der im Kapitel 4 entwickelte Ansatz nur dann anwendbar ist, wenn die Beschreibungsmethode operationalisierbar ist.

- ***Erweiterbarkeit:*** Generell existieren Testspezifikationen für mehrere verschiedene Testverfahren, Projekte und Abteilungen im kompletten Software-Entwicklungsprozess. Es ist daher sehr wichtig, dass sie leicht erweitert werden können. Konkret bedeutet dies beispielsweise, dass wenn eine neue Testart hinzukommt, dann muss die bestehende Beschreibungsmethode mit wenig Aufwand so erweitert werden können, dass auch für diese Testart im Unternehmen effektiv Testspezifikationen erstellt, verwaltet und weiterverwendet werden können. Es ist außerdem wünschenswert, dass die vorgenommenen Erweiterungen möglichst wenige Auswirkungen auf die vor der Erweiterung erstellten Testspezifikationen haben. Im schlimmsten Fall ist dann mit einer Neudefinition aller bestehenden Testspezifikationen zu rechnen. Dies sollte selbstverständlich vermieden werden.

3.3 Untersuchung der bestehenden Ansätze

3.3.1 Beschreibungsmethoden zur Definition von Testspezifikationen

Die untersuchten Beschreibungsmethoden zur Definition von Testspezifikationen lassen sich grob in zwei Kategorien unterteilen: Textbasierte und graphische. Da geschichtlich gesehen die textuellen Methoden als erste entstanden sind, wird in Abschnitt 3.3.1.1 zunächst auf diese eingegangen. Im weiteren Verlauf wurden dann zunehmend graphenbasierte Formalismen eingesetzt, mit deren Hilfe die Definition von einzelnen Bestandteilen der Testspezifikation möglich ist. Diese werden im Abschnitt 3.3.1.2 behandelt.

3.3.1.1 Textuelle Methoden

Bereits Ende der siebziger Jahre unternahm man Versuche, vor allem in der Luftfahrtforschung, das SUT-Verhalten formal zu beschreiben, um entweder bestimmte Systemeigenschaften damit nachzuweisen (Model-Checking [CGP00]) oder aber um Testfälle daraus automatisch abzuleiten. Generell lassen sich textuelle Methoden folgendermaßen klassifizieren (Abbildung 3.3). Die Ausgangsbasis bildet oft die Interaktionsbeschreibung des zu testenden Systems [FAM06]. Diese Beschreibungen sind oft in Form von strukturierten Text oder tabellarisch dargestellt [FAM06]. Im nächsten Schritt wird daraus manuell das Verhaltensmodell des SUT erstellt (Oft als Zustandsautomat oder mit den Prozessalgebren formal beschrieben). Im nächsten Schritt werden dann meistens automatisch die Testfälle erzeugt. Auf der anderen Seite wurden bereits seit Ende der achtziger Jahre Versuche unternommen, dedizierte Testspezifikationssprachen mit Hilfe von Grammatiken zu definieren und diese dann in spezifische Formate umzuwandeln [BHO89]. Da

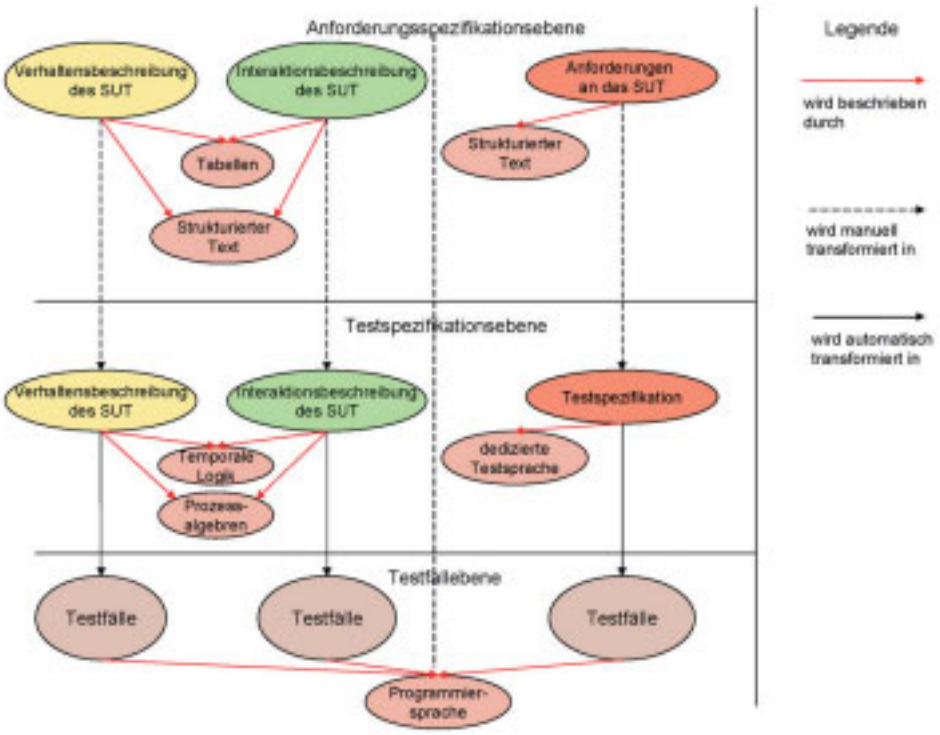


Bild 3.3: Klassifikation der textuellen Verfahren

es im Rahmen dieser Arbeit nur schwer möglich ist, auf alle möglichen Varianten der oben angeführten mathematischen oder der sprachbasierten Testspezifikationsdefinition einzugehen, werden im folgenden für sie einzelne relevanteste Beispiele behandelt. Generell lässt sich aber feststellen, dass die textbasierten Beschreibungsmethoden im Hinblick auf die in 3.2 definierten Kriterien sehr ähnliche Eigenschaften aufweisen.

Software Cost Reduction

Software Cost Reduction (SRC) Methodik [Hei02] wurde in 1978 erfunden, um die Anforderungen zunächst in Flugzeugbau zu spezifizieren. Die SCR-Anforderungsspezifikation beschreibt sowohl die SUT-Umgebung (also das Interaktionsverhalten) als auch das gewünschte Systemverhalten.

Grundsätzlich besteht die Methode aus vier Elementen: Bedingung, Ereignis, Modus und Term. Eine Bedingung ist ein Prädikat, das eine bestimmte Systemeigenschaft für eine messbare Zeitperiode charakterisiert. Ein Ereignis tritt dann ein, wenn sich der Wert der Bedingung von „wahr“ auf „falsch“ ändert und umgekehrt. Ein Modus ist als eine Klasse von Systemzuständen

definiert und der Term als ein Textmakro. Die Bezeichnung „@T(c)“ wurde eingeführt, um auszudrücken, dass die Bedingung „c“ „wahr“ und „@F(c)“ um zu beschreiben, dass „c“ falsch wird. In SRC wird das SUT-Verhalten sowie das Interaktionsverhalten mit Hilfe von Tabellen definiert. Beispielhaft ist in der Abbildung 3.4 eine Modus-Übergangstabelle dargestellt. Um eine

Alter Modus	Ereignis	Neuer Modus
EGO fährt konstant	@T(V_{ego}>$V_{testobjekt}$)	EGO schert aus
EGO beschleunigt	@T(V_{ego}=$V_{testobjekt}$)	EGO fährt konstant
EGO bremst ab	@T(V_{ego}>$V_{testobjekt}$)	EGO steht still
EGO fährt konstant	@T(V_{ego}<$V_{testobjekt}$)	EGO beschleunigt

Bild 3.4: SRC: Beispiel

bestimmte Semantik für die SRC-Tabellen festzulegen, wurde das SRC-Modell erfunden. Dieses erlaubt, eine formelle Analyse von den Anforderungsdefinitionen durchzuführen. Konkret bedeutet dies, dass daraus Testfälle abgeleitet werden können. Im SRC-Modell wird ein Software-System Σ als ein Zustandsautomat repräsentiert $\Sigma = (S, S_0, E^m, T)$, wobei S eine Menge von Zuständen ist, $S_0 \subseteq S$ der Startzustand, E^m eine Menge von beobachteten Ereignissen ist und T eine Menge von erlaubten Transitionen darstellt. Jeder Zustand wird durch eine Menge an Zustandsvariablen beschrieben. Auf den SRC-Modellen lassen sich bestimmte Eigenschaften definieren, welche dann automatisiert überprüft werden können. Beispielsweise: “Bis das EGO-Fahrzeug den Spurwechsel vollzogen hat, muss der Blinker an sein„.

In [GH99] wird ein Vorschlag gegeben, wie man mit Hilfe solcher Systemeigenschaften Testfälle für das in Code implementierte SUT ableiten kann. Dazu wird der klassische Model-Checker [FG09] auf eine etwas ungewöhnliche Art und Weise verwendet. Normalerweise wird ein Model-Checker zur Analyse von zustandsbasierten SUT-Verhaltensbeschreibungen herangezogen. Überprüft werden bestimmte Systemeigenschaften, wie sie beispielsweise oben angeführt wurden. Wenn der Model-Checker alle erreichbaren Zustände des Systems analysiert und dabei keine Eigenschaftsverletzungen feststellt, dann gilt die Eigenschaft als erfüllt. Wenn der Model-Checker im Gegenteil einen Zustand findet, der die Eigenschaft verletzt, dann gibt er ein so genanntes Gegenbeispiel zurück, also einen Pfad, der mit dem Startzustand beginnt und mit den Zustand endet, in dem die Eigenschaft verletzt ist.

Nun wird der grundsätzliche Vorgang bei der Ableitung von Testfällen skizziert. Dazu werden die Systemeigenschaften von dem Testingenieur in temporärer Logik definiert. [GH99] schlägt dazu die Computational Tree Logic (CTL) [CGP00] als Methode. Dabei werden die Systemeigenschaften negiert, von denen nachgewiesen wurde, dass sie gültig sind. Auf diese Weise veranlasst man den Model-Checker dazu, ein Gegenbeispiel zu generieren, also eine Sequenz an Ereignissen, welche in einen Zustand führen, in dem die Systemeigenschaft verletzt ist. Im nächsten Schritt wird die generierte Sequenz als Testdaten für einen Black-Box basierten Test an der Code-Implementierung des SUT verwendet. Getestet wird, ob die in CTL definierte Eigenschaft gültig ist, also ob sich das System nach den eingegebenen Testdaten in einem Zustand befindet, in dem die Eigenschaft verletzt ist.

Communication Sequential Processes (CSP)

Prozessalgebra CSP [Hoa78] ist eine formale Notation zur Beschreibung von verteilten Systemen. Der Grundgedanke von CSP basiert auf der Annahme der ereignisbasierten Kommunikation. Die Ereignisse werden mit Kleinbuchstaben gekennzeichnet. Ein anderer Grundstein von CSP sind Prozesse. Diese können von Ereignissen ausgelöst werden und sie laufen sowohl parallel als auch sequentiell ab. Diese werden mit Großbuchstaben gekennzeichnet. Prozesse können sowohl das Systemverhalten als auch das Interaktionsverhalten des SUT beschreiben. Im folgenden wird nun die vereinfachte CSP Prozessgrammatik erläutert (siehe Codeabschnitt unten).

```
0  P::=STOP
10   | SKIP
20  | a -> P
30   | P |~| P
40  | P [] P
50  | P [|X|] P
60  | Q
```

Es existieren zwei Grundprozesse wie STOP und SKIP: Der erstere definiert eine Situation, in der die teilnehmenden Prozesse miteinander nicht kommunizieren können. Der zweitere bezeichnet die Prozessbeendigung nach erfolgreicher Kommunikation. Die einfachste Verhaltensbeschreibung würde folgendermaßen aussehen: a$->$P. Dies bedeutet, dass ein Prozess der durch $a-> P$ definiert ist, uneingeschränkt auf das Eintreten des Ereignisses a wartet. Sobald a eintritt, wird mit P fortgefahren. Alternatives Verhalten wird durch zwei Auswahloperatoren gekennzeichnet: Eine nicht-deterministische Auswahl ($|\sim|$) und eine deterministische ([]). Ein Prozess, der durch P

$(| \sim |)\ Q$ definiert wird, verhält sich entweder als P oder als Q unabhängig von seiner Umgebung. Das heißt, dass der Prozess, der durch $P\ (| \sim |)\ Q$ definiert ist, sich erst als P oder Q verhält und erst dann kann die Umgebung damit interagieren. Möchte man die Ausführung von P oder Q von der Umgebung abhängig machen, dann muss folgende Notation verwendet werden $P[]Q$. Die parallele Ausführung eines Prozesses der durch P und Q definiert ist, wird definiert als $P\ [|a, b|]\ Q$. Dies ist nichts Anderes als parallele Komposition von zwei Prozessen, welche über Ereignisse a und b synchronisiert sind. Alle anderen Ereignisse laufen in den jeweiligen Prozessen unabhängig ab.

[FAM06] schlägt folgende Vorgehensweise zur formalen Definition von den Testspezifikationen und der Ableitung der Testfälle aus diesen vor (Abb. 3.5). Im ersten Schritt werden in

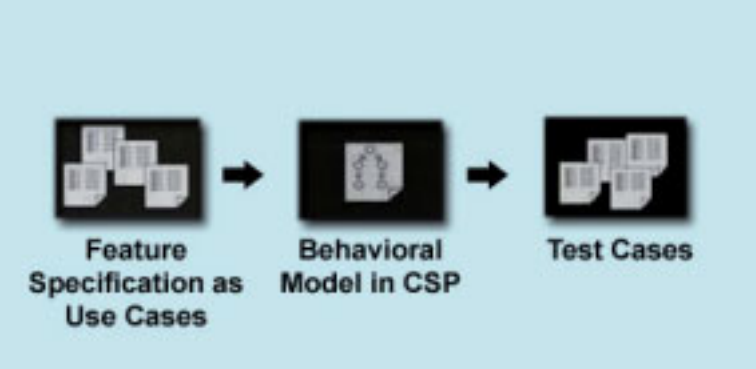


Bild 3.5: CSP: Vorgehensweise zur Generierung von Testfällen [FAM06]

tabellenartigen Dokumenten einzelne mögliche Interaktionen eines Anwenders mit dem System definiert, Systemzustände sowie Systemrekation auf eine bestimmte Aktion seitens des Anwenders. Diese Interaktionen werden anschließend in CSP formuliert. Beispielsweise würde eine solche Formulierung für einen Einschermanöver folgendermaßen aussehen:

- tor:TO_r = decelerates$- >$tor.drives-constantly
- toa:TO_a = accelerates$- >$toa.drives-constantly
- sut:EGO_e = (tor.decelerates$- >$toa.accelerates)$- >$sut.ego-cuts-in
- ego:EGO_e[decelerates]tor:TO_r = tor.decelerates$- >$(sut.ego-cuts-in$- >$ tor.drives-constantly)
- ego:EGO_e[accelerates]toa:TO_a = toa.accelerates$- >$(sut.ego-cuts-in$- >$ toa.drives-constantly)

Durch dieses Beispiel wurde folgende Fahrszene beschrieben. Auf der rechten Nachbarspur von dem Versuchsfahrzeug (EGO) befinden sich zwei Testobjekte: einer auf derselben Höhe wie das EGO (TO_r), der andere vor ihm (TO_a). In CSP wird nun folgendes beschrieben: Nachdem TO_r abbremst und TO_a beschleunigt, schert das EGO (EGO_e) in die sich gebildete Lücke auf der rechten Nachbarspur ein. Aus solchen Beschreibungen wird nach [FAM06] ein so genanntes Verhaltensmodell erzeugt. Es wird zwar nicht näher spezifiziert, wie ein solches Modell erzeugt wird und vor allem wie dieses aussieht, aber generell sind in solchen Verhaltensmodellen meist durch endliche Automaten alle möglichen Interaktionen des Benutzers mit dem SUT beschrieben. Aus diesen werden im nächsten Schritt durch dedizierte Algorithmen Testfälle abgeleitet. Ein Testfall ist dann ein bestimmter Pfad im Verhaltensmodell.

Test Specification Language

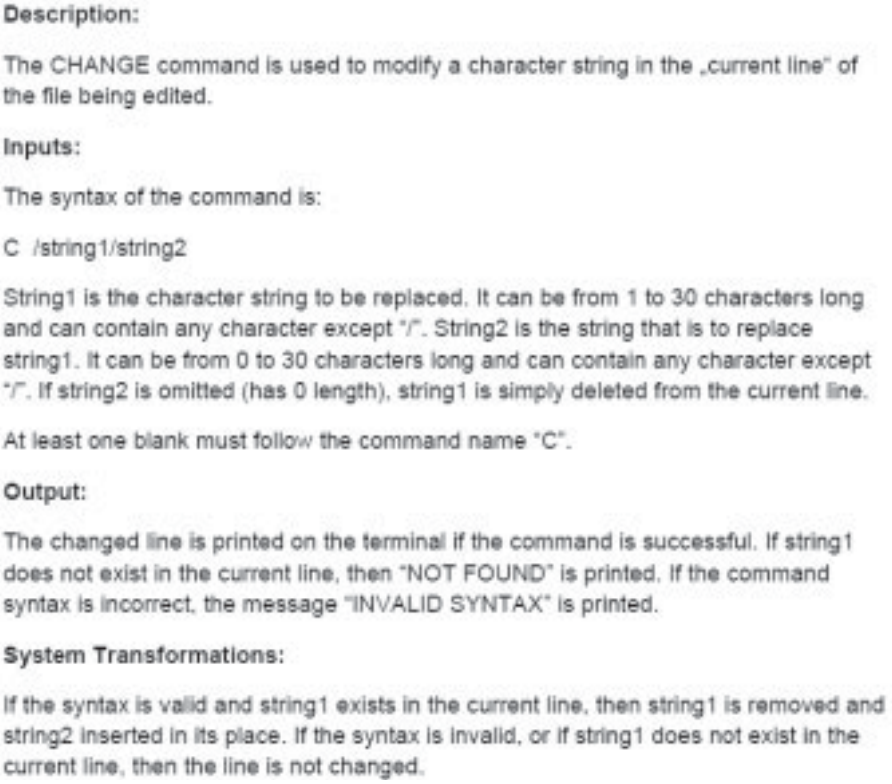
Im Jahr 1989 wurde die Testspezifikationssprache (TSL) in der Forschungsabteilung von International Business Machines [Lu94] erfunden. Sie wurde im weiteren Verlauf ebenfalls von der Forschungsabteilung der Siemens AG weiterentwickelt und verfeinert. TSL ist eine textbasierte Sprache, die eine formale Definition der Testspezifikationen für funktionales Testen (synonym zu Black-Box Testen) erlaubt. Der Formalitätsgrad der TSL ermöglicht es, aus der Testspezifikationsbeschreibung ausführbare Testskripte zu erzeugen. Die grundsätzliche TSL Struktur [BHO89] ist in der Abbildung 3.6 dargestellt. Die TSL unterteilt sich in mehrere Abschnitte, welche verschiedene Aspekte des SUT sowie dessen Umgebung repräsentieren. Grundsätzlich gibt es vier wichtige Beschreibungselemente in der TSL: „Test“, „Parameter“, „Environment“ und „Result“. Das „Test“-Element bezeichnet den Anfang jeder Testspezifikation. Nach [Balcer] stellt der Test-Element ein Code-Template dar, der bei der Skripterzeugung mit konkreten Belegungen für die dort spezifizierten Variablen instanziiert wird. Da TSL speziell auf Äquivalenzklassen-Methode basiert [Lig09], erlaubt das Parameter-Element die Definition der Äquivalenzklassen (<choice>-Element) sowie bestimmter Räpresentanten aus diesen. Das „Parameter“-Element wird u.a. zur Spezifikation der Testdaten eines SUT verwendet. Zur Spezifikation der Umgebungsbedinungen des SUT wird das „Environment“-Element verwendet. Typische Umgebungsbedingungen beschreiben laut [BHO89] den Zustand des Betriebssystems auf dem das SUT läuft (Anzahl der laufenden Prozesse, aktuelle Hauptspeicher-Belegung etc.) Vor der Ausführung eines jeden Testfalls müssen bestimmte Umgebungsbedingungen (meistens des Betriebssystems) hergestellt werden. Dazu dient das Beschreibungselement <setup-cleanup> innerhalb des Elementen „Environment“. Ebenfalls dient das <setup-cleanup> Element dazu, um nach der Ausführung eines Testfalls bestimmte Aufräumaktionen durchzuführen (zum Beispiel Speicherfreigabe für die

```
TEST <test-name>
      [<description-string>]
      [SETUP {<string>}]
      FORM {<string>}
      [CLEANUP {<string>}]

PARAMETER <param-name>
      [<description-string>]
      [<setup-cleanup>]
            * <choice-1>
                  [<value-list>]
                  [<setup-cleanup>]
            . . .
            * <choice-n>
                  [<value-list>]
                  [<setup-cleanup>]

ENVIRONMENT <environment-name>
      [<description-string>]
      [<setup-cleanup>]
            * <choice-1>
                  [<setup-cleanup>]
            . . .
            * <choice-n>
                  [<setup-cleanup>]

RESULT <result-name>
      [<description-string>]
      [<setup-cleanup>]
      [VERIFY <verification-code>]
      IF <result-expression-1>
            . . .
      IF <result-expression-n>
```

Bild 3.6: TSL: Grundsätzlicher Aufbau [BHO89]

Variablen). Schließlich wird im Beschreibungselement „Result“ (Klausel „<result-name>“) das erwartete Ergebnis für jeden spezifizierten Testfall definiert. Dazu kann zweierlei spezifiziert werden. Zum einen werden durch „IF<result-expression-n>“ Bedingungen Kombinationen von Eingangsparametern festgelegt, die das erwartete Ergebnis produzieren und zum anderen werden in der Klausel „Verify“ Auswertungsroutinen in Form von Code abgelegt. In [Lu94] wird der Einsatz von TSL am Beispiel einer konkreten bei IBM entwickleten Architektur vorgestellt. Dort werden die Testspezifikationen per Hand aus den textuellen Beschreibungen in TSL übertragen, um anschließend daraus die Testdaten zu generieren. Beispielhafte Testspezifikationen [Lu94] in englischer Sprache sind in der Abbildung 3.7 dargestellt. Es handelt sich um eine Funktion „CHANGE“, welche in einem textuellen Editor es erlaubt, die aktuelle Zeichenkette zu ändern. Nach der manuellen Übertragung der Testspezifikation in TSL wird mittels eines Code-Templates

ein ausführbares Skript zur automatisierten Durchführung von Tests erzeugt. Um die konkreten Testfälle strukturiert darzustellen, werden sie in einem „Test Specification Graph“ abgebildet. In diesem sind alle möglichen sinnvollen Kombinationen von Eingabewerten für die in TSL spezifizierten Variablen angegeben. Bei der Code-Generierung werden die sinnvollen Kombinationen von Eingabewerten in das erzeugte Skript eingebunden.



Description:

The CHANGE command is used to modify a character string in the „current line" of the file being edited.

Inputs:

The syntax of the command is:

C /string1/string2

String1 is the character string to be replaced. It can be from 1 to 30 characters long and can contain any character except "/". String2 is the string that is to replace string1. It can be from 0 to 30 characters long and can contain any character except "/". If string2 is omitted (has 0 length), string1 is simply deleted from the current line.

At least one blank must follow the command name "C".

Output:

The changed line is printed on the terminal if the command is successful. If string1 does not exist in the current line, then "NOT FOUND" is printed. If the command syntax is incorrect, the message "INVALID SYNTAX" is printed.

System Transformations:

If the syntax is valid and string1 exists in the current line, then string1 is removed and string2 inserted in its place. If the syntax is invalid, or if string1 does not exist in the current line, then the line is not changed.

Bild 3.7: TSL: Beispielhafte Testspezifikation in englischer Sprache [BHO89]

Testing And Test Control Notation (TTCN3)

TTCN [GHR+03] entstand in den 80 Jahren als Test-Spezifikationssprache in der Telekommunikationsbranche. Die erste Spezifikation dieser Sprache erschien im Jahr 2001. Inzwischen wird TTCN-3 aktiv in der Kommunikationsbranche angewandt, in welcher mit ihr ein breites Spektrum an Technologien abgedeckt wird (GSM, UMTS, ISDN, VoIP, ISDN, WiFi) [WDT+05]. Typisches Anwendungsgebiet von TTCN-3 ist das Black-Box Testen von Kommunikationsprotokollen, Web-Services, CORBA-basierten Plattformen [GHR+03]. TTCN-3 verfügt über drei verschiedene Repräsentationsformate: Hauptnotation (textuelle Programmiersprache), graphische

Notation(basierend auf Message Sequence Charts) sowie eine tabellenartige. Die wichtigste Darstellungsmöglichkeit ist selbstverständlich die textuelle, da sie von dedizierten TTCN-3 Compilern ausgeführt werden kann. Das wichtigste Konstrukt in TTCN-3 ist das Modul (siehe Abbildung 3.8). Der Definitionsteil eines Moduls ermöglicht die Definition von Testdaten, Datentypen, Testfällen

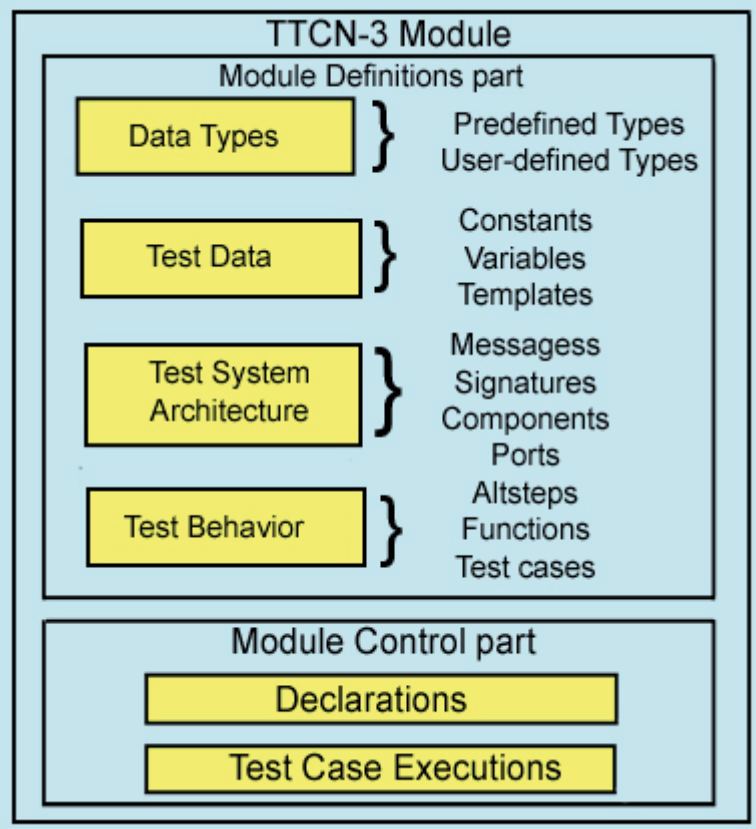


Bild 3.8: Das TTCN-3 Modul [ORLS06]

sowie den Testkomponenten. Die Testkomponente ist ein wichtiges Konzept im Rahmen der TTCN-3 Notation. Für jeden Testfall existiert eine Testkomponente. Ihr Verhalten wird in einem Testfall spezifiziert. Die Testkomponenten können parallel zu anderen Testkomponenten ablaufen und mit ihnen über den Portmechanismus kommunizieren. Es gibt zwei Kommunikationsarten in TTCN-3: Nachrichten-basierte und Prozedur-basierte. Testkomponenten können ebenfalls über ein abstraktes Interface auf das SUT zugreifen (Abbildung 3.9). Ein Testfall ist in TTCN-3 durch eine Menge an programmiersprachlichen Funktionen beschrieben. Jede programmiersprachliche Funktion wird durch eine Signatur spezifiziert. Da die Testfälle im Sinne des funktionalen Tests definiert sind, so stellen sie die gewünschte Interaktionsbescheibung des SUT mit der Umwelt dar. Durch ein Templatemechanismus ist es möglich, für jede Funktion innerhalb eines Test-

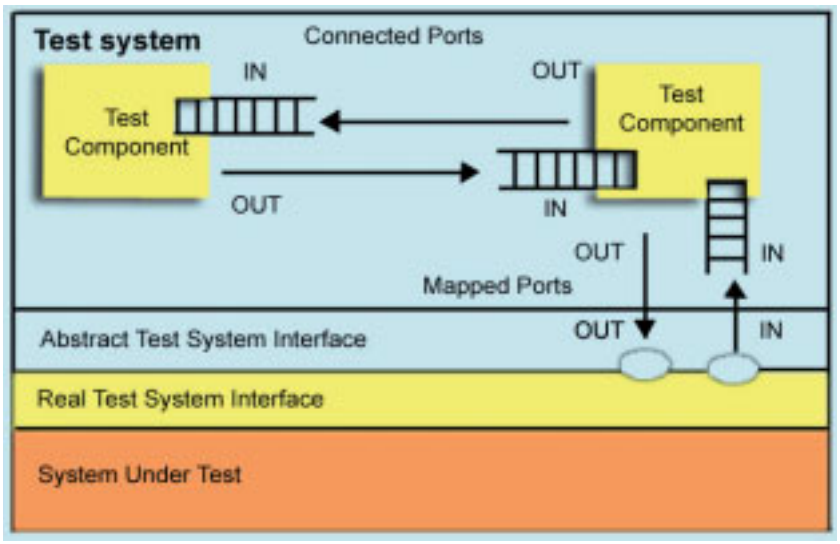


Bild 3.9: Die TTCN-3 Komponenten [GHR+03]

falls konkrete Testdaten festzulegen, mit denen diese Funktion während der Testfall-Ausführung ausgeführt wird. Was die Auswertung der Test-Ergebnisse betrifft, so ist in TTCN-3 ein „Verdikt-Mechanismus“ implementiert. Dieser definiert eine Menge von Konstanten, welche die Aussage über Erfolg oder Misserfolg des Testlaufs ermöglichen. Die Referenzwerte als solche werden oft in den Testfall „hineinkodiert“. Im Kontroll-Teil der Moduldefinition wird unter anderem die Reihenfolge der Ausführung einzelner Testfälle festgelegt. Damit die Testspezifikation ausführbar ist, wird in der TTCN-3 Spezifikation folgende Infrastruktur vorgeschrieben (Abbildung 3.10).

- TTCN-3 Executable (TE): Dieser Bestandteil ist für die Ausführung des kompilierten TTCN-3 Codes zuständig.
- Component Handler (CH): Ist für die Kommunikation zwischen den Testkomponenten zuständig. Er beinhaltet Operationen zum Starten, Beenden, zur Herstellung von Verbindungen sowie zur Abwicklung der Kommunikation zwischen diesen. Ebenso ist CH für die Verwaltung von Testergebnissen (Verdikten) zuständig.
- Testmanagement (TM): Steuert die Testausführung. In diesem Bestandteil sind Operationen zur Ausführung von Testfällen implementiert. Ebenfalls ist eine Logging-Funktionalität vorhanden.
- Coding/Decoding (CD): Kodiert und Dekodiert Variablen und ihre Werte. Diese werden zu SUT verschickt oder von SUT empfangen.

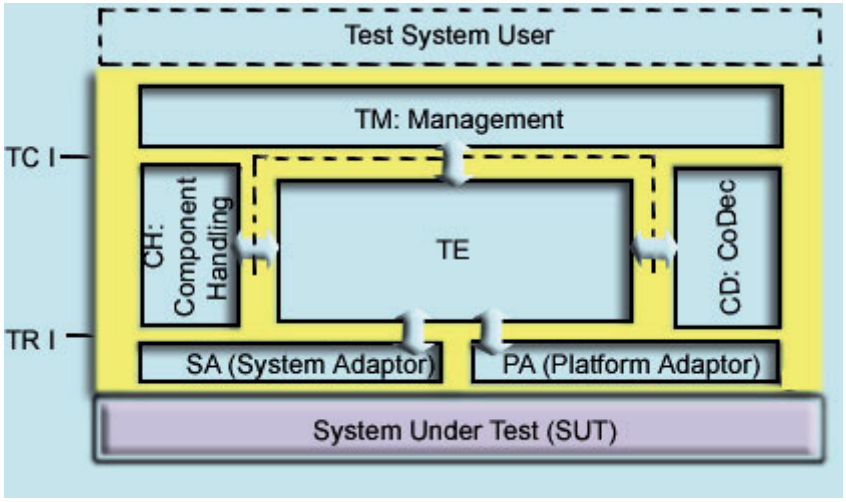


Bild 3.10: Die TTCN-3 Infrastruktur [GHR+03]

- System Adapter (SA): Implementiert die Kommunikation mit SUT. Meistens müssen solche Adapter für jedes neue SUT per Hand kodiert werden.
- Platform Adapter (PA): Implementiert einen Timer-Mechanismus. Timer sind plattformspezifische Funktionen, deshalb müssen sie separat von der bestehenden Testspezifikation entwickelt werden.

Was die Einsätze von TTCN-3 außerhalb der Telekommunikationsbranche betrifft, so ist in [Grabowski] ein Prototyp zum Blackbox-Testen eines Bankautomaten (Abbildung 3.11) vorgestellt. Dieses Beispiel wird dazu benutzt, um die TTCN-3 Notation zu erläutern.

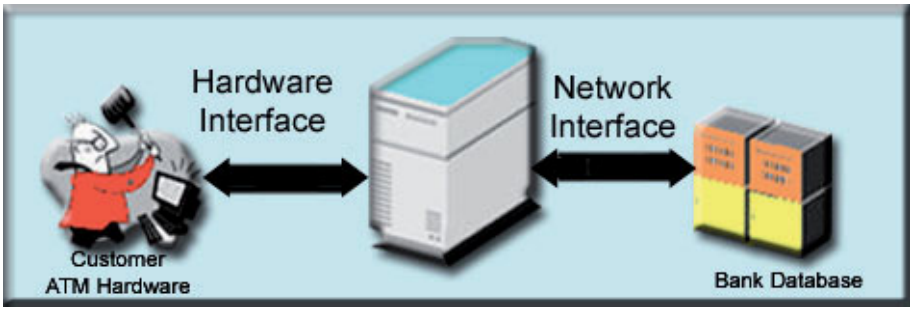


Bild 3.11: Bankautomat-Beispiel [GHR+03]

Ein Bankautomat ist ein Server, der zum einen mit der Bank-Datenbank verbunden ist und zum anderen mit der Automatenhardware, über welche ein Kunde mit dem Bankautomat-Server

verbunden wird, um Geld abzuheben. Die Datenbank wird dazu benötigt, um den Kontostand zu überprüfen. Die Verbindung zwischen den vorgestellten Bestandteilen ist durch entsprechende Interfaces realisiert. Ein TTCN-3 Testsystem für den Bankautomaten ist in der Abbildung 3.12 dargestellt. Die Bankautomat Hardware wird durch einen Emulator realisiert. Dieser ist als eine

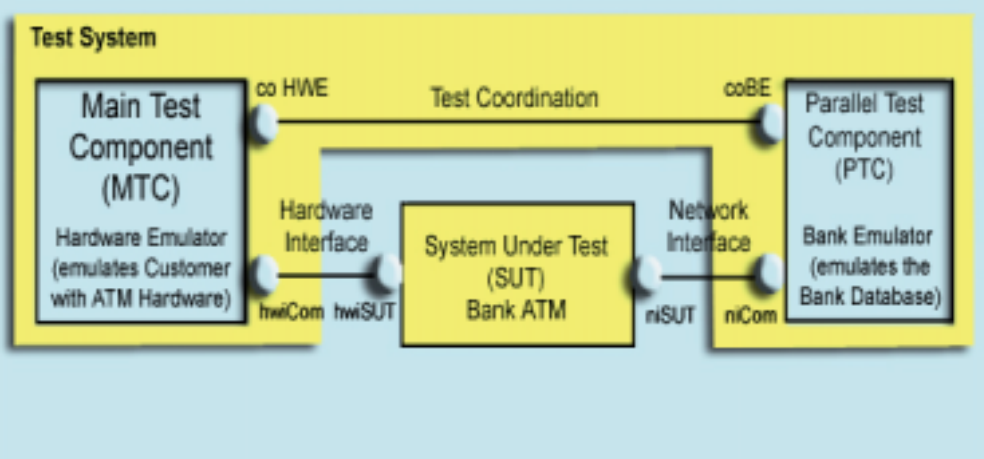


Bild 3.12: Bankautomat in TTCN-3 [GHR+03]

Testkomponente realisiert. Der Emulator kommuniziert über den Port „hwiCom“ mit dem SUT. Selbstverständlich müssen zur Kommunikation entsprechende Systemadaptoren geschrieben werden, welche die von Testfällen erzeugte Botschaften oder Funktionsaufrufe an das SUT weiterleiten. Die Datenbank wird ebenfalls durch einen Emulator realisiert, welcher als eine separate Testkomponente implementiert ist. Die Kommunikation zwischen der Datenbank und dem SUT findet über den Port „niCom“ statt. Zur Koordination der beiden Testkomponenten ist ebenfalls eine Verbindung zwischen diesen notwendig. Die Kommunikation geschieht in diesem Fall über zwei Ports „coHWE“ und „coBE“. Schließlich kann das SUT mit den beiden Testkomponenten jeweils über die Ports „hwiSUT“ und „niSUT“ kommunizieren. Der Ablauf eines typischen Testfalls ist in der Abbildung 3.13 dargestellt. Im ersten Schritt legt der Benutzer die Geldkarte in den Bankautomaten. Im nächsten Schritt werden seine Kenndaten an den Server (im Bild ATM) übertragen. Nach einer erfolgreichen Identifikation wird eine Bestätigung an den Hardwareemulator verschickt. Daraufhin spezifiziert der Benutzer die abzuhebende Geldsumme und es werden der Name der Operation und die abzuhebende Summe an den Server übermittelt. Im nächsten Schritt muss überprüft werden, ob die gewünschte Summe die auf dem Konto vorhandene nicht übersteigt. Dazu wird ruft der Server die Funktion „debitAccount“ auf. Es werden die Kontonummer sowie der Betrag übermittelt. Bei der erfolgreichen Überprüfung ruft

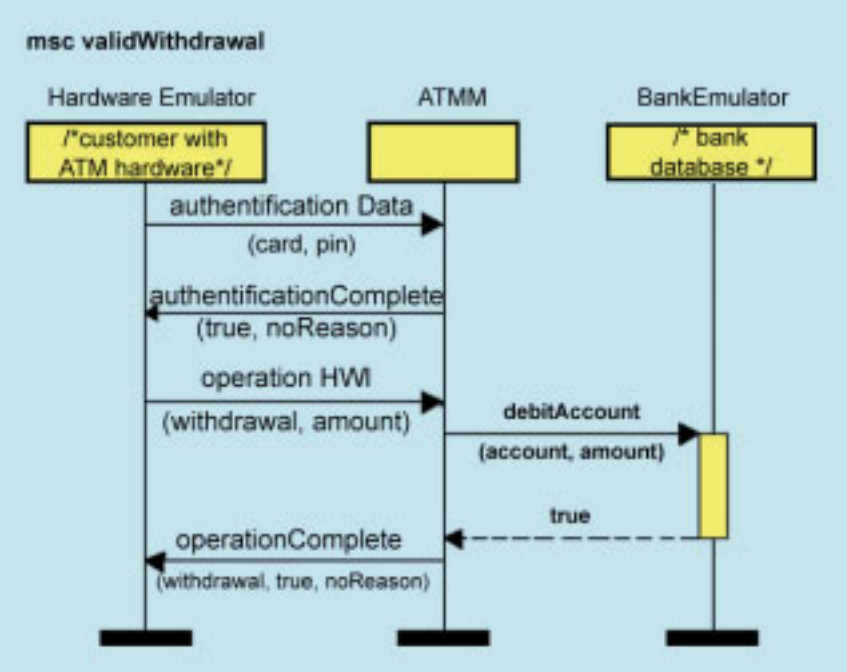


Bild 3.13: Typischer Testfall für den Bankautomaten [GHR+03]

der Server die Funktion „operationComplete“ auf, indem dem Hardwareemulator mitgeteilt wird, dass die Summe ausgezahlt werden kann.

Die restlichen untersuchten Quellen schildern meistens jeweils eine konkrete Anwendung von TTCN-3 bei einem vorhandenen „Testproblem“. So wird in [ORLS06] der Einsatz von TTCN-3 bei „Legacy Sotware“ aufgezeigt. Konkret geht es dort um einen Schiffsmotor, für welchen eine komplette auf TTCN-3 basierte Testarchitektur aufgebaut wird. Andere Gruppe von Publikationen [DDS03] befasst sich mit mit dem Testen von Web-Services. Dadurch, dass deren bestimmte Eigenschaften plattformunabhängig definiert sind, entsteht die Notwendigkeit, diese in eine ausführbare Testsprache zu transformieren. Als Zielsprache wird dabei eben TTCN-3 gewählt.

Bewertung textueller Methoden

Sowohl SCR als auch CSP sind im mathematischen Sinn korrekt, denn den beiden liegt ein mathematisches Modell zugrunde. Bei SCR-Modellen handelt es sich um Zustandsautomaten während bei CSP die Korrektheit durch eine Prozessalgebra garantiert wird. Bei TSL ist die Korrektheit nicht gegeben, da in der Literatur nicht gefunden wurde, wodurch diese gewährleistet werden soll. Denkbar wäre hier eine Grammatik oder ein Metamodell. TTCN-3 ist korrekt, da inzwischen deren genaue Spezifikation vorhanden ist. Die Validierung der in TTCN-3 erstellten

Spezifikationen kann durch den dedizierten Compiler geschehen. Ebenfalls sind alle Elemente der vorgestellten Methoden bis auf TSL vollständig definiert. Was die Einheitlichkeit betrifft, so ist bei den festgestellten Methoden anzumerken, dass es weiterer Einschränkungen bedarf, um diese zu garantieren.

Ein großes Problem bei den vorgestellten Ansätzen stellt die Wiederverwendbarkeit dar. Weder bei SCR noch bei CSP wird darauf eingegangen, wie ihre Unabhängigkeit vom Zielplattform beziehungsweise die Abbildbarkeit auf diese gewährleistet ist. Für CSP wurden in der Literatur lediglich mehrere werkzeugabhängige Notationen gefunden, welche im Bereich von Model Checking [For10] [Par96] eingesetzt werden. [BH97] beschreibt eine Methode für SCR, mit welcher eine Übertragung der SCR-Tabellen in Promela, eine Sprache des expliziten Model Checkers namens Spin [Hol04] stattfindet. Doch scheint die Übertragung manuell zu geschehen, was die Wiederverwendbarkeit deutlich erschwert. Dies liegt daran, dass man die einmal durchgeführte Übertragung in eine bestimmte plattformabhängige Zielnotation in Notationen anderer existierender Zeilplattformen scheuen wird, die beispielsweise in anderen Projekten oder den nachgelagerten Entwicklungsstufen verwendet werden. Denn oft muss diese Übertragung manuell geschehen, was einen hohen Aufwand mit sich bringt. Als einzige Methode bietet TSL eine vollständig automatisierte Umwandlung der Testspezifikationen in ein plattformspezifisches Format. Der Mechanismus basiert auf der Definition der Code-Templates für jede vorhandene Zielplattform, in welche dann die in TSL definierten Größen eingesetzt werden. TTCN-3 ist zweifelsohne zur Ausführung der Testspezifikationen bestens geeignet. Ihr Einsatz wäre zum Beispiel bei den Open Loop Testvorgängen (Kapitel 2) denkbar, um zahlreiche proprietäre Lösungen durch eine standardisierte zu ersetzen. Allerdings leidet die Wiederverwendbarkeit darunter, dass in solchen „Programmiersprachen-ähnlichen“ Notationen oft sehr schwierig ist Plattformunabhängigkeit zu erreichen. TTCN-3 erlaubt zwar die Definition von Testspezifikationen unabhängig von SUT. Jedoch wurde in der Praxis festgestellt, dass selbst bei demselben SUT die Testspezifikationen in unterschiedlichen Projekten unterschiedlich aussehen können. Dies liegt natürlich an dem Testfokus, also daran, welche SUT-Eigenschaften denn getestet werden. In TTCN-3 ist die Grenze zwischen dem projektabhängigen und und allgemeingültigen Teil der Testspezifikation bei der Untersuchung nicht deutlich geworden. Außerdem ist TTCN-3 an entsprechende Compiler gebunden. Dies führt dazu, dass, wenn eine Testspezifikation in einem anderen Projekt zumindestens teilweise wiederverwendet werden soll und dort keine TTCN-3 Plattform verwendet wird, proprietäre Lösungen entwickelt werden müssen, um diese Notation in diejenige umzuwandeln, die im Projekt gerade verwendet wird. Solche Transformationen geschehen oft leider immernoch manuell beziehungsweise es findet eine komplette Neudefinition in dem für das Projekt geeigneten Format statt. Dies ist mit einem hohen Zeitaufwand verbunden.

Was die Anpassbarkeit betrifft, so scheint diese zunächst bei SCR und CSP gegeben zu sein, da man ohne Weiteres neue Attribute definieren und in den dargestellten Notationen einsetzen kann. Doch bei genauerer Betrachtung ist folgendes festzustellen:

- Die Nachvollziehbarkeit darüber, welches Projekt innerhalb welcher Entwicklungsstufe welche Attribute definiert hat, ist nicht gegeben.
- Die Nachvollziehbarkeit darüber, welche Attribute allgemeingültig und welche projektspezifisch oder testverfahrenspezifisch sind, ist ebenfalls nicht gegeben.
- Die freie Definierbarkeit von Beschreibungselementen einer Methode führt zu Problemen bei der Wiederverwendbarkeit, da dadurch zum einen die Abbildbarkeit der Testspezifikation selbst innerhalb eines Projekts auf eine Zielplattform und zum anderen der Einsatz in den nachgelagerten Etwicklungsstufen erschwert werden.

Was TSL betrifft, so ist davon auszugehen, dass durch eine völlig automatisierte Abbildung auf eine bestimmte Zielplattform die freie Definierbarkeit von Attributen nicht gegeben ist. Allerdings konnten die Untersuchungen keine Aussage darüber liefern, wie der Abbildungsmechanismus angepasst werden kann, wenn neue Attribute hinzukommen. Vermutlich muss jedesmal das Code-Template angepasst oder gar neu erzeugt werden. Dies ist wiederum mit einem zeitlichen Aufwand verbunden.

TTCN-3 ermöglicht eine freie Definition von neuen Variablen. Dadurch, dass diese Notation ausführbar ist, müssen solche Attribute nicht transformiert werden. Allerdings ermöglichte die Publikationsanalyse keine Aussage darüber, welche Erfahrungen mit TTCN-3 gemacht wurden beim Einsatz in komplexen Softwareentwicklungsprozessen mit unterschiedlichsten Testmethodiken und mehreren beteiligten Institutionen. Daraus lässt sich nicht feststellen, wie hoch der Anpassbarkeitsgrad dieser Sprache ist. Ein großer Vorteil von TTCN-3 ist ihre Ausführbarkeit, die allerdings in komplexen Entwicklungsprozessen mit diversen Testmethoden und den mit ihnen verbundenen Notationen sehr schnell verloren gehen kann.

Bei anwenderbezogenen Kriterien besitzt SCR den höchsten Schwierigkeitsgrad, was die Erlernbarkeit betrifft. Dies liegt daran, dass die zu überprüfenden Systemeigenschaften (sowohl intern als auch bezogen auf die Interaktion des SUT mit der Umwelt) in Computational Tree Logic definiert werden. Diese besitzt zweifelsohne eine präzise mathematische Definition. Die Erfahrung zeigt aber, dass selbst für Informatiker das Erlernen und vor allem das korrekte Spezifizieren mit formalen Logiken sehr viel Zeit in Anspruch nimmt. Man bedenke aber, dass die Spezifikation von Menschen mit einem geringen Wissen in theoretischer Informatik erstellbar sein soll. Durch eine

geringe Anzahl an Beschreibungselementen sind CSP und TSL wohl am schnellsten erlernbar. So besitzt CSP lediglich zwei Primitive: Prozesse und Ereignisse sowie sechs algebraische Operatoren (Prefix, deterministische Auswahl, nicht deterministische Auswahl, Verschachtelung, Parallelität und schließlich das Ausblenden). Erleichternd kommt hinzu, dass CSP seit einigen Jahrzehnten besteht und dass dementsprechend genügend Wissen und Erfahrung da ist, um das Erlernen dieser Methode zu beschleunigen. Falls TSL eine Grammatik besitzen würde, dann wäre deren Erlernen von den vier vorgestellten Methoden am Schnellsten. TTCN-3 bietet drei Repräsentationsformen und somit kann jeder Benutzer selbst wählen, in welcher Form die Testspezifikationen beschrieben werden. Diese Flexibilität steigert zweifelsohne den Grad an Erlernbarkeit und somit auch an Einfachheit und Verständlichkeit dieser Sprache.

Die Anwendbarkeit spielt eine entscheidende Rolle bei der Methodenbewertung. CSP ist sehr gut dazu geeignet, um dynamische Aspekte sowohl des internen als auch des externen (interaktionsbezogenen) SUT-Verhaltens zu modellieren. Dahingegen findet die Modellierung statischer SUT-Aspekte weniger Beachtung. Bei den statischen Aspekten kann es sich beispielsweise um Umweltbedingungen handeln, unter denen die Interaktion des SUT mit der Umwelt stattfindet. So soll zum Beispiel bei der Beschreibung von Fahrszenen möglich sein, Wetter-, Fahrspur-, und Beleuchtungsbedingungen (aber auch solche Angaben wie Fahrspurverlauf) anzugeben, die für die gesamte Fahrszene gelten. Es handelt sich bei statischen Aspekten in gewisser Weise um Zustände. CSP operiert aber vorwiegend mit Prozessen und Ereignissen. SCR wäre aber durch die formale Repräsentation mit Hilfe von Zustandsautomaten sehr gut dazu geeignet, um sowohl statische als auch dynamische Aspekte eines reaktiven SUT zu modellieren. Dabei könnte man die bereits angeführten statischen SUT-Aspekte als Variablen kompositer Zustände darstellen. TSL weist einige Mängel in Bezug auf Anwendbarkeit im Bereich reaktiver Systeme auf. Die Analyse dieser Sprache konnte nicht herausstellen, ob dynamische Aspekte des SUT-Verhaltens modelliert werden können. Vor allem erlaubt die in [Lu94] vorgegebene Sprachdefinition keine Aussage darüber, wie die Interaktion des SUT mit der Umwelt zu modellieren ist und vor allem wie die zeitliche Synchronisation des SUT mit externen am Testvorgang beteiligten Objekten spezifiziert werden kann. TTCN-3 ist eine speziell für das Testen konzipierte Sprache, deshalb erlaubt sie die Modellierung aller zur Erstellung einer Testspezifikation notwendiger Bestandteile.

Das generelle Problem von allen textbasierten Testspezifikationsmethoden (TTCN-3 ausgeschlossen) sind die Verständlichkeit und die Anschaulichkeit, da keine graphische Möglichkeit zu deren Veranschaulichung besteht. Alle vier Methoden erlauben jedoch eine domänenbezogene Benennung von Elementen. Ohne weitere Mechanismen ist ebenfalls das oben beschriebene Sichtenkonzept nicht zu realisieren. Was TTCN-3 betrifft, so könnte unter Umständen die freie

Definierbarkeit von Attributen dazu führen, dass projektübergreifend Schwierigkeiten bei der Interpretation einer Testspezifikation entstehen.

Bei den anwendungsbezogenen Kriterien ist vor allem die Operationalisierbarkeit herauszuheben. Diese ist bei CSP und SCR durch die formale Definition grundsätzlich gegeben. Zur besseren Verwendbarkeit von formalen Testspezifikationen, die mit diesen Methoden definiert wurden, sind jedoch weitere Einschränkungen notwendig. Dies betrifft vor allem die strukturierte domänenbezogene Definition von allen möglichen Beschreibungselementen. Beispielsweise sollte eine kategorisierte Darstellung von allen möglichen Fahrmanövern in Form von Zuständen samt zugehöriger Attribute bei SCR möglich sein. Somit würde eine Art Wörterbuch entstehen, das nicht nur die Anschaulichkeit, sondern auch die Operationalisierbarkeit der Testspezifikationen erhöht. Denn ein Abbildungsmechanismus würde sich eines solchen Wörterbuches bedienen, um aus plattformunabhängiger Testspezifikationsbeschreibung eine plattformabhängige zu erzeugen. Das gerade beschriebene Konzept von einem Wörterbuch würde somit ebenfalls die freie Definition von Beschreibungselementen effektiv einschränken und somit auch die Wiederverwendbarkeit einer Test-Spezifikationsmethode wesentlich steigern. Das Konzept eines Wörterbuchs wäre ebenfalls beim Aspekt der Erweiterbarkeit der vorgestellten Methoden sehr sinnvoll. Denn ein solches Wörterbuch könnte man so strukturieren, dass dort das Hinzufügen von neuen Elementen projekt- und testartspezifisch möglich wäre. Somit wären die Auswirkungen einer Erweiterung auf andere Beschreibungselemente einer ausgewählten Methode äußerst gering. Leider ist bei allen hier analysierten Methoden kein solches Konzept zur domänenbezogenen Definition von Beschreibungselementen vorhanden. Am ehesten wäre ein solches Konzept allerdings bei TSL durch die Definition einer entsprechenden Grammatik vorstellbar. Zum anderen muss die Zeit berücksichtigt werden, die zur Erstellung einer solchen Beschreibung notwendig ist. Gänzlich ist der manuelle Aufwand nicht zu vermeiden. Was TTCN-3 betrifft, so ist ihre Operationalisierbarkeit insofern gewährleistet, als sie ausführbar ist. Unklar ist, inwieweit es möglich ist, TTCN-3 in andere Notationen möglichst automatisiert zu transformieren. Dieser Aspekt spielt, wie bereits erwähnt, eine wichtige Rolle im Rahmen dieser Arbeit. Vermutlich wäre eine solche Transformation erst durch mehrere Zwischenschritte möglich. TTCN-3 ist in Bezug auf ein bestimmtes Projekt sehr gut erweiterbar. Aus der globalen Perspektive bringt jedoch eine solche Erweiterbarkeit Probleme bei der Wiederverwendbarkeit (siehe Diskussion oben).

3.3.1.2 Graphische Methoden

Bereits in den Achtzigern entstanden erste graphische Formalismen zur Beschreibung des Systemverhaltens [Harel84]. Aus der Telekommunikationsbranche kam ebenfalls Ende der Achtziger

das Standard Message Sequence Charts [ITU04]. Im Jahre 1997 erhielt die Modell-getriebene Softwareentwicklung einen neuen Schub, indem von der Object Management Group (OMG) ein Standard namens Unified Modelling Language (UML) [Obj09] geschaffen wurde. In diesem wurden viele bereits bestehende graphische Systemrepräsentationsmethoden vereinheitlicht. Es kamen aber auch neue hinzu. Aus Sicht des Testens ist UML insofern relevant, als mit ihrer Entstehung eine ganze Forschungsrichtung namens Modell-getriebenes Testen (engl. MDT) [Bak05] entstanden ist. In diesem Abschnitt werden zwei konkrete Ansätze zum Modell basierten Testen betrachtet und bewertet. Es sei an dieser Stelle bemerkt, dass nur Ansätze betrachtet werden, die für die vorliegende Arbeit von Bedeutung sind. Das gesamte Gebiet des MDT kann hiermit nicht abgedeckt werden. Schließlich wurde im Jahr 2005 von OMG ein Profil zur Definition von Testspezifikationen konzipiert und veröffentlicht, der sich UML Testing Profile (UTP) [Obj10c] nennt. Am Ende dieses Abschnitts wird auf dieses ebenfalls eingegangen.

Testfallgenerierung mit Hilfe von Model Driven Architecture

Model Driven Architecture [MSUW04] ist eine Vorgehensweise zum Entwurf komplexer Software-Systeme, die im Jahr 2001 von OMG vorgeschlagen wurde. Das zu realisierende Softwaresystem wird in MDA auf verschiedenen Abstraktionsstufen mit Hilfe von Modellen dargestellt. Diese Modelle werden laut OMG mittels passender in UML beinhalteter Diagrammtypen dargestellt. Nicht von Diagrammtypen wird die textuelle, meist natursprachliche Systembeschreibung erfasst. Diese nennt sich CIM (engl. Computation Independent Model). CIM bildet den Ausgangspunkt von MDA. Basierend auf CIM werden manuell so genannte PIM („Platform Independent Model“) erstellt. Dieses Modell beschreibt die Systemfunktionalität auf abstrakte Art und Weise, wobei konkrete zielplattformabhängige Implementierungsdetails hier noch nicht berücksichtigt werden. Im nächsten Schritt wird das so genannte „Platform Specific Model“ (PSM) spezifiziert. Dieses besitzt ein höheres Spezialisierungsgrad im Vergleich zu PIM und ist oft der unmittelbare Ausgangspunkt der Code-Generierung. Ein wichtiger Aspekt von MDA ist die Transformation von PIM zu PSM. Dazu muss das Ausgangsmodell ,also PIM, oft manuell mit zusätzlichen Informationen angereichert werden. Es müssen ebenfalls Transformationsregeln vorhanden sein, welche von dedizierten Transformationssprachen [Obj10b] verarbeitet werden, um ein PIM in das PSM zu transformieren. Wichtig anzumerken ist, dass es mehrere PIMs und PSMs und dementsprechend mehrere Transformationsschritte zwischen diesen geben kann.

In [JSW07] wird ein auf MDA basierter Ansatz vorgestellt, der es erlaubt, aus Unit-Test Spezifikationen [Lig09] mittels Sequenzdiagramme über zwei Transformationsschritte ausführbare Testskripte zu erzeugen. Eine Testfallbeschreibung wird als ein dem Metamodell (siehe

Abbildung 3.14) konformes Sequenzdiagrammmodell erzeugt (engl. SMC Model). Anschließend wird ein Modell-zu-Modell Transformationstool (Tefkat) gestartet, welches die Transformation in das X-Unit Modell vornimmt. Unter X-Unit werden übliche Unit-Testing Verfahren, wie JUnit oder SUnit verstanden. Zur Transformation benötigt Tefkat außer Sequenzdiagramm-Modell und Metamodell ebenfalls die Spezifikation von den durchzuführenden Tranformationen sowie das Metamodell der Zielnotation (In diesem Fall xUnit). Als Ergebnis der ersten Transformation

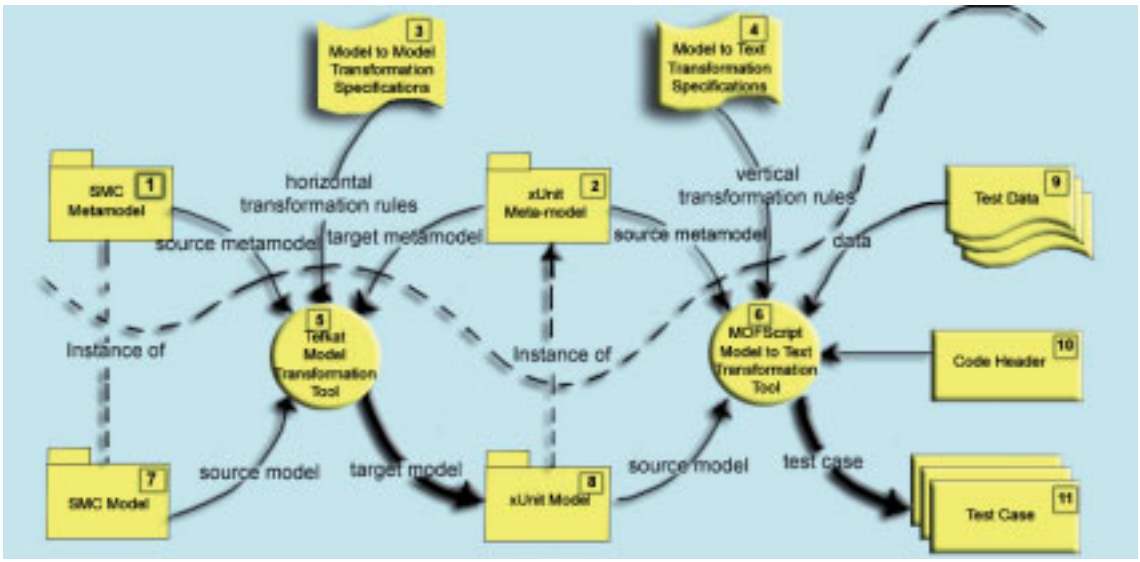


Bild 3.14: MDA-basiertes Unit-Testing [JSW07]

wird ein dem xUnit Metamodell konformes Modell betrachtet. Im nächsten Schritt findet die eigentliche Testfall-Erzeugung, indem die so genannte Modell-zu-Text Transformation gestartet wird. Dazu wird eine dedizierte Transformationssprache namens MOFScript [Ecl10a] verwendet. Zur Transformation wird ebenfalls eine Transformationsspezifikation benötigt, in welcher letztendlich die Transformationsregeln stehen. Außerdem, da es sich ja um eine Modell-zu-Text Transformation handelt, müssen konkrete Parameterbelegungen spezifizeirt werden, mit welchen der entsprechende Unit-Test durchgeführt wird (In der Abbildung 3.14 Artefakt 9). Im Code Header werden solche Informationen angegeben, wie die zum Ablauf notwendigen Pakete oder importierte Klassen. Als Ergebnis der Transformation wird eine Menge ausführbarer Testfälle für den Unit-Testing-Verfahren produziert.

UML Testing Profile

Ein UML-Profil bietet die Möglichkeit, die klassische UML in Bezug auf Anwendung in einer bestimmten Domäne einzuschränken. Das „UML Testing Profile“ (UTP) liefert eine Menge von Konzepten, welche es erlauben, Testspezifikationen kompakt zu beschreiben. Diese Konzepte definieren eine Modellierungssprache zur Visualisierung, Spezifikation, Analyse und Dokumentation der Artefakte des zu testenden Systems [Bak05]. Nach [Bak05] besteht UTP aus folgenden grundlegenden Konzepten:

- Testarchitektur: Ist eine Menge von Konzepten, mit welchen strukturelle Aspekte der Testsituation spezifiziert werden können.
- Testkontext: Erlaubt die Gruppierung der Testfälle, Beschreibung der Testkonfiguration sowie die Definition der Ausführungsreihenfolge der Testfälle.
- Testkonfiguration: Ist die komposite Struktur des Testkontexts, welche die Kommunikation zwischen den Testkomponenten und dem SUT definiert.
- Das zu testende System (SUT): Typischerweise können innerhalb der Testspezifikation mehrere Objekte als SUT indentifiziert werden.
- Testkomponenten: Sind Objekte innerhalb des Testsystems, die mit dem SUT oder anderen Komponenten kommunizieren können, um das Testverhalten oder den Testablauf zu gewährleisten.

Zur Definition von Testverhalten werden folgende Aspekte herangezogen:

- Testziel: Definition von Referenzwerten.
- Testfall: [Bak05] definiert den Testfall als Operation des Testkontexts, welche festlegt, wie die Menge kooperierender Komponenten mit der SUT interagiert, um das Testziel zu erreichen.
- Verdikt: Ist eine vordefinierte Aufzählung, die alle möglichen Testergebnisse spezifiziert (z.B. „pass“, „fail“, „inconclusive“ und „error“).
- Timer: Werden verwendet um den Testablauf zu manipulieren und zu kontrollieren und um die Beendigung einzelner Testfälle zu gewährleisten.
- Arbiter: Stellt Funktionalität zur Auswertung der Testergebnisse.
- Scheduler: Überwacht die Testausführung.

Der Zusammenhang zwischen den einzelnen Konzepten ist in der Abbildung 3.15 gezeigt. Für eine detaillierte Darstellung sei auf [Obj10c] verwiesen.

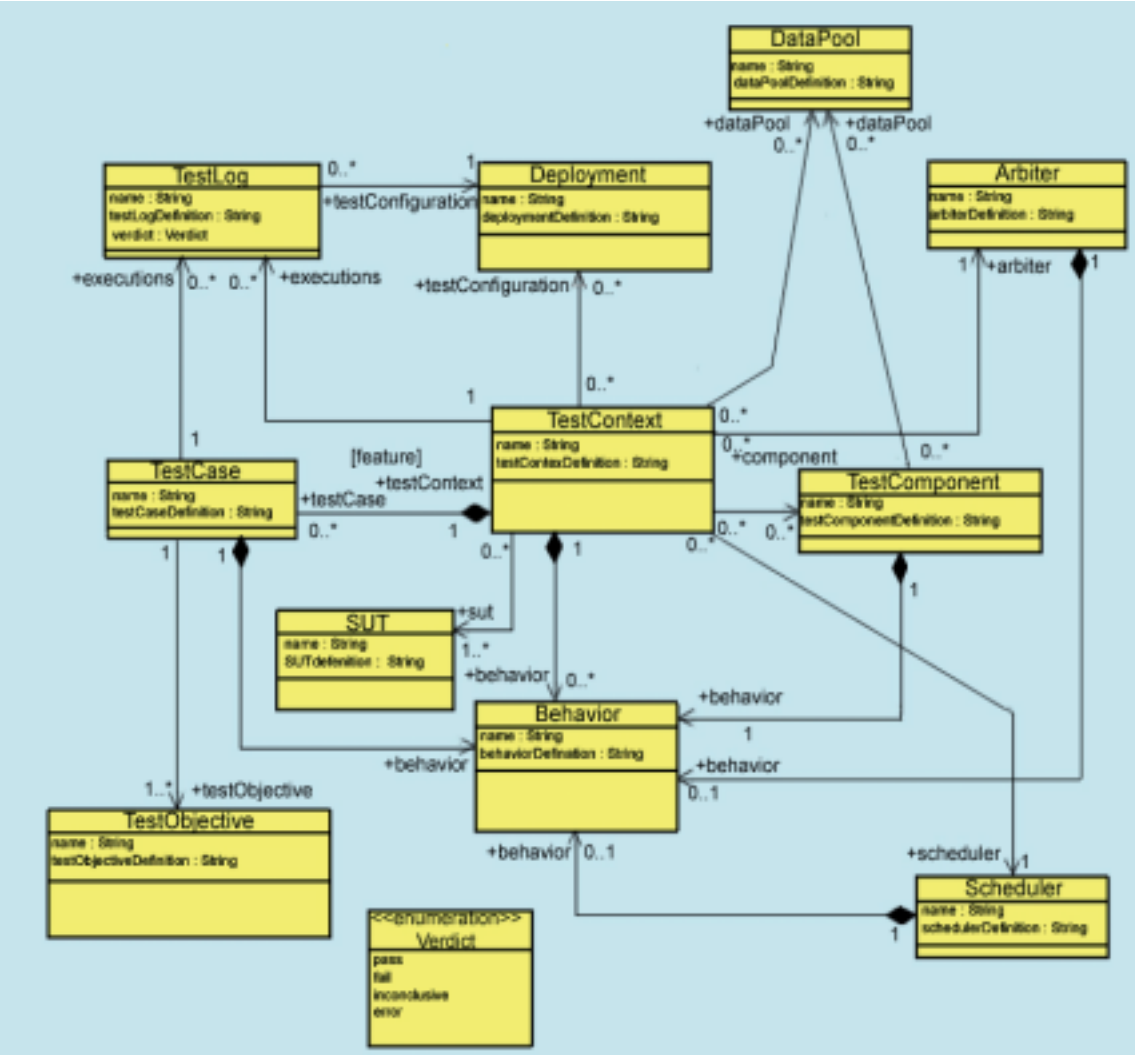


Bild 3.15: UTP: Testarchitektur [Obj10c]

Modell-getriebenes Testen mit UML Testing Profile

In [Dai04] wird folgende Methodik für das Modell getriebene Testen mit UML Testing Profile vorgeschlagen (Abb. 3.16). Basierend auf dem Grundgedanken der MDA wird als erstes ein

UML-Modell erzeugt, welches das Systemverhalten beschreibt. Zur Beschreibung des Systemverhaltens kann einer der von UML Diagrammtypen herangezogen werden. Oftmals werden Sequenzdiagramme, Zustandsautomate, Aktivitätsdiagramme oder Statecharts verwendet. Im nächsten Schritt wird über mehrere Transformationsschritte aus dem PIM des SUT der Code erzeugt. Gleichzeitig ist es möglich, schon zur Designzeit aus dem PIM mittels Modell-zu-Modell Transformation (mit Hilfe von Query View Transformation Language (QVT)) entsprechendes Testmodell (PIT) in UTP zu erzeugen. Mit Hilfe dieses Testmodells kann man bereits in frühen

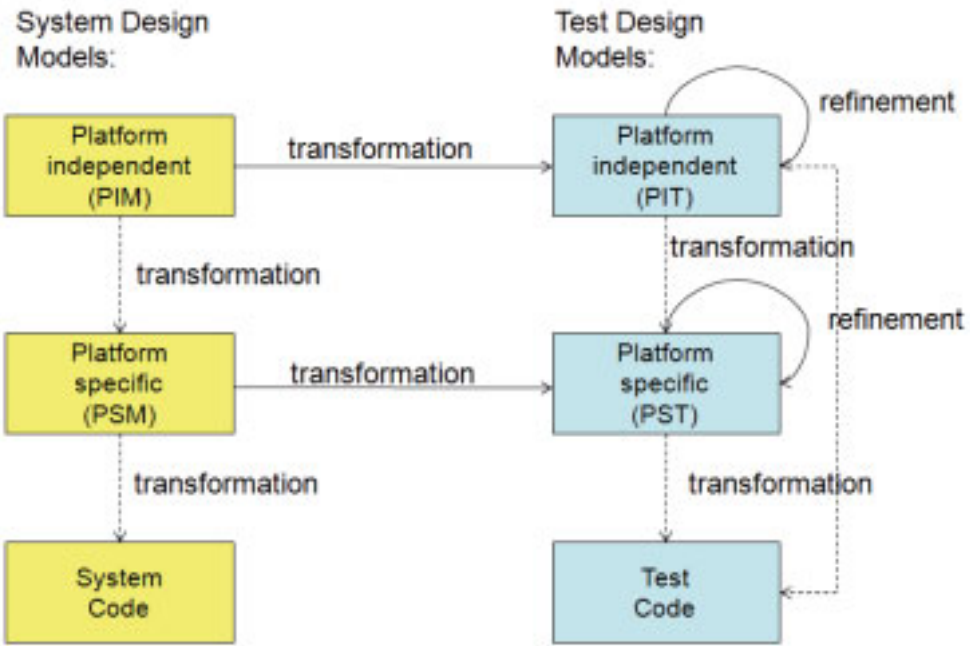


Bild 3.16: Methodik zum Testen mit UTP [Dai04]

Entwicklungsphasen das PIM des zu entwicklenden Systems testen, was eindeutig ein großer Vorteil dieser Methodik ist. Selbstverständlich kann nach MDA-Prinzip vom Test-Modell ebenfalls über mehrere Transformationsschritte ein ausführbarer Testcode abgeleitet werden, mit dem das SUT auf Code-Ebene funktional getestet werden kann. Es ist wichtig anzumerken, dass bei allen Transformationsvorgängen oftmals manuelle Ergänzungen notwendig und oft auch unvermeidlich sind. So muss das Testmodell vor der Transformation in den ausführbaren Code um zielplattformspezifische Informationen ergänzt werden, welche beispielsweise die Kontrolle der Testausführung betreffen. In der Abbildung 3.17 ist nochmals der detaillierte Transformationsvorgang von SUT PIM zum Test Modell PIM gezeigt.

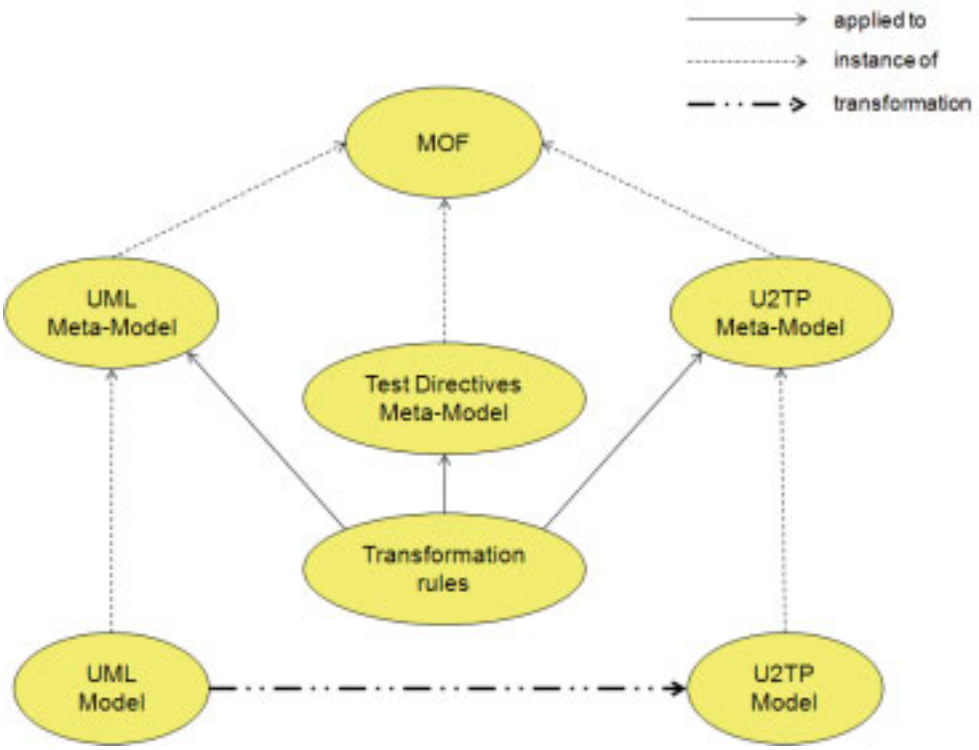


Bild 3.17: Transformation von SUT PIM-Modell zum Testmodell in UTP [Dai04]

Testspezifikationen mittels domänenspezifischer Sprachen

Die domänenspezifischen Sprachen sind ein unabdingbarer Teil der modellgetriebenen Softwareentwicklung (MDSD). „MDSD ist keine fertig einsetzbare Lösung, sondern die Beschreibung eines Vorgehensmodells, d.h. es werden z.B. keine konkreten Methoden für die Modellierung benannt, sondern lediglich festgestellt, dass eine Methode zur domänenspezifischen Modellierung eingesetzt werden muss. In jedem Fall muss eine Anpassung der MDSD an die lokalen Gegebenheiten, wie z.B. das behandelte Fachgebiet oder die genutzte Software-Plattform, erfolgen. Diese Anpassung darf aber nicht im Sinne eines Customizing oder gar Tailoring missverstanden werden, da die notwendigen Anpassungen deutlich über den üblichen Aufwand dieser Techniken hinausgehen, indem beispielsweise eine komplette Modellierungssprache zu entwerfen ist oder eigene Generatoren zu entwickeln sind“ [Mü07]. Eine domänenspezifische Sprache (engl. Domain Specific Language DSL) ist nach [DKV00] eine Programmiersprache oder eine ausführbare Spezfikationssprache, welche durch geeignete Abstraktionen und Konstrukte, genügend Ausdruckskraft bietet, um eine spezifische Domäne zu beschreiben. Die erwähnten Konstrukte und Abstraktionen sind meistens auf diese Domäne auch beschränkt. Die DSLs bieten folgende Vorteile [Cza04]:

- „Domänenspezifische Abstraktionen: DSL stellen vordefinierte Abstraktionen zur Verfügung, welche die Konzepte der Anwendungsdomäne direkt darstellen.

- Domänenspezifische Syntax: DSL bietet eine auf die Domäne ausgerichtete Syntax und sie vermeidet damit Unklarheiten und Mehrdeutigkeiten, die sich beim Einsatz generischer Sprachen wie zum Beispiel UML ergeben können.

- Domänenspezifische Fehlerüberprüfung: DSLs erlauben die Definition eines auf die Domäne zugeschnittenen Validierungsmechanismus, der in einer dem Fachexperten verständlichen Notation realisiert ist.

- Domänenspezifische Optimierungen: Der Einsatz von DSLs erlaubt Entwicklung spezifischer Codegeneratoren, welche den produzierten Code im Hinblick auf den Einsatz in der Domäne entsprechend optimieren. Bei generischen Codegeneratoren gestaltet sich eine solche Optimierung als sehr schwierig.

- Domänenspezifische Werkzeug-Unterstützung: Die Spezialisierung von DSLs bietet Möglichkeiten zur Entwicklung hochangepasster Werkzeuge, mit denen das Arbeiten mit diesen noch produktiver als mit generischen Sprachen wird.“

Die restlichen Bestandteile der MDSD sind (Meta)Modelle, Plattformen sowie ein Entwicklungsprozess. Entwicklungprozess wird im Abschnitt 3.3.2 behandelt während (Meta)Modelle und Plattformen im Kapitel 4 eingeführt und näher betrachtet werden.

Was die Definition der Testspezifikationen mit Hilfe von domänenspezifischen Sprachen betrifft, so wird dort meistens versucht, die Verhaltensmodelle von SUT mittels solcher DSLs zu beschreiben. Einige dieser Ansätze versuchen bestehende graphische Formalismen zur Systemspezifikation, wie beispielsweise Message Sequence Charts oder Statecharts durch Definition geeigneter Metamodelle an den Ansatz in der Domäne anzupassen. In [Ben08] wird beispielsweise folgende Methodik angewandt (Abbildung 3.18). Das SUT-Verhalten wird mittels Statecharts beschrieben. Diese werden mittels eines domänenspezifischen Statechartmodels beschrieben. Um die aus dem Verhaltensmodell abgeleiteten Testfälle repräsentieren zu können, wird ein Metamodell definiert, das eng mit dem Verhaltensmodell verwoben ist. Das alles wird vom Autor als Testmodell bezeichnet (Siehe Abbildung 3.18). In der Terminologie dieser Arbeit handelt es sich jedoch um Verwebung von Verhaltens-und Interaktionsmodellen. In der Abbildung 3.19 ist im linken Teil der Abbildung ein Metamodell für Statecharts dargestellt. Bei genauer Betrachtung

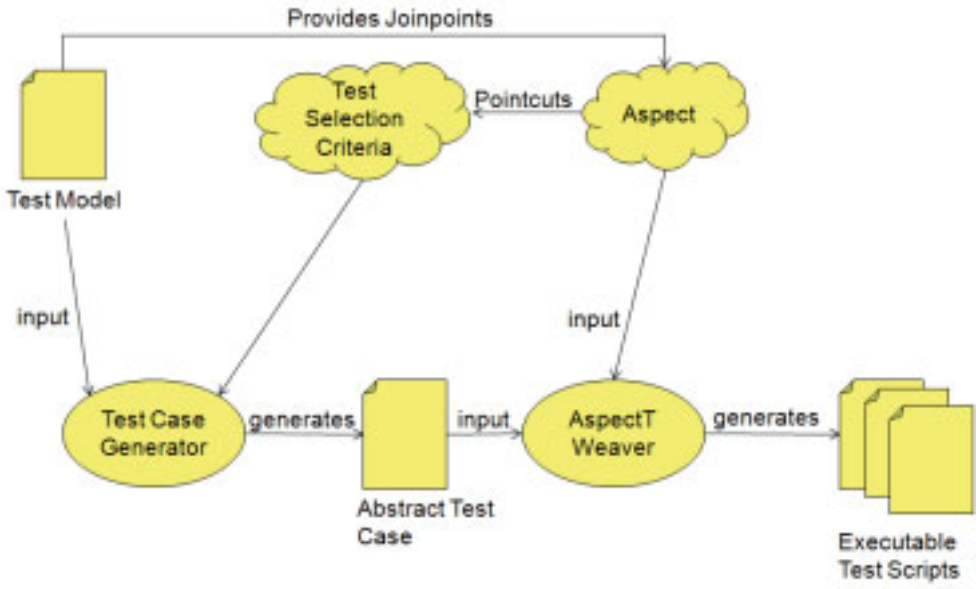


Bild 3.18: Aspektorientierte Testfall-Generierung [Ben08]

merkt man, dass die Definition von Guards und Actions fehlt. Dies liegt daran, dass in der Anwendungsdomäne (Das Testen von GUIs im automobilen Infotainment-Bereich) das SUT-Verhalten mittels Zustände und Ereignisse beschrieben wird. Diese sind notwendige und ausreichende Modellierungselemente. Das, was in der Abbildung 3.19 als „Test Case Metamodel“ dargestellt ist, stellt nichts anderes als die Interaktionsbeschreibung des SUT. Die Instanz dieses Metamodells ist ein bestimmter Pfad im Statechart mit konkreten Ereignisangaben, welche in diesem Fall die Zustandsübergänge auslösen. Das Paar Ereignis, Zustand wird als Teststep bezeichnet. In der Realität handelt es sich aber um eine bestimmte Interaktion des Systembenutzers mit dem SUT. So kann ein Ereignis „Taste „Anruf annehmen“ drücken“ dazu führen, dass das SUT in den Zustand übergeht „CD-Player unterbrochen“. Für jeden Teststep kann noch ein Testorakel definiert werden, der den tatsächlichen Systemzustand zur Laufzeit überprüft.

Da der hier vorgestellte Ansatz aspektorientiert ist, kann nach der Generierung eines abstrakten Testfalls aus dem Verhaltensmodell dessen Beschreibung um konkrete testrelevane Aspekte ergänzt werden. In der Infotainmentdomäne kommt es oft vor, dass man mit einem Testfall mehrere Systemeigenschaften abprüfen kann. Beispielsweise kann man mit einem und demselben Fall sowohl das zeitliche Verhalten einer GUI abprüfen als auch deren graphische Repräsentation. Je nachdem welche Systemeigenschaften man testen möchte, wird die abstrakte Testfallbeschreibung im nächsten Schritt um entsprechende Aspekte ergänzt, welche durch einen Weaver (AspectT Weaver) in Teile des ausführbaren Testskripts transformiert werden. Die Aspekte können bei-

spielsweise spezifizieren, ob die SUT-Umgebung simuliert oder ob während der Testausführung bewusst Fehler eingestreut werden (engl. Fault Injection), indem zum Beispiel die Reihenfolge der Ereignisse vertauscht wird, um zu sehen, wie das SUT darauf reagiert.

Bild 3.19: Testmodell: Verwebung von Verhaltens- und Interaktionsbeschreibung [Ben08]

Bewertung graphischer Ansätze

Alle drei vorgestellten graphischen Methoden sind korrekt. Sowohl der MDA-basierte Ansatz als auch UTP basieren auf einem von OMG vorgegebenen Metamodell. Anhand dieses Metamodells kann man eindeutig feststellen, welche Bestandteile einer Testspezifikation zulässig sind und welche nicht. Die DSL-basierten Ansatze, welche sich an MDSD anlehnen, sind ebenfalls metamodellbasiert. Dies erlaubt, domänenspezifische Werkzeuge zur Validierung der erstellten Testspezifikationen zu entwickeln. Diese überprüfen wiederum die Testspezifikationen auf Korrektheit. Ebenfalls ist Vollständigkeit bei allen drei Ansätzen durch das Vorhandensein eines entsprechenden Metamodells gegeben. Allerdings lässt sich beobachten, dass bei der Entwicklung eines DSLs sowohl sehr gute Domänenkenntnisse als auch diese in der Metamodellierung vorhanden sein müssen. Sonst kann die Definition sehr schnell unvollständig werden und die in der Domäne vorhandenen Sachverhalte inkorrekt abbilden.

Da UML eine generische Sprache ist, welche sich nicht auf den Einsatz in einer bestimmten Domäne beschränkt, ist die Einheitlichkeit im Sinne der Redundanzfreiheit bei den UML-basierten Ansätzen nicht immer vorhanden. Verwendet man beispielsweise zur Beschreibung von

Fahrszenen das von OMG vorgegebene Statechart-Metamodell, so kann man den Sachverhalt „Regenbeginn“ sowohl als Ereignis als auch als Bedingung modellieren. Dasselbe gilt für den Sachverhalt „Messtechnik einschalten“ oder „Spurwegfall“. Die DSL-basierten Ansätze liegen bei diesem Kriterium deutlich im Vorteil, da durch geeignete Metamodell-, beziehungsweise Modelldefinition sich spezifizieren oder gar festschreiben lässt, welche Sachverhalte mit welchen Modellierungselementen darzustellen sind. Möglicherweise können einzelne nicht benötigte Beschreibungselemente im Metamodell gar ausgelassen werden (siehe aspektorientierte Testfall-Instanziierung).

Grundsätzlich unterstützen die UML-basierten Ansätze die Wiederverwendbarkeit, indem von OMG dedizierte Transformationssprachen wie beispielsweise QVT oder MOFM2T [Obj08b] vorgegeben werden. Somit ist eine stufenweise Transformation von UML-Modellen in den ausführbaren Code im Sinne von MDA möglich. Bei UTP finden sich in der Literatur [Bak05] Vorschläge zu dessen Abbildung auf JUnit und TTCN-3, es wird allerdings nicht erwähnt, auf welche Art und Weise die Transformation bewerkstelligt werden soll. Die untersuchten Quellen beinhalten lediglich Gegenüberstellungen von Beschreibungselementen der jeweiligen Methoden und sie geben einige Beispiele, wie eine mit UTP beschriebene Testspezifikation in JUnit oder in TTCN-3 aussehen würde. Der Aspekt der Automatisierung des Abbildungsvorgangs kommt auch nicht deutlich zum Ausdruck. Bei DSL-basierten Ansätzen existiert hingegen eine Reihe von ausgereiften und in der Praxis erprobten Transformationssprachen, welche sowohl automatisierte Modell-zu-Modell (z.B. XTend [Ecl10b]) als auch Modell-zu-Text [Ecl10b] Transformationen erlauben.

Alle hier vorgestellten Ansätze unterstützen bis zu einem bestimmten Grad die Anpassbarkeit. UML besitzt den Profil-Mechanismus, mit welchem man die Metamodelle graphischer Formalismen an die Gegebenheiten einer bestimmten Domäne anpassen kann. Die DSLs besitzen in dieser Hinsicht den höchsten Grad an Flexibilität, da man nicht nur projektspezifische Attribute für die Beschreibungsformalismen definieren kann, sondern auch die Beschreibungselemente dieser Methoden selbst. Bei der UML ist es hingegen so, dass man lediglich bestehende von OMG vorgegebene Beschreibungselemente einer Methode durch Angabe domänenspezifischer Attribute an die Domäne anpassen kann. Im Kontext dieser Arbeit würde der Einsatz von UML also bedeuten, dass man ein sehr umfangreiches Profil definieren müsste, der Gegebenheiten eines jeden Subprojekts im gesamten Entwicklungsprozess berücksichtigt. Dies ist zwar theoretisch möglich, doch würden sich aus Sicht eines einzelnen Subprojektes sehr viele überflüssige Attribute ergeben, die die Gegebenheiten anderer Projekte berücksichtigen. Außerdem müssten beim Hinzufügen von neuen Attributen zum Profil, alle am Prozess beteiligten Subprojekte ei-

ne und dieselbe Profilversion verwenden, damit die Konsistenz bewahrt wird. Dies ergibt sich angesichts des in Kapitel 2 beschriebenen Entwicklungsprozesses als sehr aufwändig. Sinnvoll wäre hingegen die Möglichkeit, die Gegebenheiten eines jeden Subprojekts durch eine Menge generischer Attribute zu beschreiben, die von allen Prozessbeteiligten verwendet werden. Die projektspezifischen Attribute sollen durch einen Erweiterungsmechanismus definierbar sein und zwar so, dass sie und nur sie von einem bestimmten Subprojekt verwendet werden können. Dies ist durch die Definition eines geeigneten Metamodells mittels DSL möglich.

Generell geht man davon aus, dass graphische Methoden einfacher zu erlernen sind als textuelle. Dies hängt jedoch stark vom Benutzer ab, vor allem davon ob dieser Kenntnisse in Programmierung hat. Deshalb kann man die Einfachheit allein anhand dieser Eigenschaft nicht beurteilen. Generell lässt sich bemerken, dass bei den bewerteten Ansätzen die DSLs im Vorteil liegen. Denn oft gibt es bei einzelnen Diagrammtypen Kosntrukte, welche für die Anwendung in der Domäne irrelevant sind, das Erlernen der Methode aber erschweren könnten. So gibt es beispielsweise ab UML 2.0 bei Sequenzdiagrammen die Möglichkeit nebenläufige Ausführung zu modellieren. Es stellt sich die Frage, warum dieses Konstrukt im Diagrammtyp aufgenommen wurde, wenn es in UML bereits andere Diagrammtypen gibt, welche auf die Modellierung der Nebenläufigkeit ausgerichtet sind (Statecharts oder Aktivitätsdiagramme). Bei den DSLs ist man insoweit flexibler im Vergleich zu UML, als man die Anzahl der Modellierungskonstrukte auf die minimal für die gewählte Domäne notwendige beschränken kann. Somit wird der Grad der Einfachheit gesteigert.

Sowohl UML- als auch DSL-basierte Ansätze besitzen einen hohen Grad an Anwendbarkeit. UML bietet sechs Diagramme zur Strukturmodellierung und sieben Diagramme zur Verhaltensmodellierung, sodass beispielsweise zur Definition von Verhaltens- oder Interaktionsbeschreibung eine breite Palette an Modellierungsmitteln zur Verfügung steht. Was die DSLs betrifft, so erlauben sie durch Definition eigener Metamodelle ebenfalls einen hohen Grad an Anwendbarkeit. Oftmals verwendet man sogar bestehende graphische Formalismen, die durch bestimmte Einschränkungen oder Erweiterungen an die Domäne angepasst sind.

Alle untersuchten graphischen Ansätze besitzen einen hohen Grad an Verständlichkeit. UML erlaubt die freie Benennung von Modellierungselementen einzelner Diagramme durch den Benutzer. Bei den DSL-basierten Ansätzen besteht die Möglichkeit, die Namen der Modellierungselmente im Modell festzulegen („Wörterbuch“-Konzept). Dieses Konzept bietet Vorteile im Vergleich zu UML, da man zum Definitionszeitpunkt der Benennungen für Beschreibungselemente die einzelnen Begriffe mit allen am Subprojekt beteiligten Abteilungen und Personen abstimmen kann. Somit steigt der Verständlichkeitsgrad der Methode. Bei UML hingegen besteht die Gefahr,

dass durch freie Definierbarkeit von Benennungen Missverständnisse zwischen beispielsweise Funktionsentwickler und Tester entstehen, weil sie verschiedene Begriffe für dasselbe Konzept verwenden.

Wie bereits erwähnt, kann man die Anschaulichkeit nur anhand der Möglichkeit zur graphischen Repräsentation sehr schwer beurteilen. Versucht man diese anhand der Möglichkeit zur Sichtenbildung (siehe Definition des Kriteriums „Anschaulichkeit“) zu beurteilen, so liegen die DSL-basierten Ansätze eindeutig im Vorteil, da dort solche „Sichten“ leicht zu definieren sind. Man könnte zwar auch bei UML pro Subprojekt ein eigenes Profil definieren, darunter würde aber die Wiederverwenbarkeit und somit auch eine der Hauptanforderungen in dieser Arbeit leiden.

Was die Ausdrucksmächtigkeit betrifft, so lässt sich beobachten, dass UML (UTP) und umso mehr die DSL-basierten Ansätze den adäquaten Grad an Ausdrucksmächtigkeit besitzen, um Testspezifikationen zu definieren. Bei UML ergibt sich ebenfalls aus den obigen Ausführungen, dass durch ihre Generizität die Ausdrucksmächtigkeit für eine bestimmte Domäne sogar zu hoch sein kann, was zu Problemen bei der Beschreibung führen kann (Siehe Beispiel über die Modellierung mit Ereignissen/Bedingungen).

Wie bereits erwähnt, liegt UML ein Metamodell zugrunde. Somit existiert eine formale Basis, mit welcher weitere Aktivitäten wie Transformation in bestimmte Zielformate möglich ist. Dieselbe Aussage lässt sich für DSL im Rahmen von MDSD treffen. Somit sind hier beide Ansätze gleich stark.

Das Erweiterbarkeitsaspekt ist eng an das bereits diskutierte Kriterium der Anpassbarkeit gebunden. Dort wurde festgestellt, dass UML zwar grundsätzlich eine Anpassung oder Erweiterung ermöglicht, dies hat jedoch Auswirkungen auf alle bereits bestehenden Testspezifikationen. Dahingegen hat die Diskussion in diesem Abschnitt ergeben, dass bei den DSL-basierten Ansätzen im Rahmen der MDSD der Grad an Erweiterbarkeit wesentlich höher ist, als der bei UML/UTP. Dies liegt an der Möglichkeit, Metamodelle für Modellierungsmethoden domänenspezifisch zu definieren sowie sehr effektive Tools zu entwickeln, welche Teile der Metamodelle benutzen können, um das bereits angesprochene Sichtenkonzept zu realisieren.

3.3.1.3 Datenaustauschformate

Eine besondere Stellung nimmt ein Datenaustauschformat aus dem automobilen Bereich namens ASAM ODS [Bar10] ein. ASAM steht für „Association for Standardisation of Automation and Measuring Systems“ und ODS steht für „Open Data Services“. Die Bestandteile der ASAM ODS Spezifikation sind in der Abbildung 3.20 gezeigt. Es wird zwischen folgenden Spezifikationsbestandteilen unterschieden:

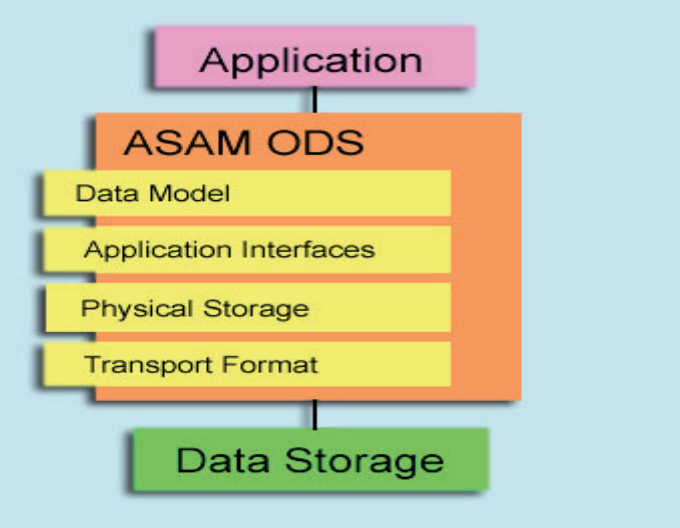


Bild 3.20: Bestandteile des ASAM ODS Standards [Bar10]

- Ein allgemeines Datenmodell (engl. „base model“) für eine eindeutige und vollständige Beschreibung der aufgenommenen Messdaten im automobilen Bereich. Diese Daten werden als Metadaten betrachtet, die eine Semantik den aufgenommenen Messdaten hinzufügen. Dadurch soll es laut ASAM „verschiedenen Systemen möglich sein, dieselben Daten auf dieselbe Art und Weise zu interpretieren“. Durch die Bildung eines so gennanten Anwendungsmodells (engl. „Application Model“) soll die Adaption des allgemeinen Datenmodells an die Bedürfnisse eines konkreten Systems oder sogar Projekts möglich sein.

- Anwendungsinterfaces (engl. „Application Programmer’s Interfaces, APIs), um auf die Daten von ASAM-kompatiblen Systemen und Werkzeugen auf eine standardisierte Art und Weise zugreifen zu können.

- Physikalische Abspeicherung: Beinhaltet Anweisungen zur physikalischen Abspeicherung des Datenmodells in den gängigen Datenbanksystemen.

- Transport Format: ASAM Transport Format (ATF) ist ein ASCII basierter Austauschformat, um sämtliche ASAM ODS Daten zwischen den verschiedenen Systemen auszutauschen. Seit der Version 5.0 gibt es auch eine XML-basierte Version von ATF.

Aus Sicht dieser Arbeit ist sicherlich der „Data Model“-Teil der Spezifikation interessant, da er gerade versucht, einen Modell-basierten Ansatz zur Definition von Metadaten zur Beschreibung

einer Messung zu verfolgen mit besonderem Hinblick auf die Austauschbarkeit von solchen Metadaten.

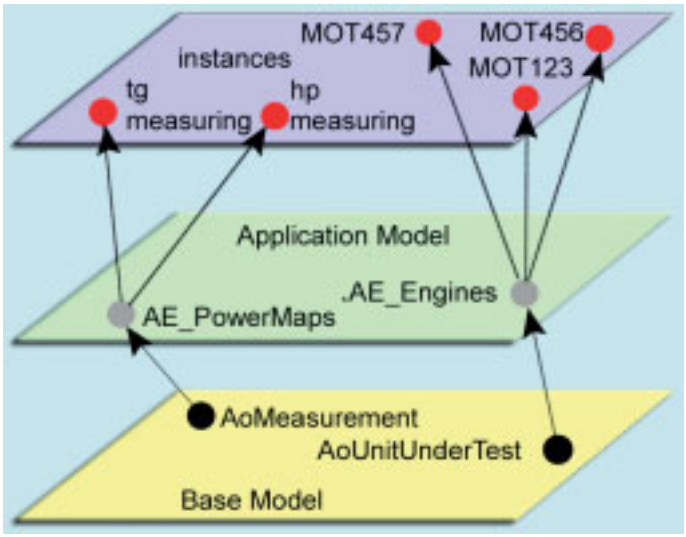


Bild 3.21: Bestandteile des ASAM ODS Standards [Bar10]

Der prinzipielle Aufbau des ODS Datenmodells ist in der Abbildung 3.21 dargestellt. Alle Elemente des allgemeinen Datenmodells (engl. Base Model) sind von ASAM festgelegt und sie dürfen nicht instanziiert werden. Sie entsprechen dem M1-Niveau des Meta Object Facility (MOF) [Obj10a]. Auf derselben Modellierungsebene befindet sich das Anwendungsmodell. Jedes Element des allgemeinen Datenmodells ist durch eine fixe Menge an Attributen und Beziehungen beschrieben. Die Elemente des Anwendungsmodells sind von Elementen des allgemeinen Datenmodells abgeleitet. Die Ableitung bedeutet in diesem Fall eine „typeof“ Beziehung zwischen den Elementen des Anwendungsmodells und des allgemeinen Datenmodells. Alle Elemente des Anwendungsmodells dürfen eigene Attribute und Beziehungen zu sich selbst oder zu anderen Elementen definieren. Sie müssen allerdings bestimmte Attribute des allgemeinen Datenmodells vererben. Der Standard schreibt fest, welche Attribute und Beziehungen des allgemeinen Datenmodells im jeweiligen Element des Anwendungsmodells undbedingt erscheinen müssen und welche nur optional sind. Schließlich werden auf der M0 Ebene des MOF die einzelnen Instanzen des Anwendungsmodells definiert. Der Grundgedanke dieser Vorgehensweise ist folgender. Jedes Projekt definiert für jede innerhalb dieses Projekts eingesetzte Methode ein solches Anwendungsmodell. ASAM verspricht sich, die Semantik eines jeden Elements des Anwedungsmodells dadurch zu definieren, dass es von einem bestimmten Typ des allgemeinen Modells abgeleitet ist. Aus Sicht des ASAM besteht darin der Schlüssel zur Austauschbarkeit von solchen „Metabeschreibungen

von den Messdaten“ über die Projektgrenzen hinweg. Denn jedes beliebige Projekt, das die Beschreibung eigener Messdaten auf ASAM ODS aufbaut, kann von einem fremden Projekt zumindest den Typ der jeweiligen im Anwendungsmodell verwendeten Elemente ermitteln. In der Abbildung 3.22 sind die Elemente des allgemeinen Datenmodells gruppiert dargestellt. Für deren genaue Beschreibung sei auf die Spezifikation [Bar10] verwiesen. Schließlich ist an einem Bei-

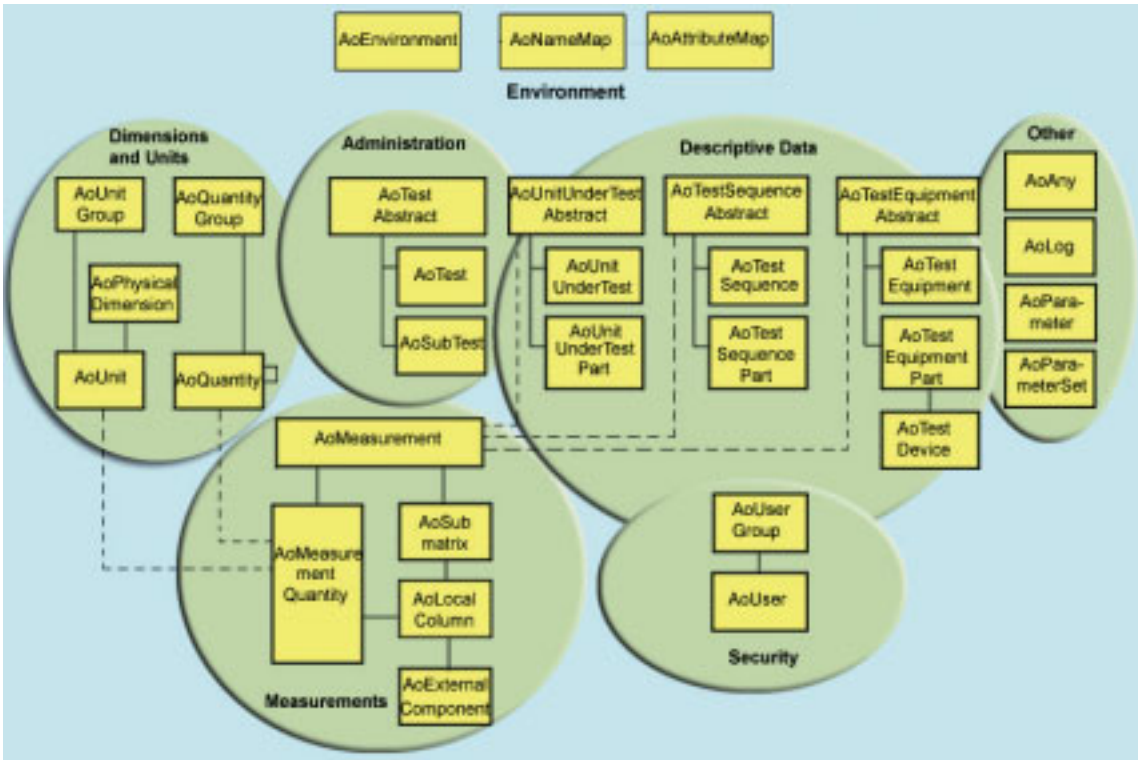


Bild 3.22: Elemente des Basismodells [Bar10]

spiel der Zusammenhang zwischen allgemeinen Datenmodell und Anwendungsmodell dargestellt (Abbildung 3.23). Jede Messaktion („AoMeasurement“) wird in einem bestimmten projektspezifischen Anwendungsmodell als Gewinnung von Messdaten bezeichnet (engl. „Acquisition“). Jedes „Acquistion“-Element ist vom Typ „AoMeasurement“. Als SUT wird das Fahrzeug (engl. „Vehicle“ vom Typ „AoUnitUnderTest“ im allgemeinen Datenmodell) betrachtet. Zur Messungsaufnahme ist es für dieses bestimmte Projekt notwendig, das SUT in weitere Bestandteile zu untergliedern: Motor (engl. Engine), Reifen (engl. Tire) und schließlich das Getriebe (engl. Gearbox). Der Motor besteht wiederum aus Kurbelachse und Zylindern. Jedes der gerade beschriebenen Elemente ist vom Typ „AoUnitUnderTestPart“ im allgemeinen Datenmodell.

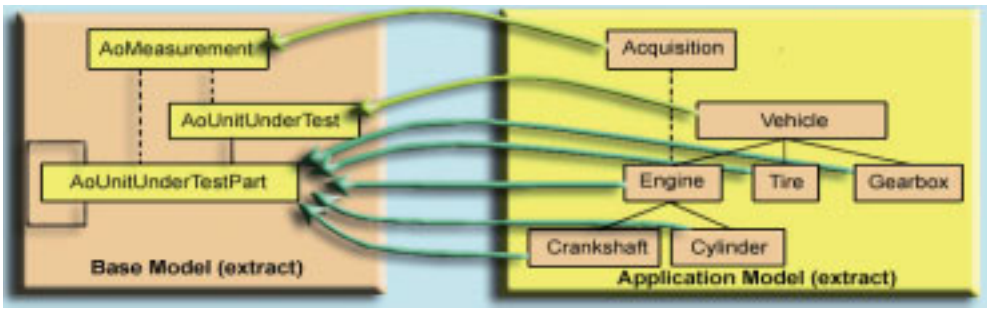


Bild 3.23: Zusammenhang zwischen dem allgemeinen Datenmodell und dem Anwendungsmodell [Bar10]

Bewertung von ASAM ODS

Das allgemeine Datenmodell von ASAM ODS ist korrekt, da ihm sowohl eine Spezifikation als auch ein vollständig spezifiziertes Datenmodell zugrunde liegen, anhand von welchen die Korrektheit der von den erzeugten Messdaten-Spezifikationen nachgewiesen werden kann. Außerdem exisitiert mit ATF ein XML-Format zum Austausch von Messdatenbeschreibungen, welches auf einem XML-Schema basiert. Somit kann die Überprüfung der Korrektheit auch werkzeugbasiert verlaufen. Was die Vollständigkeit betrifft, so wurden hier einige Schwächen festgestellt. Zum einen ist in der Spezifikation des allgemeinen Datenmodells nicht für jedes Element klar spezifiziert, welche Semantik es besitzt. Betrachtet man zum Beispiel das Element „AoTestSequence“, so ist es ohne eine Definition unklar, ob es sich um Testdaten handelt, um einzelne Testaktionen oder gar um Schritte, welche zur Aufnahme einer Messung notwendig sind. Außerdem ist das Vorhandensein des Begriffs „Test“ bei vielen Elementen sehr verwirrend, da es sich beim Standard um die „Verwaltung von Messdaten auf eine von einer bestimmten Architektur unabhängigen Weise“ [Bar10] handelt. Es sind klarerweise viele Elemente vorhanden, welche auch in einer Testspezifikation ihren Platz haben wie beispielsweise SUT-Konfiguration („AoUnitUnderTest“), Testfall(„AoTest“). Gleichzeitig konnten solch wichtige Bestandteile einer Testspezifikation wie Referenzwerte nicht gefunden werden. Was die Testdaten betrifft so können diese vermutlich über das Element „AoTestSequence“ beschrieben werden. Dies kann man jedoch nicht mit hundertprozentiger Sicherheit behaupten, da die Semantik des Elementen in der Spezifikation nicht genau definiert ist. Das allgemeine Datenmodell legt allerdings nicht fest, wie die Testdaten zu beschreiben sind. Dies ist nicht verwunderlich, da ASAM ODS den Anspruch hat, möglichst alle Messdatenformate beschreiben zu können. Um Aussagen über die Einheitlichkeit treffen zu können, muss bei allen Elementen des allgemeinen Datenmodells klar definiert sein, für welche

Zwecke sie eingesetzt werden beziehungsweise was sie darstellen, um dann entsprechende Elemente im Anwendungsmodell von ihnen ableiten zu können. Bei einigen Elementen des ASAM ODS ist es möglich, bei anderen wiederum nicht.

Von allen untersuchten Ansätzen ist ASAM ODS der wohl am weitesten auf die Wiederverwendbarkeit und Anpassbarkeit an die Bedürfnisse eines bestimmten Projekts ausgerichtete Standard. Durch die Möglichkeit, ein eigenes Anwendungsmodell von einem allgemeinen Datenmodell abzuleiten, ermöglicht er die Definition eigener projektspezifischer Attribute und ihre Benennung gemäß der Terminologie, die im Projekt benutzt wird. Allerdings lässt sich auch bei diesen zwei Kriterien folgendes feststellen. Dadurch, dass im allgemeinen Modell nicht die Unterscheidung zwischen projektspezifischen und generischen Attributen gibt, werden unterschiedliche Abteilungen nicht dazu dringend gezwungen, auf der Ebene des Applikationsmodell Gedanken darüber zu machen, was an den Beschreibungsinformationen für die Messungen generisch ist oder sein kann und was nur für ein bestimmtes Projekt gilt. Dies führt dazu, dass beim Austausch der Messdatenbeschreibungen über Projektgrenzen hinweg eine empfangende Abteilung über das allgemeine Datenmodell zwar die Semantik des Elementen im Anwendungsmodell kennt (z.B. „beim Element „Vehicle“ handelt es sich um ein SUT, denn das Element „Vehicle“ ist vom Typ „AoUnitUnderTest“), sie muss aber möglicherweise komplett neu die entsprechenden Attribute für das Element „Vehicle“ definieren, da die vorhandenen nur innerhalb der gebenden Abteilung angewandt werden können. Es wäre also sinnvoll, das vorhandene allgemeine Modell in dieser Hinsicht zu erweitern. Tatsächlich wurde bei der bei AUDI entstehenden Plattform „Messdatamanagement“ (ASAM ODS basiert) [Aud10] beobachtet, dass dort jeder Geschäftsbereich für sich eigene Attribute definiert, was zweifelsohne richtig ist, allerdings gibt es bei der Definition keinerlei Möglichkeit ein Beschreibungsattribut als generisch zu definieren, um in einem nächsten Schritt sich mit anderen Geschäftsbereichen darüber abzustimmen. Da ASAM ODS durch seinen Anspruch auf Allgemeingültigkeit (zumindestens im automobilen Bereich) keine klaren Vorgaben auf der „Base Model“ Ebene darüber macht, durch welche Methoden beispielsweise Testdaten beschrieben werden, kann dies ebenfalls dazu führen, dass zwei Abteilungen, welche eventuell Messdatenbeschreibungen austauschen würden, in ihren Anwendungsmodellen unterschiedliche Methoden zur Beschreibung beispielsweise von Fahrszenen wählen: Aktivitätsdiagramme und Statecharts. Dies führt zur schlechten Operationalisierbarkeit von solchen Beschreibungen. Denn die Abteilung, welche die Fahrszenen mit Statecharts beschreibt, wird zwar die Aktivitätsdiagramme auch verstehen, sie wird sie aber ohne weiteres in andere plattformspezifische Formate nicht transformieren können, weil dazu erst Transformatoren geschrieben werden müssen. Sicherlich gilt, würde man auf der „Base Model“ Ebene eine präzise Form zur Darstellung von bestimm-

ten Teilen der Messdatenbeschreibung vorgeben, dann würde ASAM ODS möglicherweise den Anspruch auf die Allgemeingültigkeit verlieren. Doch wurde in der Praxis beobachtet, dass die Ansätze Erfolg haben und von Dauer sind, welche versuchen, Lösungen zu finden, die zwar für eine spezifische Domäne gültig sind, dafür aber Projekt- und Unternehmensgrenzen-übergreifend eingesetzt werden können.

Es können nicht alle anwenderbezogenen Kriterien von ASAM ODS beurteilt werden, da sie sehr stark von dem jeweiligen Anwendungsmodell abhängig sind. Generell würde aber eine genaue textuelle Definition von einzelnen Elementen im Allgemeinen Datenmodell die Verständlichkeit des Standards erhöhen. Anwendbarkeit ist ebenfalls sehr stark von dem jeweiligen Anwendungsmodell abhängig. Es wird aber davon ausgegangen, dass ASAM ODS durch freie Definierbarkeit von Anwendungsmodellen für einzelne Projekte, die ihre Messdaten mit dieser Methode beschreiben wollen, einen hohen Grad an Anwendbarkeit besitzt.

Was die Operationalisierbarkeit betrifft, so ist diese innerhalb einzelner Projekte gegeben. Abteilungs- und projektübergreifend können allerdings Schwierigkeiten entstehen, da im Allgemeinen Datenmodell bestimmte Elemente nicht näher spezifiziert sind (Siehe auch die Diskussion oben). Die Erweiterbarkeit von Messdaten-Beschreibungen ist innerhalb einzelner Projekte gegeben.

3.3.2 Auswahl der Methodik zur Modellierung von Testprozessen

Die Analyse der im Kapitel 2 untersuchten Testprozesse im Unternehmen hat vor allem ergeben, dass die verwendeten Prozessmodellierungsmethoden eher zu Dokumentations- und Informationszwecken verwendet werden. Die graphisch oder textuell beschriebenen Testprozesse werden meistens entweder in Form von elektronischen Dokumenten [Mic10a] oder in dedizierten Prozessportalen [Met10] abgelegt. Es handelt sich also um so genannte Soll-Prozesse, die vor Beginn der Ausführung eines Testprozesses Auskunft darüber geben sollen, wie der Testprozess ablaufen soll und vor allem welche Artefakte in welchem Prozessschritt entstehen sollen und in welcher Form. Es wäre jedoch sinnvoll, solche Prozessunterstützungslösungen anzubieten, welche aktiv, also zur Laufzeit, zur Steuerung eines Testprozesses beitragen, indem sie die Aktivitäten einzelner am Prozess beteiligter Personen koordinieren und diese über den Stand einzelner Schritte informieren. Um dies zu gewährleisten, müssen Testprozesse nicht nur graphisch dokumentiert werden können, sondern sie müssen vor allem operationalisierbar und formal beschrieben sein. Im Weiteren wird nun untersucht, inwieweit die aktuell eingesetzte Methode dies gewährleistet und welche bestehenden Modellierungsmethoden eventuell diese Kriterien erfüllen.

Ereignisgesteuerte Prozessketten

Im Rahmen der bei HiL-Testprozessen eingesetzten ARIS-Methodik [Sch01] werden in der Steuerungsschicht so genannte Ereignisgesteuerte [Sch01] Prozessketten (EPK) zur strukturierten Darstellung von Testprozessen verwendet. EPK ist eine graphische Methode, die einen gerichteten Graphen darstellt, welcher aus drei Modellierungselementen besteht:

- Funktion: „Aufgabe, die eine durch physische oder geistige Aktivitäten zu verwirklichende Soll-Leistung darstellt. Im Rahmen der Informationsmodellierung steht somit das „Ziel“ und nicht der Weg, mit dem das Ziel erreicht wird, im Mittelpunkt. Der hier definierte Funktionsbegriff bezieht sich somit auf das „was“ und nicht auf das „wie“ der Funktion“ [Sch96].

- Ereignis: „Ein Ereignis ist in Anlehnung an die DIN 69900 das Eingetretensein eines definierten Zustandes, der eine Folge von Aktivitäten bewirkt. Abzugrenzen von ereignisbezogenen Zuständen sind Systemzustände, die keine unmittelbare Folgewirkungen für das System haben. Innerhalb der Informationsmodellierung stellt ein Ereignis einen eingetretenen Zustand eines oder einer Gruppe von Informationsobjekten dar. Ein Ereignis ist somit eine passive Komponente des Informationssystems“ [Sch96].

- Verknüpfungsoperatoren: Diese können zwischen den Ereignissen und Funktionen auftreten. Beispielsweise handelt es sich um wohlbekannte logische Operatoren wie UND, OR, XOR.

Funktionen werden von einem Auslösemechanismus gestartet, dem Ereignis. Ereignisse starten somit Funktionen und können wiederum ein Ergebnis von Funktionen sein. Ein Ereignis ist somit das Eingetretensein von Ausprägungen (Werten) von Attributen, das eine Funktion auslöst [Sch96]. Ein beispielhafter Prozess aus dem HiL-Bereich (EXAM) ist in der Abbildung 3.24 abgebildet. Nachdem der Testadministrator einen Testauftrag erhalten hat (Ereignis „Testauftrag erteilt“), überprüft er, ob die zur Durchführung notwendigen Testfälle in der entsprechenden Datenbank sind (Funktion „Testpool prüfen“). Der Testadministrator verwaltet den Testpool (Gesamtheit aller verfügbaren Testfälle) bzw. einen Teil davon. Er hat den Überblick über vorhandene Testfälle und veröffentlicht gegebenenfalls neue Testfälle. Ein Testauftrag stellt eine offizielle Beauftragung der Durchführung von Prüfungen gemäß einer Testspezifikation dar. Die Testspezifikation ist also ein zwingender Bestandteil des Testauftrags. Falls dies der Fall ist, dann wird der entsprechende Anpassungsauftrag an den Testdesigner erteilt, welcher den Testfall an einen vorhandenen Prüfstand anpasst. Andernfalls wird ein Auftrag zur Neuentwicklung der notwendigen Testfälle an den Testdesigner erteilt. Der Testdesigner entwirft wiederum die notwendigen Testfälle und

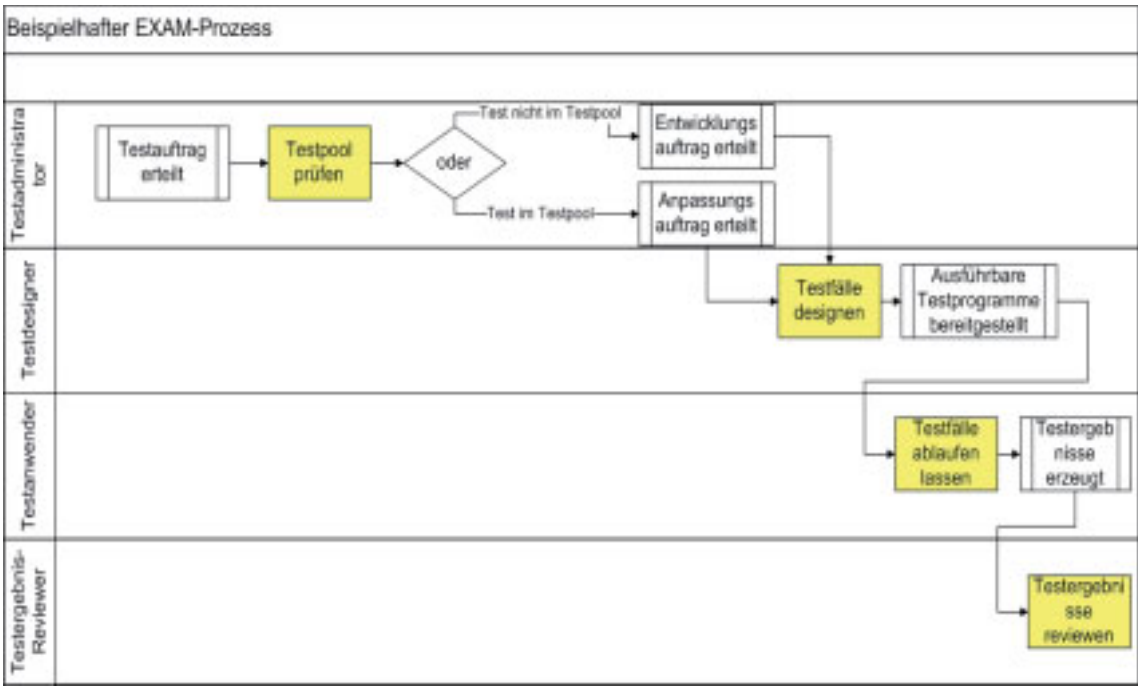


Bild 3.24: Beispielhafter Prozess mit EPK[Mic10a]

erzeugt aus ihnen ausführbare Programme beispielsweise in Form eines Skripts. Im nächsten Schritt werden die vom Testdesigner erzeugten Testfälle von Testanwender ausgeführt. Der Testanwender ist für die Ausführung und vor allem Ablaufüberwachung der Testfälle am Prüfstand zuständig. Schließlich muss der Testreviewer die produzierten Testergebnisse reviewen und auf ihre Korrektheit im Sinne des definierten Prozesses überprüfen. Die aufgetretenen Probleme werden von ihm dokumentiert. Es findet keine inhaltliche Überprüfung statt. Dies ist die Aufgabe des Auftraggebers, der meistens eine Entwicklungsabteilung ist [Mic10a].

Zur Bewertung von EPK lässt sich anmerken, dass trotz geringer Anzahl an Modellierungskonstrukten und leichter Erlernbarkeit es für EPK keine wohldefinierte Syntax und Semantik gibt [van99]. Dies bedeutet, dass es keinerlei Mechanismus exisitiert, der es erlaubt, korrekte EPK Prozessbeschreibungen von den nicht korrekten zu unterscheiden. Ebenfalls ist die Operationalisierbarkeit von den EPK dadurch gefährdet. Deshalb wird oftmals versucht, die ereignisgesteuerten Prozessketten anhand von bestehenden Formalismen, wie beispielsweise Petrinetzen [van99], auf Korrektheit zu prüfen. Dazu muss EPK in die Notation von Petrinetzen übersetzt werden. Eine andere Möglichkeit wäre natürlich, eine eigene Semantik durch eine entsprechende DSL zu definieren. Durch keine klare Semantik- und Syntaxisdefinition ergeben sich ebenfalls sehr große Interpretationsräume, die möglicherweise das eindeutige Verständnis solcher Darstellungen von

allen Testprozess-Beteiligten gefährden können und zu Problemen bei maschineller Verarbeitung führen. Es ist also deutlich zu sehen, dass ohne weitere Nacharbeitung die EPKs lediglich zu Dokumentationszwecken dienen können.

Business Process Modelling Notation

Die erste Spezifikation von Business Process Modelling Notation (BPMN) [Obj08a] erschien im Jahr 2004. BPMN ist ein Standard, das ursprünglich von „The Business Process Management Initiative“ entwickelt wurde. Später wurde die Formalisierung von BPMN von OMG übernommen. Aktuell liegt BPMN in Version 1.1 vor. Das Hauptziel von BPMN ist „to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. BPMN will also be supported with an internal model that will enable the generation of executable BPEL4WS. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation“ [Whi08]. Um die gerade beschriebene Lücke zu schließen wird in der Spezifikation von BPMN die Abbildung auf ausführbare XML-Sprache für Geschäftsprozesse namens BPEL4WS dargestellt. Dies impliziert, dass BPMN an sich lediglich zu Dokumentationszwecken (ähnlich wie EPK) dient. Allerdings geht BPMN einen Schritt weiter als EPK, indem sie sich explizit die Ausführbarkeit (auch wenn über einen Transformationsschritt) als Ziel setzt. Im Vergleich zu EPK besitzt BPMN eine höhere Anzahl an Konstrukten (siehe Abbildung 3.25). So wird beispielsweise zwischen drei Typen von Ereignissen

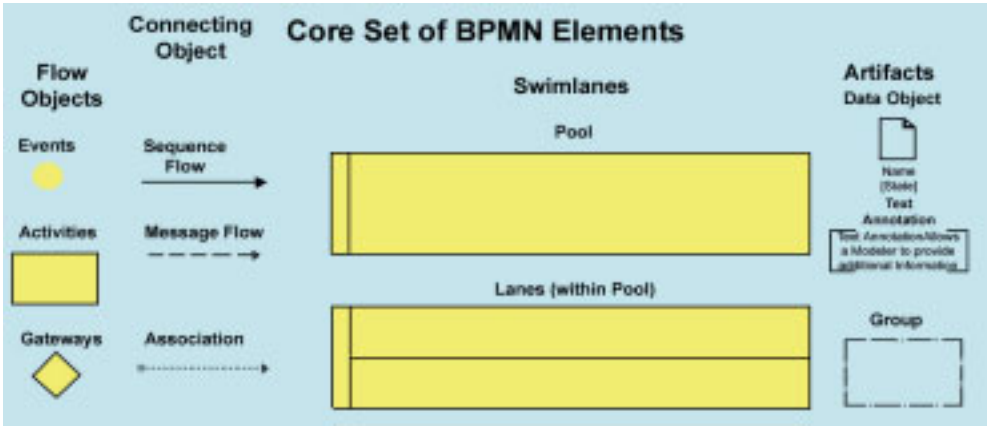


Bild 3.25: Grundlegende BPMN-Elemente [Whi08]

unterschieden: Diejenigen, die einen Prozess auslösen, jene, die mitten im Prozess auftreten können und schließlich Endereignisse, welche am Ende eines Prozesses vorkommen. Ebenfalls ist eine Verfeinerung von Flüssen zu beobachten. Zum einen gibt es Sequenzflüsse, welche Ereignisse und Aktivitäten verbinden, zum anderen exisitieren so genannte Nachrichtenflüsse, die Daten zwischen den Aktivitäten und Ereignissen transportieren. Außerdem exisiteren noch die Assoziationen, mit denen diversen Aktivitäten Artefakte zugewiesen werden können. Der beispielhafte EXAM-Testprozess, der mit BPMN modelliert wurde, ist in der Abbildung 3.26 zu sehen.

Die Bewertung von BPMN fällt im Vergleich zu EPK deutlich positiver aus. Dies betrifft vor allem die Vollständigkeit, welche in einer äußerst umfangreichen Spezifikation definiert ist. Ebenfalls ist, wie bereits erwähnt, die Operationalisierbarkeit von BPMN wesentlich höher als die von EPK. BPMN verfügt über einen hohen Grad an Verständlichkeit, da zum einen die Anzahl an Konstrukten relativ gering ist und zum anderen die Semantik der Konstrukte recht intuitiv ist und zunächst keine tiefergehenden mathematischen Kenntnisse erfordert. Es lassen sich aber auch einige Schwächen von dieser Methode feststellen, welche ihre Anwendung im Rahmen der Arbeit erschweren. Die Abbildungsregeln sind zurzeit nur für BPEL4WS in der OMG-Spezifikation definiert. Die Erweiterbarkeit von BPMN ist nur bedingt möglich. Die Spezifikation erlaubt außerdem lediglich die Definition von neuen Artefakten, wodurch die Anpassbarkeit an die Beschreibung von Testprozessen eingeschränkt wird, da dort möglicherweise spezifische Attribute für die Beschreibungselemente definiert werden müssen.

UML-Aktivitätsdiagramme

In der Version 2.0 von UML sind so genannte Aktivitätsdiagramme [Obj09] zur Modellierung von Prozessen beinhaltet. Die Aktivitätsdiagramme von UML haben ihren Ursprung in den Zustandsautomaten, Flussdiagrammen sowie den Petrinetzen [SA03]. Das wichtigste Modellierungselement ist eine Aktion (engl. Action). Eine Aktion „takes a set of inputs and converts them to a set of outputs, though either or both sets may be empty“ [Obj09]. Aktionen können ebenfalls den Zustand des Systems verändern [Sch96].

In der Abbildung 3.27 ist ein beispielhafter EXAM-Testprozess mittels Aktivitätsdiagramm dargestellt. Die Bewertung von UML wurde bereits in diesem Kapitel unternommen. Dies geschah jedoch auf einer niedrigeren Abstraktionsebene und zwar auf der Ebene der Testspezifikationen. Auf einer höheren Abstraktionsebene und zwar auf dieser der Testprozesse scheinen die Aktiviätsdiagramme für die bisher untersuchten Testprozesse am besten geeignet zu sein, denn:

- Wie bereits erwähnt, liegt UML ein Metamodell (MOF) zugrunde. Dadurch ist Korrektheit von Aktivitätsdiagrammen garanitert. Die Vollständigkeit ist durch das Vorhandensein einer

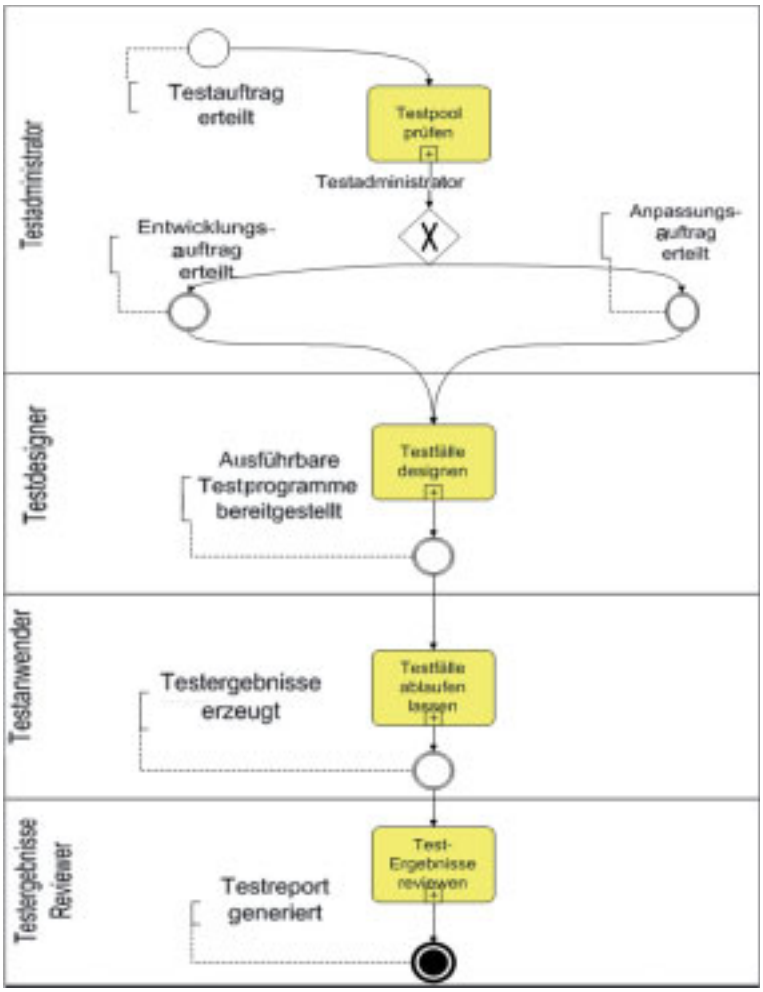


Bild 3.26: EXAM-Testprozessbeschreibung mit BPMN

umfangreichen Spezifikation [Obj09] gewährleistet.

- Ein leichtgewichtiger Erweiterungsmechanismus ist in Form von Stereotypen gegeben, sodass ein Mindestmaß an Anpassbarkeit von Prozessbeschreibungen durch die Beschreibungssprache garantiert ist. Es müssen allerdings immer die Aspekte in Betracht gezogen werden, die bei der Bewertung von UML im Rahmen der Beschreibung der Testspezifikationen erwähnt wurden.

- Aktivitätsdiagramme sind leicht zu erlernen vor allem auch für Menschen mit geringem technischen Hintergrund, welche Testprozesse analysieren möchten, denn die Flussdiagramme, von welchen Aktivitätsdiagramme teilweise abstammen, auch beispielsweise in der betriebswirtschaftlichen Domäne wohlbekannt sind.

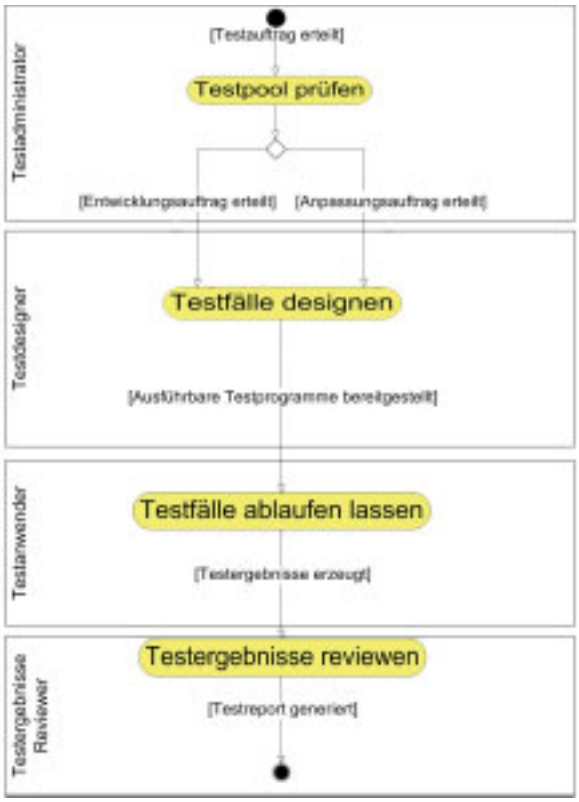


Bild 3.27: Das beispielhafte EXAM-Testprozess mit einem Akivitätsdiagramm modelliert

- Operationalisierbarkeit ist bei UML durch solche Ansätz wie MDA [MSUW04] oder Executable UML [MBB02] gegeben. Grundsätzlich kann man unter Anwendung von MDA die durch Aktivitätsdiagramme definierten Prozesse auf unterschiedlichste ausführbare Notationen abbilden. Unter anderem wurde von IBM [Man05] beschrieben, wie unter Anwendung eines geeigneten Profils die in UML beschriebenen Aktivitätsdiagramme auf BPEL abgebildet werden können.

Abschließend lässt sich feststellen, dass von den untersuchten Ansätzen zur Prozessmodellierung und für die bisher untersuchten Testprozesse UML-Aktivitätsdiagramme am besten geeignet zu sein scheinen, diese zu modellieren. Vor allem herauszuheben ist gute Operationalisierbarkeit im Vergleich zu BPMN und EPK. Eventuell können aber Probleme bei der Anpassbarkeit an Anforderungen der Testprozesse diverser Geschäftsbereiche eines Industrieunternehmens entstehen. Dies betrifft vor allem die Verhältnissmäßigkeit des Aufwandes, der zur Anpassung notwendig ist [Mü07].

3.3.3 Fazit

Abschließend ist in der Abbildung 3.28 eine zusammenfassende Betrachtung und Bewertung aller untersuchten Methoden zur formalen Definition von Testspezifikationen gegeben. Es ist deutlich zu sehen, dass basierend auf den in dieser Arbeit definierten Zielen die modellgetriebene Software-Entwicklung als Methodik ausgewählt werden soll. Letztendlich sind die entscheidenden Kriterien aus Sicht dieser Arbeit (Wiederverwendbarkeit, Erweiterbarkeit, Wartbarkeit) bei MDSD allesamt ohne feststellbare Schwächen erfüllt. Es muss gleichzeitig betont werden, dass MDSD lediglich

Kriterium/Methode	SCR	CSP	TSL	TTCN-3	MDA	UTP	MDSD	ASAM ODS
Korrektheit	+	+	-	+	+	+	+	+
Vollständigkeit	+	+	-	+	+	+	0/+	0/-
Einheitlichkeit	0	0	0	0/+	-	-	0/+	-
Wiederverwend-barkeit	-	-	+	0/-	+	0/-	+	0/+
Wartbarkeit	-	-	0	0/-	0/-	0/-	+	0/+
Einfachheit	-	+	0	+	0/+	0/+	+	0/-
Anwendbarkeit	+	-	-	+	+	+	+	+
Verständlichkeit	-	-	-	0	+	+	+	0
Anschaulichkeit	-	-	-	0	0/+	0/+	+	0
Erweiterbarkeit	-	-	-	0/-	0/+	0/+	+	+
Ausdrucksmächtig-keit	+	-	-	+	0/+	0/+	+	-
Operationalisier-barkeit	0/+	0/+	-	0/+	+	+	+	0/+

Bild 3.28: Testspezifikationsmethoden: Abschließende Betrachtung

ein Rahmenwerk zur Verfügung stellt dafür aber keinen Bezug auf die Testvorgänge und vor allem auf die Definition und Verwendung von Testspezifikationen nimmt. Es muss also ein umfassendes Vorgehensmodell definiert werden, welches die Definition, Verwaltung, Wiederverwendung und Transformation von domänenspezifischen Testspezifikationen in einem Industrieunternehmen ermöglicht. Dies geschieht im Kapitel 4 „Das wissenschaftliche Konzept“. Im Rahmen dieses Vorgehensmodells müssen vor allem folgende Aspekte berücksichtigt werden:

- Eingehende Untersuchung der zu testenden Domäne: Es muss spezifiziert werden, was konkret untersucht werden muss, worauf zu achten ist und wie die Ergebnisse dokumentiert und weiterverwendet werden können.
- Formalisierung der Testspezifikationsbeschreibung: Hierbei gilt es festzustellen, welche domänenspezifischen Anpassungen und vor allem Ergänzungen für die Wahl bestehender formaler Methodiken zur Definition von Testspezifikationen getroffen werden müssen.
- Validierung der Beschreibung: Ohne Zusammenarbeit mit dem Domänenexperten bei der Wahl und Anpassung der Spezifikationsmethode ist der eflogreiche Einsatz formaler Testspezifikationen nahezu unmöglich. Aus diesem Grund muss bestimmt werden, wie im industriellen Kontext die Evaluierung der gewählten Methode am Effizientesten erfolgen kann. Selbstverständlich spielt hier der zeitliche Aspekt eine große Rolle.
- Wahl der Implementierungswerkzeuge: Worauf muss bei der Wahl von Technologien und Werkezugen geachtet werden, um die Verwendung von formalen, metamodellbasierten, domänenspezifischen Testspezifikationen im Entwicklungsprozess möglichst nahtlos einführen zu können.
- Vorgehensweise zur Pflege und Verwaltung von formalen metamodellbasierten Testspezifikationen: Da solche domänenspezifische Testspezifikationsbeschreibungen niemals unverändert bleiben, muss untersucht werden, wie die Erweiterbarkeit sowie eine zentralisierte Verwaltung von diesen zu gewährleisten ist.
- Schließlich muss eine für den Domänenexperten verständliche Vorgehensweise zur Konzeptanwendung definiert werden.

Kapitel 4

Das wissenschaftliche Konzept

In diesem Kapitel wird ein Vorgehensmodell definiert, welches es ermöglicht, für eine bestimmte Domäne, basierend auf dem Gedanken der Modell-getriebenen Software-Entwicklung, eine Zielplattform-unabhängige und formale Testspezifikation zu definieren. Das Modell wird allgemein gehalten ohne einen konkreten Bezug auf Fahrer-Assistenzsysteme herzustellen. Die domänenbezogene Konzeptanwendung wird im Kapitel 5 behandelt. Zum besseren Verständnis werden jedoch kleinere Beispiele aus der FAS-Domäne verwendet. Da das Vorgehensmodell auf dem Begriff „Szenario“ beruht, muss in einem ersten Schritt eine entsprechende Begriffsdefinition vorgenommen werden. Das gesamte Vorgehensmodell ist in der Abbildung 4.1 dargestellt. Jede der dargestellten Aktivitäten ist komposit und besteht dementsprechend aus mehreren Subaktivitäten. Im ersten Schritt muss die Wahl der Szenarienmodellierungsmethode für die Domäne erfolgen (Abschnitt 4.2). Das Ergebnis dieser Aktivität sind Fallstudien, der erste Prototyp der Anwendung sowie eine bestimmte Modellierungsmethode. Danach muss die Definition eines geeigneten Metamodells zur Formalisierung der gewählten Modellierungsmethode vorgenommen werden (Abschnitt 4.3). Innerhalb dieser Aktivität erfolgt ebenfalls die Wahl der Modellierungssprachen beziehungsweise der damit verbundenen Implementierungswerkzeuge. Die Modellierungssprachen beschreiben die gewählte Szenarienmodellierungsmethode auf maschinenlesbare Art und Weise. Als Ergebnis dieses Schrittes ist eine fertige technische Infrastruktur zur formalen Definition von Testspezifikationen zu betrachten. Nach der Umsetzung der Umgebung muss für jede beteiligte Abteilung, Projekt, Subprojekt und die Testmethodik eine partielle Instanziierung des Metamodells ausgeführt werden. Außerdem muss ein Transformationsmechanismus spezifiziert werden, welcher die Testspezifikationen aus einer plattformunabhängigen Notation in die plattformspezifischen umwandelt. All dies wird im Abschnitt 4.4 beleuchtet. Am Ende dieser Aktivität entsteht eine im täglichen Entwicklungsprozess einsetzbare Anwendung. Schließlich erfolgt die

Definition der Testspezifikation durch den Domänenexperten (Entwickler) (Abschnitt 4.5). Am Ende der Definition wird eine konkrete projektspezifische Testspezifikation erzeugt.

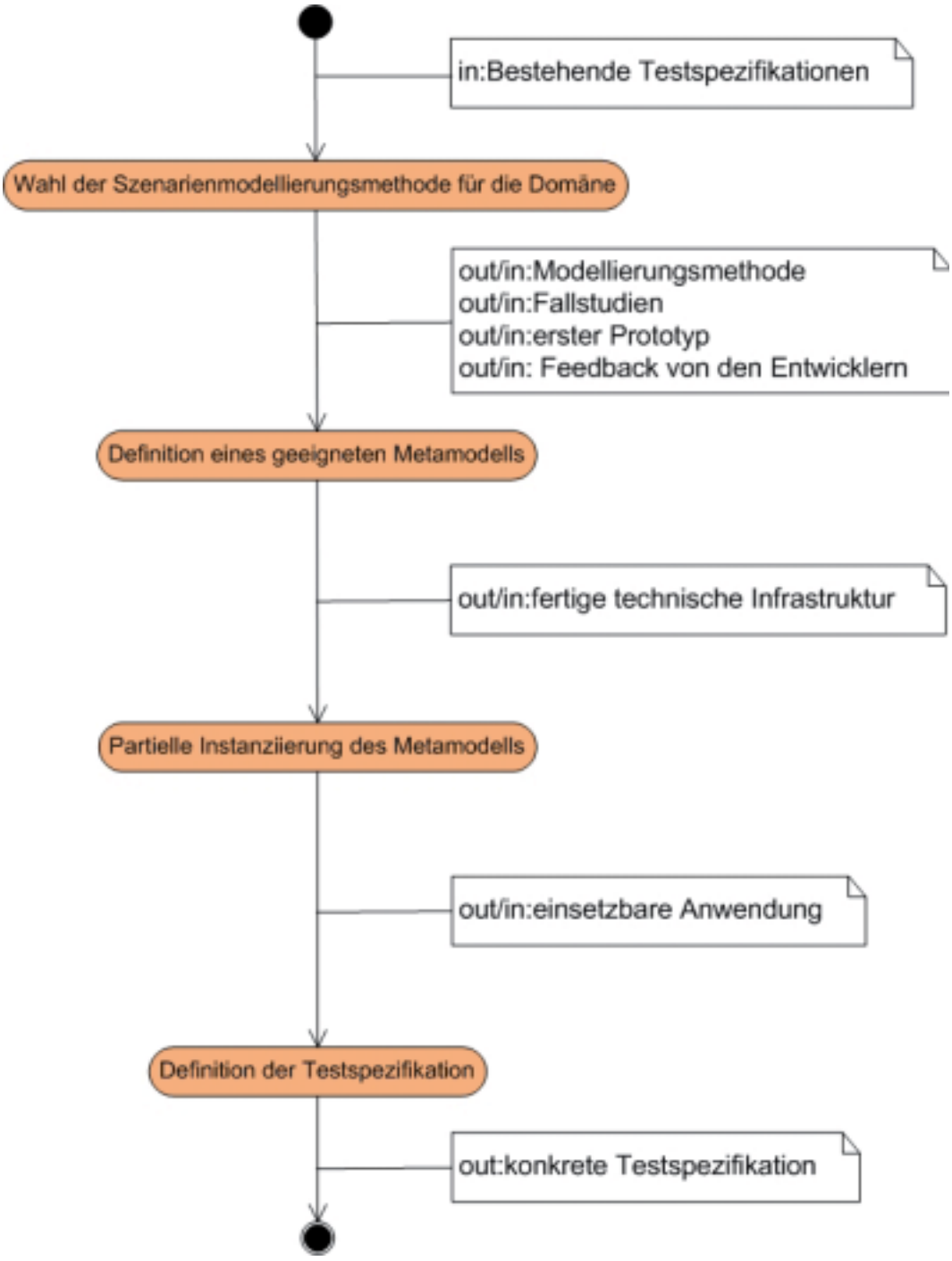


Bild 4.1: Das Vorgehensmodell

4.1 Begriffsklärung: Szenario

Im Kapitel 2 wurden Verhaltens- sowie die Interaktionsbeschreibung des SUT mit der Umwelt eingeführt. Zur Erinnerung beschreibt die Interaktionsbeschreibung das externe Verhalten des SUT samt dessen Umwelt, während die Verhaltensbeschreibung die internen Verhaltensaspekte berücksichtigt. Ein Szenario spezifiziert Verhaltens- und/oder Interaktionsbeschreibung des SUT auf verschiedenen Abstraktionsebenen. Ein Szenario definiert auf formale Art und Weise lediglich eine mögliche Benutzung/Interaktion oder eine mögliche Verhaltensbeschreibung des SUT. „Möglich“ bedeutet in diesem Zusammenhang, dass die Interaktionsbeschreibung oder die Verhaltensbeschreibung den Vorgängen in der Realität entspricht. Eine unmögliche Interaktionsbeschreibung würde demnach vorliegen, wenn beispielsweise ein EGO-Fahrzeug auf eine Position ausschert, auf welcher bereits ein Testobjekt fährt. Bei der Verhaltensbeschreibung spielen solche Aspekte eine Rolle wie interne Systemzustände, Eingangssignale, Ausgangssignale sowie Zustandsübergänge. Bei Interaktionsbeschreibung müssen neben den Eingangs- und Ausgangssignalen auch externe, also von außen beobachtbare, Systemzustände berücksichtigt werden. Außerdem spielt hier die Beschreibung der Umgebung eine große Rolle, in der sich ein SUT befindet und mit welcher es interagiert. Diese Umgebung kann statisch sein, das heißt sie ändert sich während der gesamten Szenariendauer nicht. Andererseits kann die Umgebung auch ein dynamisches Verhalten aufweisen, welches sich in der Zeit ändert.

Normalerweise sind Szenarienbeschreibungen schon in der Domäne vorhanden, allerdings, wie bereits dargestellt, in unterschiedlichsten Formaten, oftmals unstrukturiert und nicht formal. Aus diesem Grund muss in einem ersten Schritt die Formalisierung von solchen Szenarien vorgenommen werden (siehe Abschnitt 4.2).

In bestimmten Fällen können Szenarien durch Anforderungsdokumente definiert sein. Dies ist vor allem dann der Fall, wenn ein Szenario das gewünschte SUT-Verhalten beschreiben soll. Was die Interaktionsbeschreibung betrifft, so sind hier die Szenarienbeschreibungen oft unabhängig von den Anforderungsspezifikationen und sie existieren meistens entweder als separate Dokumente oder gar im Bewusstsein jeweiliger Domänenexperten. Der Hintergrund ist folgender: Die Soll-Reaktionen bei Interaktionsbeschreibungen sind sehr stark projektspezifisch. Würde man sie bei der Szenarienbeschreibung gleich auf der obersten Abstraktionsebene berücksichtigen, dann würde dies den Wiederverwendbarkeitsgrad eines Szenarios wesentlich verringern. Aus diesem Grund werden die Szenarien erst dann um Soll-Werte ergänzt, wenn es darum geht, ein generisches Szenario im Rahmen des Testens einer bestimmten „Functionality Under Test“ (FUT) zu verwenden. FUT stellt ein bestimmtes Modul im Rahmen des SUT dar. So ist beispielsweise Adaptive Cruise Control als ein Teil des Gesamtfunktionalität des Fahrzeugs (SUT) zu betrachten.

Was die Abstraktionsebenen der Szenariendefinition betrifft, so werden diese im Abschnitt 4.5 genauer definiert. Ein Szenario muss folgende Aspekte berücksichtigen:

- Zeitliche Aspekte: Es muss bei Interaktionsbeschreibungen klar definiert werden, welche Szenarioteilnehmer in welcher Reihenfolge mit dem SUT interagieren und wie das SUT darauf reagiert, wobei die SUT-Reaktion bei diesen nicht unbedingt das Soll-Verhalten darstellt (siehe Diskussion oben).

- Statische Aspekte: Ein Szenario muss imstande sein, Umweltbedingungen beschreiben zu können, welche zum Zeitpunkt seiner Ausführung geherrscht haben. Diese können beispielsweise Wetter, Fahrbahnzustände, Lichtkonditionen u.ä. umfassen.

- Ortsbezogene Aspekte: Vor allem bei Interaktionsbeschreibungen spielen die Positionen von den mit dem SUT interagierenden Szenarioteilnehmern, aber auch von SUT selbst eine wichtige Rolle. Ebenfalls müssen Positionsänderungen in der Zeit durch ein Szenario beschreibbar sein.

Im weiteren werden nun die im allgemeinen Vorgehensmodell definierten Aktivitäten näher betrachtet.

4.2 Wahl der Szenarienmodellierungsmethode für die Domäne

Die generelle Vorgehensweise zur Wahl einer formalen Szenarienmodellierungsmethode für die Domäne ist in der Abbildung 4.2 dargestellt. Im Weiteren werden nun einzelne Aktivitäten genauer betrachtet.

4.2.1 Analyse bestehender Testtechniken

Abteilungsübergreifend geschieht bei dieser Aktivität die Analyse der Testtechniken sowie vor allem der verwendeten Testspezifikationen. In dieser Arbeit werden die Begriffe Testtechnik sowie Testverfahren synonym verwendet. Zur Erinnerung ist Testverfahren ein Verfahren, das dazu eingesetzt wird, Software-Fehler im SUT zu finden. Dabei soll folgendes herausgestellt werden:

- Unternehmensstruktur in Bezug auf existierende Domäne: Hierbei muss festgestellt werden, welche Projekte oder Subprojekte sich mit der Entwicklung welcher Module (FUT) im

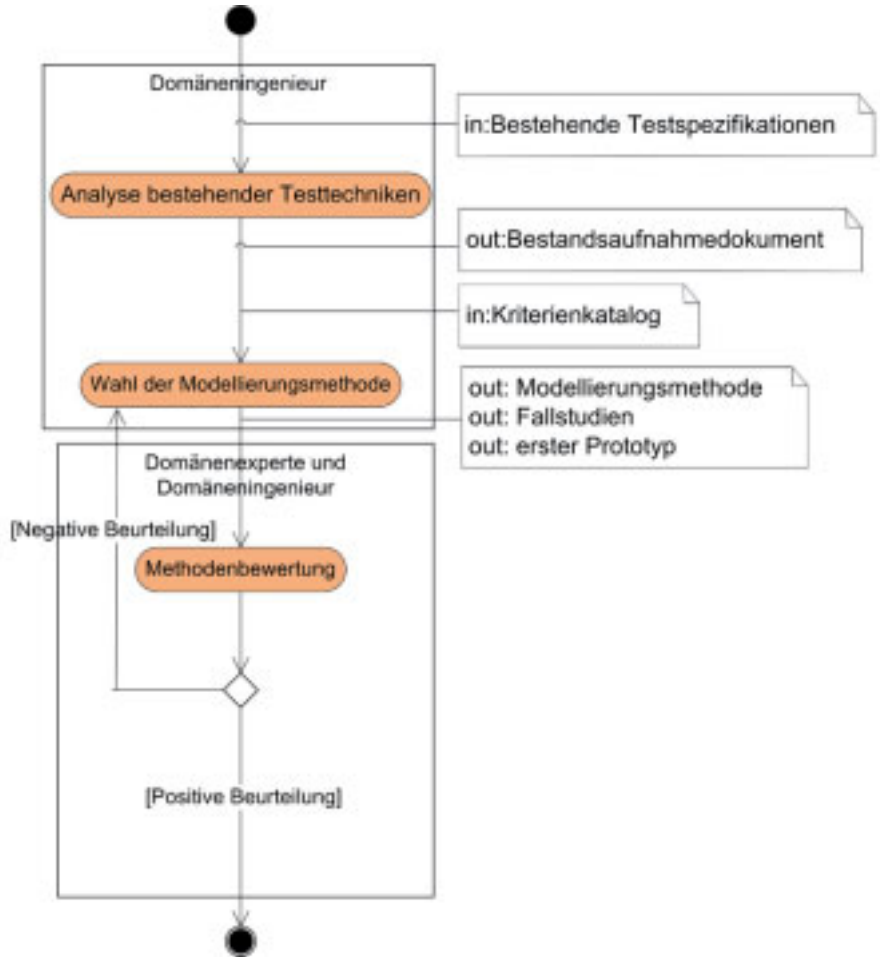


Bild 4.2: Szenarienmodellierungsmethode für die Domäne

ganzen Entwicklungsprozess in Bezug auf eine bestimmte Domäne beschäftigen. Zur Darstellung sind beispielsweise Organigramme sehr gut geeignet.

- Untersuchung der eingesetzten Testtechniken: Nachdem die Projekte und die einzelnen Subprojekte innerhalb dieser bekannt sind, sollen nun beispielsweise in Form von Interviews mit den Entwicklern einzelne Testverfahren ermittelt werden. Dabei muss vor allem herausgefunden werden, in welchen Formaten die Definition der Testspezifikationen erfolgt. Selbstverständlich werden oftmals mehrere Formate dazu eingesetzt und es kann sich sehr wohl herausstellen, dass es gar keine einheitliche Testspezifikationsbeschreibung gibt. All dies muss ebenfalls dokumentiert werden. Gut geeignet dazu wären beispielsweise gängige Tabelleneditoren. In dem Ergebnisdokument (Synonym zu Bestandsaufnahmedokument) soll eine Übersicht über eingesetzte Testspezifikationsformate in allen untersuchten Projek-

Abteilung/Testverfahren	MiL	SiL	SiL Open Loop	HiL	Reale Messfahrt
Abteilung A					
Projekt: Staupilot					
Subprojekt: Umfeldwahrnehmung	X	X	Testautomat: .xml, .dbc, .avi, .dat, .ini, ascii	X	Szenarienkatalog
Subprojekt: Situationsanalyse	X	X	Testautomat: .xml, .dbc, .avi, .dat, .ini, ascii	X	Szenarienkatalog
Subprojekt: Sicherheitskonzept	X	Virtual Test Drive: .xml	Testautomat: .xml, .dbc, .avi, .dat, .ini, ascii	X	Szenarienkatalog
Subprojekt: Reglerentwicklung	X	X	Testautomat: .xml, .dbc, .avi, .dat, .ini, ascii	X	Szenarienkatalog
Projekt: ACC					
Subprojekt: Gesamtbewertung	X	Virtual Test Drive: .xml, Matlab: .m Testautomat: .xml	Testautomat: .xml, .dbc, .avi, .dat, .ini, ascii	VTD + EXAM: .xml, UML-Sequenzdiagramme, Python-Skripte	Szenarienkatalog
Abteilung B					
Projekt: PreCrash					
Subprojekt: PreAct	X		Proprietärer Batch-Skript, .dat, .dbc	Exam: UML-Sequenzdiagramme, Python-Skripte	Eigener Fahrszenenkatalog
Subprojekt: Fußgängerschutz	EXACT: .m-Files, Klassifikationsbäume, MS-Word		Proprietärer Python-Skript, .dat, .dbc	EXAM	Eigener Fahrszenenkatalog

Bild 4.3: Beispielhafte Dokumentation der Testspezifikationsformate

ten im ganzen Entwicklungsprozess stehen (Beispiel basierend auf der Domäneneanalyse aus dem Kapitel 2 Abb. 4.3). Ebenfalls muss eine kurze Beschreibung der eingesetzten Testverfahren abgelegt werden.

- Untersuchung der Schnittstellen: Nachdem nun feststeht, welche Formate für Testspezifikationen in den einzelnen Entwicklungsphasen eingesetzt werden, muss untersucht werden, wie aktuell im Prozess die Übergabe der Testspezifikationen von einer Phase in die nächste stattfindet. Beispielsweise wird in der FAS-Domäne mit elektronischen Dokumenten gearbeitet, in denen Fahrmanöver sowie erwartete FUT-Reaktionen festgehalten sind. Möglicherweise können auch komplette Testskripte ausgetauscht werden, falls beidseitig ein und dieselbe Plattform zur automatisierten Ausführung der Testabläufe verwendet

Übergebende Abteilung/Annehmende Abteilung	Abteilung F	Lieferant M
Abteilung A	Anforderungsdokument, Szenarienkatalog	Funktionscode, Lastenheft
Projekt: Staupilot		
Subprojekt: Umfeldwahrnehmung		
Subprojekt: Situationsanalyse		
Subprojekt: Sicherheitskonzept		
Subprojekt: Reglerentwicklung		
Projekt: ACC		
Subprojekt: Gesamtbewertung		
Abteilung B	Anforderungsdokument, Szenarienkatalog	Funktionscode, Lastenheft
Projekt: PreCrash		
Subprojekt: PreAct		
Subprojekt: Fußgängerschutz		

Bild 4.4: Dokumentation der übergebenen Spezifikationen

wird (beispielsweise das auf Matlab aufsetzende kommerzielle Werkzeug namens EXACT [Ext10]). Es kann sich auch herausstellen, dass gar keine Übergabe stattfindet, da sich die eingesetzten Testmethodiken zu stark unterscheiden oder die Anpassung der übergebenen Spezifikationsformate an die in der nachgelagerten Entwicklungsphase verwendeten Notationen für die Testspezifikation zu hoch ist. Das Ergebnis kann in Form einer Matrix erfasst werden, deren Einträge eine Aufzählung der Bestandteile einer Testspezifikation beinhalten, welche übergeben werden (Beispiel basierend auf der Analyse im Kapitel 2, siehe Abb. 4.4).

4.2.2 Wahl der Modellierungsmethode

Die Wahl der Modellierungsmethode unterteilt sich in drei Aktivitäten, welche in der Abbildung 4.5 dargestellt sind.

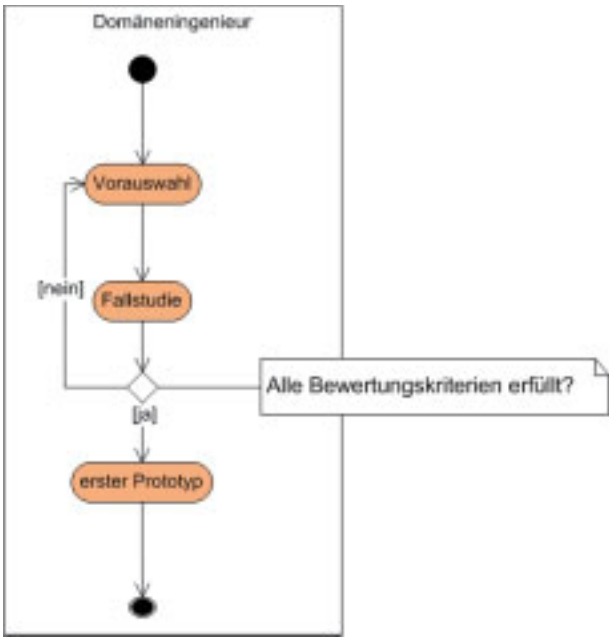


Bild 4.5: Aktivitäten bei der Wahl der Modellierungsmethode

4.2.2.1 Vorauswahl

In diesem Schritt muss entschieden werden, welche Bestandteile der Testspezifikation mit graphischen Formalismen (z.B. Statecharts [Har87], Sequenzdiagramme [ITU04], Aktivitätsdiagramme [HKKR05]) dargestellt werden müssen. Hierbei geht man von den untersuchten Testtechniken aus. Falls beispielsweise das Systemverhalten überprüft wird, kann eine graphische Methode zur dessen Beschreibung gewählt werden. In den frühen Entwicklungsphasen wird oft die Interaktion der per Hand kodierten Module mit der Umwelt getestet. In diesem Fall benötigt man unter Umständen zum Beispiel eine formale graphische Interaktionsbeschreibung des SUT mit der Umwelt. Es geht hier also darum, festzustellen, was eigentlich konkret das Szenario beschreiben soll (Interaktionsverhalten oder Systemverhalten). Ausgehend davon soll bereits hier die Vorauswahl einer geeigneten Modellierungsmethode geschehen. Diese erfolgt anhand der Überprüfung der im Kapitel 3 eingeführten Bewertungskriterien. Die Ergebnisse der Überprüfung sollen abschließend in einem elektronischen Dokument festgehalten werden.

4.2.2.2 Fallstudie

Zur weiteren Überprüfung empfiehlt es sich, konkrete Fallstudien durchzuführen, in denen anhand bereits vorhandener Dokumentation (zum Beispiel die erwähnten Fahrszenenkataloge oder

Videoaufnahmen von den Testfahrten auf einem Prüfgelände) die dort beschriebenen Sachverhalte mit Hilfe der in Frage kommenden formalen Beschreibungsmethode modelliert werden sollen. Es muss erwähnt werden, dass die Fallstudien nicht primär dem Domänenexperten dienen. Sie dienen eher dem Domäneningenieur, welcher dadurch eine präzisere Bewertung der in Frage kommenden Modellierungsmethode nach benannten Kriterien vornehmen kann. Um eine möglichst objektive Bewertung vornehmen zu können, soll ein möglichst breites Spektrum an Anforderungen untersucht werden. Das Ergebnis dieser Untersuchung ist ebenfalls ein elektronisches Dokument, in welchem die formal mit der Methode modellierten Szenarien abgebildet sind. Die Ergebnisse der Fallstudien können in demselben Dokument zusammengefasst werden, in welchem diese der Vorauswahl festgehalten sind. Schließlich muss betont werden, dass, sollten die Ergebnisse der Fallstudie die Anwendbarkeit der untersuchten Modellierungsmethode unmöglich machen, zum vorigen Schritt, also zur Vorauswahl, zurückgekehrt werden muss. Dort sollte eine andere Modellierungsmethode gewählt werden. Die Unmöglichkeit der Anwendbarkeit bedeutet in diesem Schritt, dass mindestens eines der im Kapitel 3 definierten Kriterien nicht erfüllt werden kann.

4.2.2.3 Prototyp

Sobald die Anwendbarkeit der gewählten Methode ebenfalls durch die Fallstudien bestätigt wurde, muss für den nachfolgenden Schritt im Vorgehensmodell eine Implementierungsumgebung und Plattform gewählt werden. Es sei an dieser Stelle ausdrücklich erwähnt, dass es sich in diesem Schritt nicht unbedingt um eine endgültige Auswahl handeln soll. Vielmehr dient die erste Implementierung zur Veranschaulichung der Modellierungsmethode für die Domänenexperten. Hierbei soll eine Art „Spielkasten“ für die Domänenexperten entstehen, mit welchem die Anwender-bezogenen Bewertungskriterien unmittelbar überprüft werden sollen. Ausgehend von dieser Hauptanforderung soll die Auswahl der geeignten Technologien und Werkzeuge für die Implementierung erfolgen. Die Implementierung soll bei weitem nicht komplett sein, so kann ohne weiteres beispielsweise die Anbindung an ein Backend-System ausgelassen werden, da sie meistens sehr viel Zeit in Anspruch nimmt. Ein Backend-System dient der zentralisierten Verwaltung von Testspezifikationen. Die meisten Beschreibungselemente, wie beispielsweise Attribute, können ebenfalls festkodiert werden. Gleichzeitig müssen solche Technologien gewählt werden, bei denen ein Domänenexperte selbständig eine oder mehrere Szenarienbeschreibungen vornehmen, sie ändern, abspeichern und wieder laden kann. Die üblichen Design-Werkzeuge wie beispielsweise Enterprise Architect [Spa] sind in diesem Fall schlecht geeignet, da mit ihnen zwar ein Oberflächendesign möglich ist, aber es entsteht damit keine effektive Anwendung.

4.2.3 Methodenbewertung

Die Methodenbewertung dient dazu, die wissenschaftliche Untersuchung seitens des Domäneningenieurs durch unmittelbare Anwendung durch Domänenexperten zu bestätigen oder zu widerlegen. Zudem können Schwächen herausgestellt werden, welche iterativ behoben werden können. Es muss in der Vorbereitung dieser Bewertung zunächst die Form der Befragung festgelegt werden. Die wohl bekannteste Methode ist Fragebogen-Befragung. Im industriellen Umfeld kann diese jedoch schnell scheitern, da die Entwickler oft unter extremem Zeitdruck stehen und deshalb einen zusätzlichen „Papieraufwand“ meistens scheuen. Wesentlich sinnvoller sind ein oder mehrere Workshops, bei welchen zwei bis drei Entwickler mehrere in Worten formulierte Szenarienbeschreibungen bekommen, welche sie mit dem entwickelten Prototyp dann nachmodellieren müssen. Die textbasierten Szenarienbeschreibungen sollen idealerweise aus den den Entwicklern bekannten Quellen kommen, beispielsweise aus den elektronischen Dokumenten mit den Fahrszenen-Beschreibungen. Es ist sehr wichtig, dass die Domänenexperten nicht nur am Ende des Workshops ein Feedback geben und ihre Verbesserungsvorschläge äußern, sondern dass der Domäneningenieur die Domänenexperten beim Modellieren genauestens beobachtet, denn der Domänenexperte kann sich nicht alle Modellfehler merken oder er empfindet sie nicht als solche. Deshalb ist eine direkte Beobachtung unabdingbar. Die Beurteilung muss sich nach folgenden Kriterien richten:

- Gravierende Modellfehler: Diese entstehen dann, wenn die Entwickler nicht imstande sind, bestimmte Sachverhalte zu modellieren, weil die gewählte Methode dies nicht zulässt. Beispielsweise lässt die Methode die Modellierung von zeitlichen Abhängigkeiten nicht zu. Diese Art von Fehlern ist ein Indiz dafür, dass die Analyse im vorhergehenden Schritt unpräzise war und dass noch eine Iteration notwendig ist.

- Ambiguitäten: Diese entstehen dann, wenn die Methode zu mächtig ist und einfach „zu viele Freiheiten“ bei der Modellierung bietet. „Freiheit“ ist in diesem Zusammenhang die Möglichkeit, einen und denselben Sachverhalt mit mindestens zwei Elementen der Modellierungsmethode darzustellen. So kann man beispielsweise den Sachverhalt „Regenbeginn“ unter Anwendung von Statechart sowohl als Bedingung als auch Ereignis modellieren. Diese können auf folgende Art vermieden werden: Man schränkt die gewählte Modellierungsmethode ein (siehe Abschnitt 4.3). Dazu ist eine weitere Iteration im Vorgehensmodell notwendig (siehe Abbildung 4.2), um die Eindeutigkeit der Modellierung zu erreichen. Denn oftmals wird man als Domäneningenieur erst nach einem Feedback von den Entwicklern auf solche Ambiguitäten aufmerksam. Die Ambiguitäten sind in einer spezifischen

Domäne insofern schädlich, als sie das Erlernen und das Verständnis der Methode seitens der Entwickler wesentlich erschweren. Somit sinkt auch deren Akzeptanz.

- Unklare Bezeichner: Unklare Bezeichner von Modellierungselementen sollen ebenfalls vom Domäneningenieur vermerkt werden, um sie in den weiteren Schritten des Vorgehensmodells zu berücksichtigen. Sie führen aber nicht unmittelbar zur nächsten Iteration.

- Unvollständigkeit der Beschreibung: Die Unvollständigkeit der Beschreibung ergibt sich durch menschliche Eigenschaften wie Konzentrationsmangel oder Ablenkung durch äußere Faktoren. Die Unvollständigkeit lässt sich durch die gewählte Methode allein nicht beheben. Dazu muss die entsprechende Werkzeugunterstützung vorhanden sein, welche die erzeugte Szenarienbeschreibung automatisch auf Vollständigkeit überprüft. Ein Beispiel für die Unvollständigkeit wären mangelnde Transitionen in einer Statechart-basierten Fahrszenenbeschreibung. Man kann also in diesem Kontext von einer Assistenzfunktion sprechen, welche die Domänenexperten zur Laufzeit bei der Szenarienmodellierung unterstützt.

- Allgemeines Feedback vom Domänenexperten: Bei diesem geht es darum, herauszustellen, ob die Entwickler mit der Methodik im Allgemeinen zurechtkommen und was außer den benannten Kriterien noch berücksichtigt werden muss.

Zum Schluss ist anzumerken, dass, sollten bei der Methodenbewertung vor allem gravierende Modellierungsfehler festgestellt worden sein, muss gemäß dem Vorgehensmodell (siehe Abbildung 4.2) ein Rücksprung zum Schritt „Wahl der Modellierungsmethode“ stattfinden.

4.3 Definition eines geeigneten Metamodells

Die in dieser Arbeit verwendete Bezeichnung für verschiedene Metamodellierungsebenen orientiert sich an der von OMG definierten Meta Object Facility (MOF). Gleichzeitig sei an dieser Stelle ausdrücklich betont, dass zur Beschreibung von Metamodellen nicht die von OMG definierten Beschreibungssprachen herangezogen werden. Dort wird zwischen 4 Ebenen unterschieden (Abb.4.6). Für den hier vorgestellten Ansatz werden die Ebenen M0 bis M2 benötigt. Die Definition eines Metamodells findet demnach auf dem Niveau M2 statt. Zur weiteren Einarbeitung in das Gebiet der Metamodellierung sei an dieser Stelle auf [Mü07] [GH08] [GDD09] verwiesen. Ein Metamodell bietet eine Erleichterung bei der Umsetzung der gewählten Modellierungsmethode, denn durch dieses wird formal festgelegt, woraus die Modellierungsmethode bestehen darf. Dadurch wird vor allem die Transformation einer so definierten Testspezifikation erleichtert.

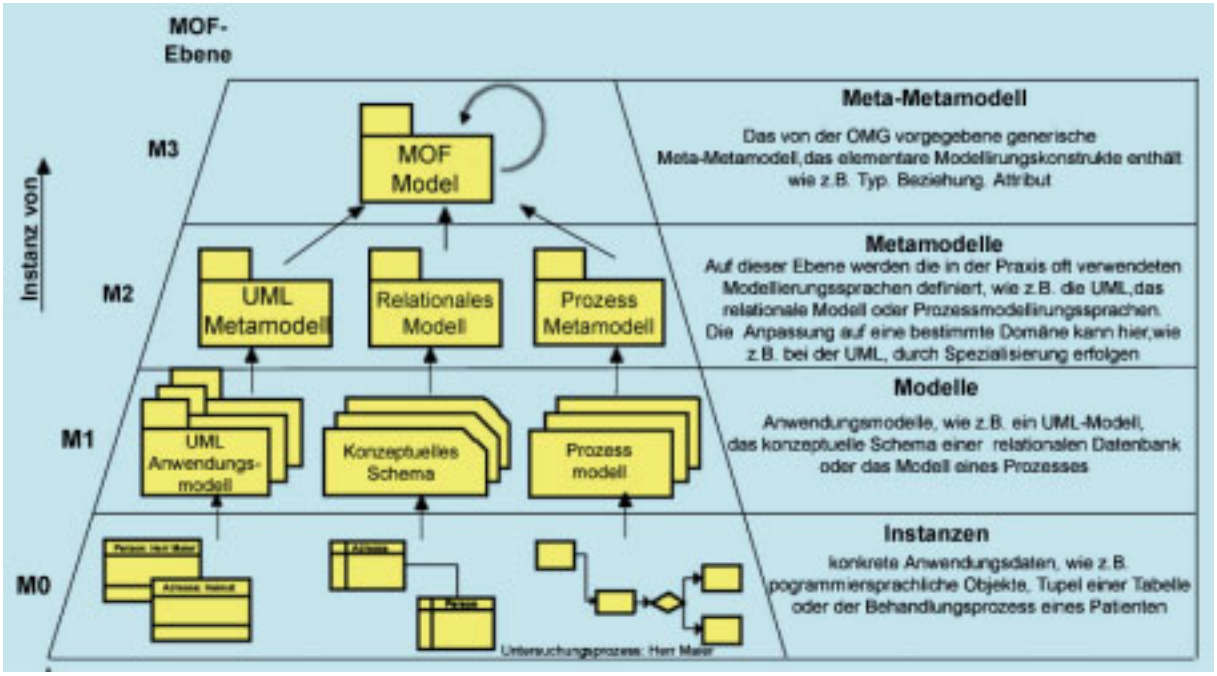


Bild 4.6: Meta Object Facility [Pet06]

Letztendlich wird damit ein wesentliches in dieser Arbeit definiertes Kriterium der Wiederverwendbarkeit für die Testspezifikation erfüllt.

Grundsätzliche Aktivitäten, die bei der Definition eines geeigneten Metamodells stattfinden, sind in der Abbildung 4.7 dargestellt. Die in diesem Ansatz verwendete Modellierungshierarchie ist in der Abbildung 4.8 aufgezeigt. Das Basis-Metamodell wird bei der Anwendung dieses Ansatzes als gegeben vorausgesetzt. Dieses ist unveränderbar und unabhängig von einer konkreten Domäne. Vom Basis-Metamodell wird vom Domäneningenieur das so genannte Domänen-Metamodell abgeleitet. Wie schon der Name sagt, ist dieses domänenabhängig und definiert Elemente der domänenspezifischen Szenarienmodellierungsmethode. Auf M1-Ebene wird vom Domäneningenieur das partielle Modell definiert (siehe Abschnitt 4.4). Das partielle Modell beinhaltet alle relevanten Beschreibungselemente der Modellierungsmethode, mit welchen der Domänenexperte eine Testspezifikation beschreiben kann. Schließlich definiert der Domänenexperte sowohl das komplette Modell einer Testspezifikation als auch eine konkrete Instanz davon. Dabei bedient er sich des vom Domäneningenieur im vorigen Schritt erzeugten partiellen Modells. Dies geschieht auf den Ebenen M1 beziehungsweise M0. Da das Basis-Metamodell als gegeben gilt, ist dessen Definition kein Bestandteil des Vorgehensmodells. Dieses muss jedoch präzise definiert werden, was im Folgenden geschieht.

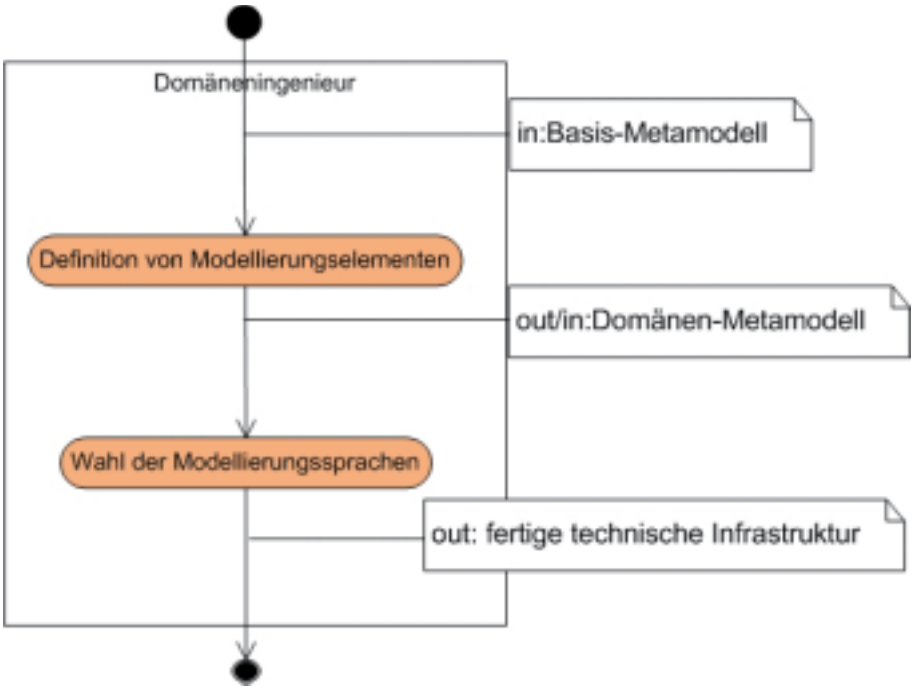


Bild 4.7: Definition eines geeigneten Metamodells

Wichtige Modellierungskonzepte des Basis-Metamodells

- Für jedes Beschreibungselement (Synonym zu Modellierungselement) der Szenarienmodellierungsmethode existiert eine Kategorisierung. Jede Kategorie kann Unterkategorien besitzen (siehe Abbildung 4.9). Die Modellierungselemente dürfen nur auf die Kinderknoten des Kategorisierungsbaums verweisen. Jede Kategorie besitzt selbstverständlich einen eindeutigen Namen.

- Jedes Beschreibungselement der gewählten Methode zur Definition von Testspezifikationen wird durch generische, komposite und atomare Attribute beschrieben wie in der Abbildung 4.10 gezeigt. Jedes atomare Attribut kann einen Wertebereich sowie eine (Mess)Einheit besitzen. Auf der M2-Ebene wird jedes Attribut außerdem durch einen eindeutigen Namen beschrieben.

- Erweiterungen (engl. Extension): Projektspezifische atomare und komposite Variablen werden in den so genannten Erweiterungen zusammengefasst (Abbildung 4.11). Selbstverständlich müssen projektspezifische Variablen einen eindeutigen Namen besitzen sowie möglicherweise einen Wertebereich und eine Messeinheit. Jedes Beschreibungselement

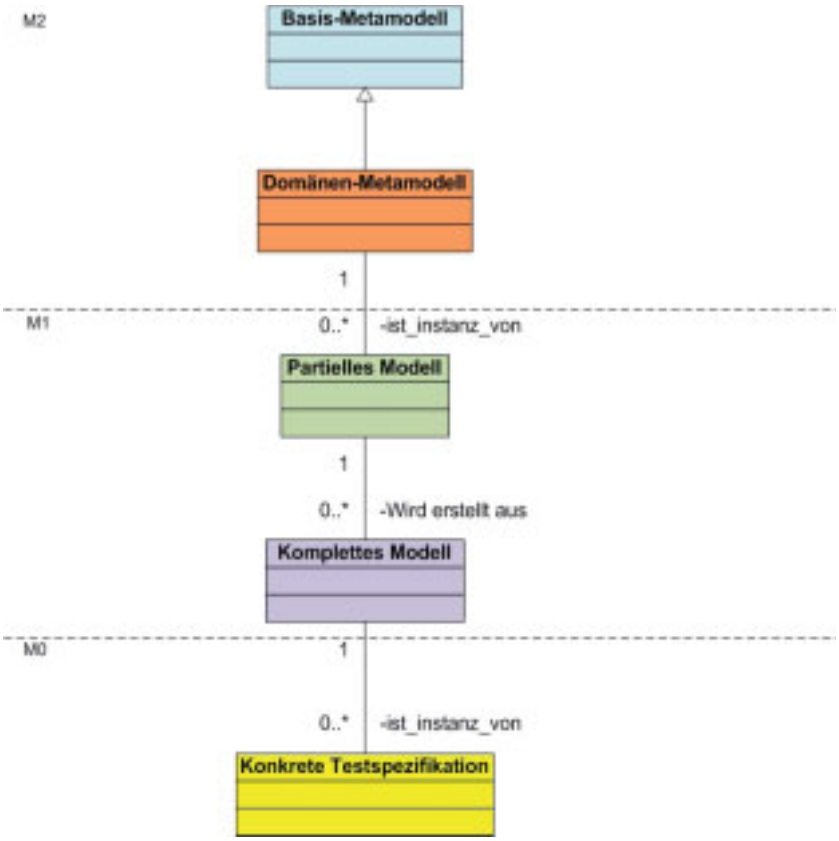


Bild 4.8: Die verwendete Modellierungshierarchie

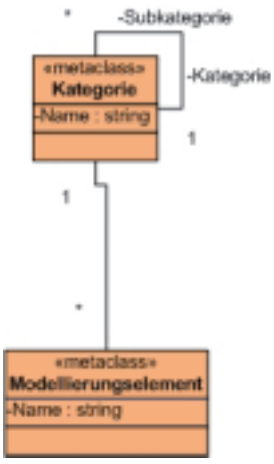


Bild 4.9: Kategorisierung von Beschreibungselementen

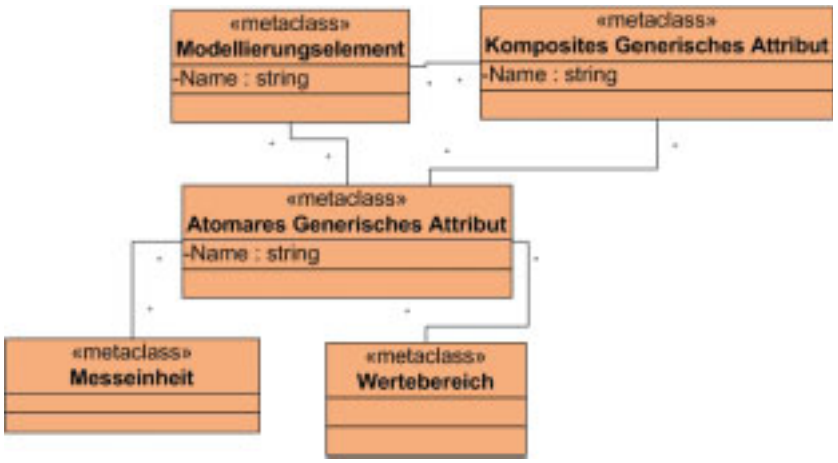


Bild 4.10: Generische Attribute eines Beschreibungselements

besitzt eine oder mehrere projektspezifische Erweiterungen im Rahmen eines bestimmten Projekts, eines Subprojekts und einer Testmethode. Die im Kapitel 1 ermittelte Abteilungs- und Projektstruktur wird durch das Metamodellelement „Extension Category“ zum Ausdruck gebracht. Durch einen terziären Beziehungstyp zwischen „Extension Category“, „Modellierungselement“ und „Extension“ ist es jederzeit möglich, die Zuordnung einer bestimmten Erweiterung einem bestimmten Beschreibungselement im Rahmen eines bestimmten Projekts, Subprojekts oder der Testmethode vorzunehmen.

Im weiteren werden nun die Aktivitäten näher betrachtet, welche bei der Definition eines geeigneten Domänen-Metamodells durchgeführt werden müssen (Abb. 4.7).

4.3.1 Definition von Modellierungselementen

Was die Elementtypen betrifft, welche ein Domänen-Metamodell beinhalten muss, so sind es jene, die im Kapitel 3 definiert worden sind: „Testdaten“, Testfall“, „Referenzdaten“, „Transformationdaten“, „SUT-Verhaltens- und/oder Interaktionsbeschreibung“, „Setup-Daten“, „SUT-Daten“. Abhängig vom gewählten Testverfahren muss das Domänen-Metamodell natürlich um methodenspezifische Modellierungselemente erweitert werden wie beispielsweise im Fall eines Statecharts „Ereignis“, „Transition“ oder „Zustand“. Zusätzlich dazu muss es möglich sein, Testfälle zu Testläufen zu gruppieren, also muss ein Element „Testlauf“ eingeführt werden. Außerdem muss zu einer sinnvollen Verwaltung nachvollziehbar sein, welcher Domänenexperte welche Testspezifikation zu welchem Zeitpunkt angelegt hat.

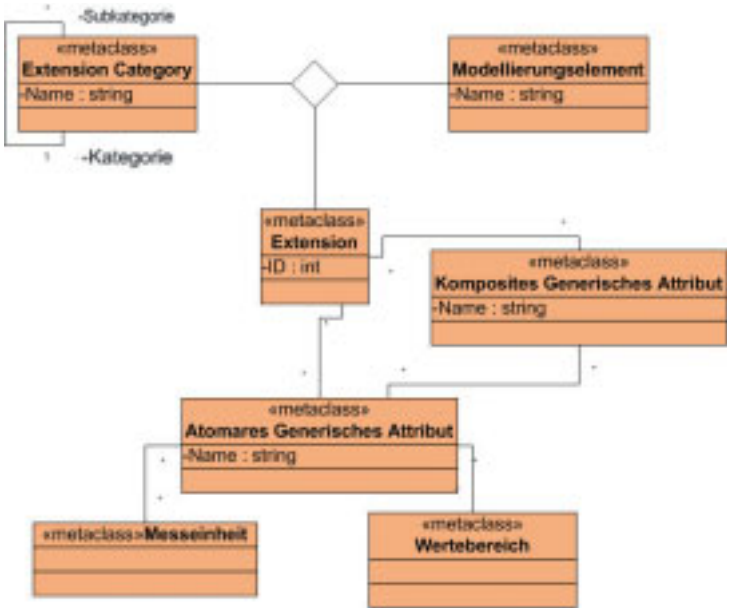


Bild 4.11: Projektspezifische Erweiterungen eines Beschreibungselements

Bei der Aktivität „Definition von Modellierungselementen“ müssen vom Domainingenieur zwei grundsätzliche Aspekte berücksichtigt werden. Zum einen dient ein Domänen-Metamodell bei der Definition einer Testspezifikation dazu, ein Wörterbuchkonzept zu realisieren. In diesem Wörterbuch sind kategorisiert alle möglichen Begriffe dargestellt, welche bei der Definition von der Testspezifikation eine Rolle spielen können. Dies bedeutet, dass ein Domänen-Metamodell in erster Linie eine kategorisierte Darstellung von Beschreibungselementen ermöglichen muss. Gerade dieser Aspekt wurde von keiner der in dieser Arbeit untersuchten Methoden zur Definition von Testspezifikationen in Betracht gezogen. Das Wörterbuchkonzept bietet folgende Vorteile:

- Strukturiertheit der Darstellung: Jeder Begriff wird einer Kategorie zugeordnet und ist unter dieser für den Fachexperten (Synonym zu Domänenexperte) zu finden. Dies kann unter Umständen die Definition einer Testspezifikation beschleunigen.

- Verwendung domänenspezifischer Begriffe: Beschreibungselemente können domänenspezifische Namen erhalten. Somit steigt der Verständlichkeitsgrad der Methode und folglich auch ihre Akzeptanz. Außerdem können auf diese Art und Weise erstellte Testspezifikationen abteilungs- und projektübergreifend als Kommunikationsgrundlage innerhalb einer Domäne verwendet werden.

- Formalisierung durch das zugrunde liegende Metamodell: Die Formalisierung ermöglicht eine automatisierte Abbildung der erstellten Testspezifikationen auf bestimmte Zielformate.

- Wiederverwendbarkeit: Die Erfahrung zeigt, dass Testspezifikationen oftmals in solchen Bestandteilen wie Interaktions- oder Verhaltensbeschreibung ähnliche Struktur aufweisen. So wurde beispielsweise beobachtet, dass bei Fahrszenenbeschreibungen in der FAS-Domäne im HiL-Bereich immer wieder dieselben Manöverabfolgen getestet werden, welche sich lediglich in einzelnen Attributbelegungen unterscheiden. Dies führte dann zu sehr umfangreichen semi-strukturierten Testspezifikationen, in welchen die Testfälle sehr geringe Unterschiede aufwiesen. Solche Redundanzen lassen sich beim Wörterbuch-Konzept vermeiden, denn ein auf der Modell-Ebene (M1) definiertes Szenario kann mehrfach mit unterschiedlichen Wertebelegungen für Attribute instanziiert werden. Somit geht der hier vorgestellte Ansatz einen Schritt weiter als Model-Driven Architecture, denn er strebt eine formale Darstellung von CIM an, indem Text-basierte, semi-strukturierte Testspezifikationsbeschreibungen durch „Wörterbuch-basierte“, also formale auf einem Metamodell beruhende, ersetzt werden. Dadurch werden automatisierte Transformationen zwischen CIM und PIM möglich, während bei MDA diese noch per Hand gemacht werden müssen.

Der zweite wichtige Aspekt betrifft die Konzepte, welche in dem Wörterbuch verwaltet werden. Jeder dieser Begriffe muss durch Attribute beschreibbar sein. Nun, dies ist nichts Neues, denn betrachtet man beispielsweise gängige OMG-Metamodelle für die UML-Diagramme, so erlauben viele von ihnen die Definition von Attributen für die jeweiligen Modellierungselemente. Doch zur Erhöhung des Wiederverwendbarkeitsgrades einer Testspezifikation ist dies noch nicht ausreichend, denn es muss schon auf der Metamodell-Ebene eine Unterscheidung zwischen generischen und projektspezifischen Attributen getroffen werden. Genau das ist ein weiterer wichtiger Schritt in Richtung Wiederverwendbarkeit. Denn nur so werden die jeweiligen Konzeptanwender, also ein Domäneningenieur, gezwungen, sich bereits bei der Modelldefinition Gedanken darüber zu machen, welche Beschreibungsattribute der jeweiligen Modellierungselemente generisch und welche projektspezifisch sind. Darüberhinaus gilt es zu überlegen, ob und welche Modellierungselemente noch weiterer Verfeinerung bedürfen. Beispielsweise kann man die Zustände bei einer Fahrszenenbeschreibung weiterverfeinern, um damit zum Ausdruck zu bringen, dass bei bestimmten SUT-Zuständen wie beispielsweise „Das EGO-Fahrzeug fährt mit einer konstanten Geschwindigkeit“ keine Test-relevanten Aktivitäten stattfinden, während beispielsweise bei „Fahrspurwechsel“ es sich um eine Aktivität handelt, welche für die Überprüfung des jeweiligen SUT von Bedeutung ist.

Interessant ist in diesem Zusammenhang, anhand des gerade erwähnten Beispiels die einzelnen Bestandteile des Wörterbuchs zu erklären. Demnach muss für die zwei verschiedenen

Arten von Zuständen jeweils eine Kategorie angelegt werden. Für die Initial- und End-Zustände könnte man den Namen „quasistatischen Zustände“ vergeben, während für die Test-relevanten die Benennung „Aktionszustände“ am besten passt. Nun würde man folgende Beschreibungselemente der Kategorie „Aktionszustände“ zuordnen: „Fahrzeug schert aus“, „Fahrzeug beschleunigt“, „Fahrzeug bremst ab“. Weiterhin kann man den Begriff „Fahrzeug schert aus“ durch solche generischen Attribute beschreiben wie „Aktuelle Fahrspur“ und „Zielfahrspur“. Als ein projektspezifisches Attribut würde dann „Querbeschleunigung“ dienen, welche beispielsweise für die Funktion „Spurwechsel-Assistent“ von Bedeutung ist.

Im Folgenden wird nun auf die zweite wichtige Aktivität bei der Definition eines geeigneten Metamodells eingegangen: „Wahl der Modellierungssprachen“ (siehe Abb. 4.7).

4.3.2 Wahl der Modellierungssprachen

Bei der Wahl der Modellierungssprachen sind drei wichtige Aspekte zu berücksichtigen:

- Graphische Darstellung: Die graphische Sprache muss nicht den im Kapitel 2 definierten Anwender-bezogenen Kriterien entsprechen. Denn es geht hier um die Darstellung des Domänen-Metamodells, welche durch den Domäneningenieur vorgenommen wird. Der Domänenexperte arbeitet nicht umittelbar mit den Metamodellen. Die graphische Sprache dient deshalb in diesem Kontext zu Design-Zwecken und letztendlich zur Veranschaulichung für den Domäneningenieur. Gleichwohl muss eine Metamodellierungssprache vollständig sein und alle notwendigen Modellierungskonstrukte zur Verfügung stellen, mit denen ein solches Domänen-Metamodell entworfen werden kann.

- Operationalisierbarkeit: Diese impliziert, dass die mit der gewählten Methode erstellten Metamodelle transformierbar sind. Dazu müssen von diesen Metamodellen Instanzen gebildet werden können (M1-Modelle). Die Transformationen werden meist mit den dedizierten regelbasierten Sprachen durchgeführt.

- Verwaltungsaspekt: Die mit einer Modellierungssprache erstellten Metamodelle (M2), Modelle (M1) und ihre Instanzen (M0) müssen zentralisiert so verwaltet werden, dass mehrere Fachexperten auf sie zugreifen können, indem sie beispielsweise nach den bereits bestehenden Testspezifikationen suchen, diese ergänzen oder modifizieren können.

Bei genauer Betrachtung besteht zwischen den benannten Aspekten ein Konflikt. Betrachtet man beispielsweise graphische Sprachen zur Darstellung von Metamodellen, so sind diese vollständig

und es lassen sich damit Metamodelle gut entwerfen. Operationalisierbarkeit ist aber ohne weiteres nicht gegeben. Das heißt, es muss meistens eine XMI [Obj] Repräsentation der graphischen Sprache erstellt werden, um diese zu gewährleisten. Genauso ist beispielsweise Structured Query Language (SQL) [Bea09] Data Definition Language (DDL) ideal dazu geeignet, die komplette Modellierungshierarchie, also Metamodell, Modell sowie Instanzen einer Testspezifikation, zentral für verschiedene Projekte, Subprojekte und Testtechniken zu verwalten, sodass die Mehrbenutzerfähigkeit gewährleistet ist. Allerdings ist die in diesem Kontext definierte Operationalisierbarkeit nicht gegeben, denn solche Anfragesprachen wie SQL sind nicht dazu geeignet Transformationen von Testspezifikationen durchzuführen. Sie sind zwar selbst transformierbar, können aber keine Transformationen beschreiben. Es wird also ersichtlich, dass diverse Sprachen in Bezug auf die erwähnten Aspekte unterschiedliche Stärken, aber auch Schwächen aufweisen. Es ist also sinnvoll, jede der Sprachen dort einzusetzen, wo sie ihre Stärken aufweist und eine Art „Brücke“ zwischen diesen zu schaffen, wenn es um die praktische Verwendung geht. Um nicht unnötige Begriffe einzuführen (z.B. „Brücke“) wird ein bereits in der Literatur bestehender Begriff verwendet: „Modellierungsraum“(engl. „Modelling Space“) [GDD09]. Ein Modellierungsraum (MS) ist „a modelling architecture based on a particular super-metamodel.“[GDD09] Beispiele für zwei Modellierungsräume sind in der Abbildung 4.12 zu sehen. Zum einen wird dort der so genannte MOF-Modellierungsraum dargestellt, welcher auf der M3-Ebene auf MOF basiert. MOF, wieder-

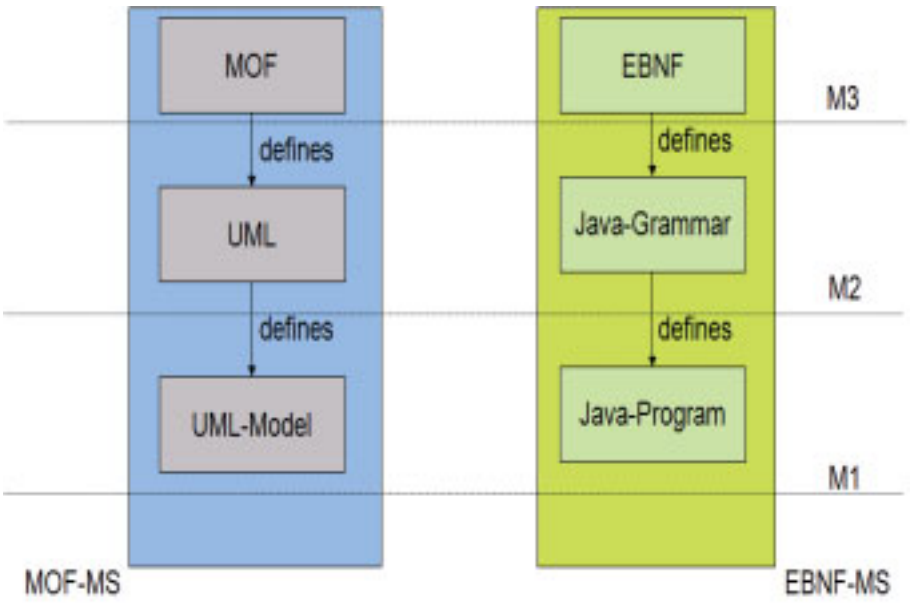


Bild 4.12: Modellierungsraum: Ein Beispiel [GDD09]

um, definiert UML und die mit UML erzeugten UML-Notationen sind dann auf der M1-Ebene angesiedelt. Zum anderen wird ein EBNF (Erweiterte Backus-Naur-Form) Modellierungsraum dargestellt (M3). Mit diesem ist es möglich, Grammatiken zu definieren, wie beispielsweise diese für die Programmiersprache Java [Sun] (M2). Auf der M1-Ebene können dann Java-Programme geschrieben werden, die durch die Java-Grammatik definiert sind. [GDD09] unterscheidet darüberhinaus zwischen konzeptuellen und technischen Modellierungsräumen (MS): Konzeptuelle MS dienen, wie der Name sagt, zur Modellierung von Konzepten. Zu diesen gehören beispielsweise MOF-MS oder Ecore-MS [Ecl]. Andererseits dienen die technischen Modellierungsräume zur maschinellen beziehungsweise serialisierten Darstellung konzeptueller Modellierungsräume. Schließlich muss noch eine wichtige Unterscheidung bei der Definition von Modellierungsräumen getroffen werden: Parallele MS und Orthogonale MS. Parallele MS repräsentieren dieselben Objekte der realen Welt, es werden lediglich unterschiedliche Notationen verwedet. Beispielsweise kann eine Kategorie bei Entity-Relationship-Modellierung [EN02] als ein Entitätstyp definiert werden, während in der Ecore-Modellierung ein „EClass“ angebracht ist. Bei orthogonalen Modellierungsräumen modelliert ein MS Konzepte aus dem anderen MS, indem sie als Objekte der realen Welt betrachtet werden. Beispielsweise wird in EBNF-MS die Java-Grammatik auf der M2-Ebene als Instanz von EBNF definiert. Dahingegen wird sie im MOF-MS auf der M0-Ebene angesiedelt, da sie eine Instanz (M0) des mit der UML dargestellten Java-Grammatik-Modells (M1) ist.

Zum besseren Konzeptverständnis ist in der Abbildung 4.13 eine mögliche Darstellung des Domänen-Metamodells in unterschiedlichen Modellierungsräumen aufgezeigt. Demnach kann das Basis-Metamodell in Form von UML in einem elektronischen Dokument dokumentiert werden. Bei der Definition des Domänen-Metamodells mittels Entity-Relationship-Diagramme (E/R) muss das Basis-Metamodell in E/R übertragen werden. Zu Dokumentationszwecken kann das Domänen-Metamodell ebenfalls in einem elektronischen Dokument festgehalten werden. Möchte man darüberhinaus eine maschinenlesbare Notation des Domänen-Metamodells erzeugen, so geschieht dies entweder per Hand oder mit Hilfe eines dedizierten Tools. Dazu muss das E/R-Diagramm selbstverständlich in diesem Tool zunächst angelegt werden. Die Transformation in SQL-DDL geschieht automatisch. Auf der anderen Seite kann ein Domänen-Modell einer Testspezifikation in ECore spezifiziert werden. Das Ergebnis ist ebenfalls ein elektronisches Dokument. Zur Transformation in eine maschinenlesbare Notation muss das Domänen-Metamodell in ECore mit Hilfe eines speziellen Editors von Eclipse Modelling Framework angelegt werden. Die Erzeugung einer JAVA-Bibliothek, welche zur Instanziierung des Metamodells und zur Serialisierung der Instanzen des Metamodells notwendig ist, geschieht automatisch. Schließlich

muss möglicherweise ein proprietäres Werkzeug geschrieben werden, welches die konkreten (M0) Testspezifikationen aus SQL-DDL in EMF umwandelt, damit in einem nächsten Schritt die Transformation in ein bestimmtes Zielformat stattfinden kann. Somit ergibt sich folgende

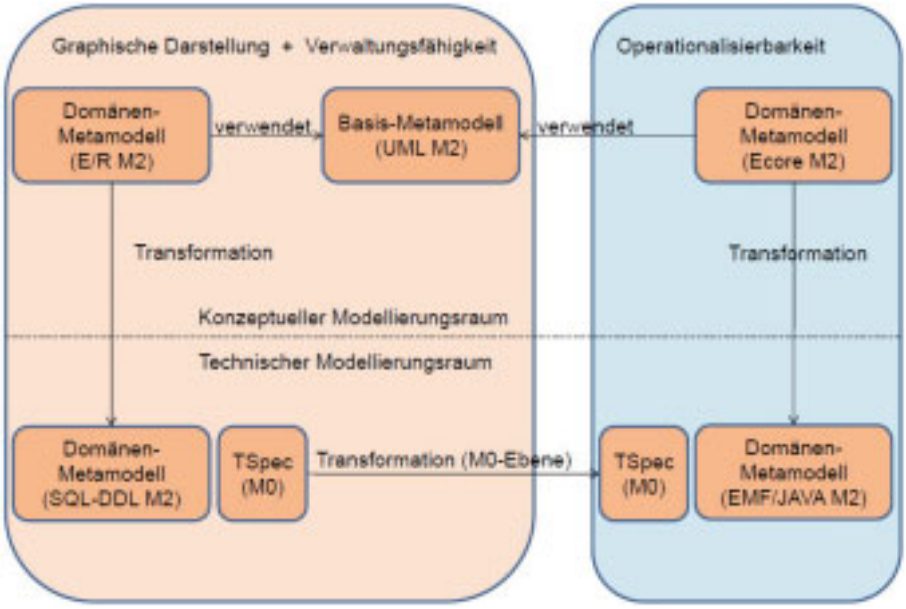


Bild 4.13: Mögliche Darstellung des Domänen-Metamodells in unterschiedlichen Modellierungsräumen

Vorgehensweise bei der Wahl der Modellierungssprachen:

- Wahl eines konzeptuellen Modellierungsraums zur graphischen Darstellung des Domänen-Metamodells: Hierbei spielt die Wahl des technischen MS noch eine geringe Rolle, da, wie bereits erwähnt, die graphischen Sprachen lediglich zu Designzwecken vom Domäneningenieur verwendet werden. Es könnte zum Beispiel ein gängiges UML-Modellierungswerkzeug eingesetzt werden, welches die erzeugten Domänen-Metamodell-Entwürfe im XMI-Format abspeichert.

- Wahl eines konzeptuellen Modellierungsraums zur Erzeugung „verwaltungsfähiger“ Modellierungshierarchien. Hierbei muss das im ersten Schritt erzeugte Domänen-Metamodell in der gewählten MS-Sprache um verwaltungstechnische Aspekte ergänzt werden. Diese könnten im Fall von Testspezifikationen sein: Die Benutzerverwaltung, Organisation von Testläufen, Verwaltung von Transformationsvorschriften für die Testspezifikationen etc. Sicherlich stellt sich sofort die Frage, könnte man denn nur einen Modellierungsraum für

die Berücksichtigung des graphischen und des Verwaltungsaspekts wählen? Es ist sicherlich möglich, allerdings hängt die Wahl sehr stark von methodologischen Kenntnissen des jeweiligen Domäneningenieurs sowie von der im gesamten Prozess eingesetzten Tool-Landschaft ab. Wird beispielsweise kein UML-Designwerkzeug eingesetzt, sondern nur eines zur E/R-Modellierung, dann kann der Domäneningenieur das Domänen-Metamodell unter Berücksichtigung beider Aspekte sofort in E/R entwerfen. Ist ein Domäneningenieur allerdings mit E/R nicht vertraut, so kann er beispielsweise Klassendiagramme von UML oder die Ecore Notation verwenden. Diese kann dann per Hand von einem anderen Domäneningenieur in die E/R-Notation übertragen werden.

- Wahl des technischen Modellierungsraums zur Erzeugung „verwaltungsfähiger“ Modellierungshierarchien: In den meisten Fällen handelt es sich bei diesem Schritt um eine Datenbank. Es muss also eine Entscheidung getroffen werden, um welche Art von Datenbank-Management-System es sich handeln soll: Relational, objektorientiert, objekt-relational oder eine XML-Datenbank. Der Übergang zwischen dem konzeptuellen und dem technischen Modellierungsraum ist in diesem Fall stark abhängig von den eingesetzten Werkzeugen: Viele E/R-Modellierungswerkzeuge [Toa] ermöglichen inzwischen einen automatisierten Export beispielsweise in das SQL DDL Format. Notfalls muss aber die Übertragung in den technischen MS manuell erfolgen.

- Wahl eines konzeptuellen Modellierungsraums für die Erzeugung transformationsfähiger Modellierungshierarchien (Operationalisierbarkeit): Wenn die ersten zwei Aspekte sich theoretisch noch in einem konzeptuellen Modellierungsraum berücksichtigen ließen, so ist für die Berücksichtigung der Operationalisierbarkeit definitiv ein eigener Modellierungsraum notwendig. Das liegt daran, dass das erzeugte Metamodell domänenspezifisch ist (Synonym zu Domänen-Metamodell). Außerdem verwendet es zusätzliche im Basis-Metamodell festgelegte Konstrukte, wodurch die Definition einer Testspezifikation in gängigen Sprachen wie UML und somit auch deren Transformation unmöglich sind. In der Praxis erfolgt die Definition eines Domänen-Metamodells per Hand ausgehend von der bereits erzeugten graphischen Darstellung.

- Wahl eines technischen Modellierungsraums zur Erzeugung transformationsfähiger Modellierungshierarchien: Oftmals sind konzeptuelle MS auf diesem Gebiet auch an einen bestimmten technischen Modellierungsraum gebunden, sodass die Auswahl sich eigentlich erübrigt. So bietet zum Beispiel Ecore ein eigenes XML-Format zur Definition von Modellinstanzen (Eclipse Modelling Framework).

Letztendlich entsteht die Frage, welcher Zusammenhang zwischen den gewählten technischen Modellierungsräumen besteht. Denn, wie bereits erwähnt, die einzelnen Modellierungsräume können nur einzelne Aspekte wie Operationalisierbarkeit oder Verwaltung sinnvoll berücksichtigen. Wie kommt man von einem technischen Modellierungsraum zum anderen, falls man beispielsweise nach der SQL-basierten Suche einer bestimmten Testspezifikation (M0) diese in eine bestimmte Zielnotation transformieren möchte? Dazu ist es notwendig Transformationstools zu definieren, welche die Testspezifikation aus dem Quell-MS in eine des Ziel-MS umwandeln. Nachdem nun die Wahl der Modellierungsräume erfolgt ist und die erste stabile Version der Implementierung vorliegt, muss die partielle Instanziierung des angelegten Metamodells erfolgen.

4.4 Partielle Instanziierung des Metamodells

Zunächst muss geklärt werden, warum die Instanziierung als „partiell“ bezeichnet wird. Partiell wird sie deshalb genannt, weil dadurch noch keine M1-Modelle der Testspezifikationen entstehen, sondern „nur“ das besagte Wörterbuch angelegt wird, aus welchem dann ein Domänenexperte ein Modell der Testspezifikation anlegt, um es im nächsten Schritt zu instanziieren (auf der M0-Ebene). Die eigentliche partielle Instanziierung im System geschieht durch den Domänenexperten, während die Ermittlung der Beschreibungselemente sowie zugehöriger generischer und projektspezifischer Attribute in Zusammenarbeit zwischen diesem und dem Domäneningenieur geschieht. Zum besseren Verständnis ist der Zusammenhang zwischen Metamodell und partiellem Modell in der Abbildung 4.14 gezeigt. Auf der M2-Ebene (Abbildung 4.14, Step 1) ist beispielshaft eine kategorisierte Darstellung für Zustände des Eigenfahrzeugs aus der FAS-Domäne dargestellt. Dort unterscheidet man zwischen quasistatischen und Aktionszuständen. Ein quasistatischer Zustand, wie bereits erwähnt, ist derjenige, bei dem keine testrelevanten Aktionen geschehen wie zum Beispiel konstante Fahrt oder der Stillstand. Aktionszustände sind testrelevant: Abbremsen, Beschleunigen, Einscheren. Auf der M1-Ebene kann folgende partielle Instanz des M2-Metamodells erzeugt werden: Abbildung 4.14, Step 2. Man beachte, dass nicht nur die M2-Zustände selbst instanziiert werden, sondern auch die jeweiligen Kategorien und Subkategorien. Selbstverständlich geschieht hier auch die Zuordnung der M2-Zustandsinstanzen den jeweiligen M1-Kategorien oder M1-Subkategorien. Im Step 3 geschieht die eigentliche Modell-Erzeugung, indem der Entwickler die einzelnen Begriffe aus dem Wörterbuch miteinander verbindet, sodass daraus ein Szenario wird: Zunächst steht das Fahrzeug still, dann beschleunigt es und fährt anschließend mit einer konstanten Geschwindigkeit. Danach muss der Entwickler die Werte für die Attribute vergeben, durch welche die einzelnen Begriffe beschrieben sind. Somit entsteht eine M0-Instanz der Test-

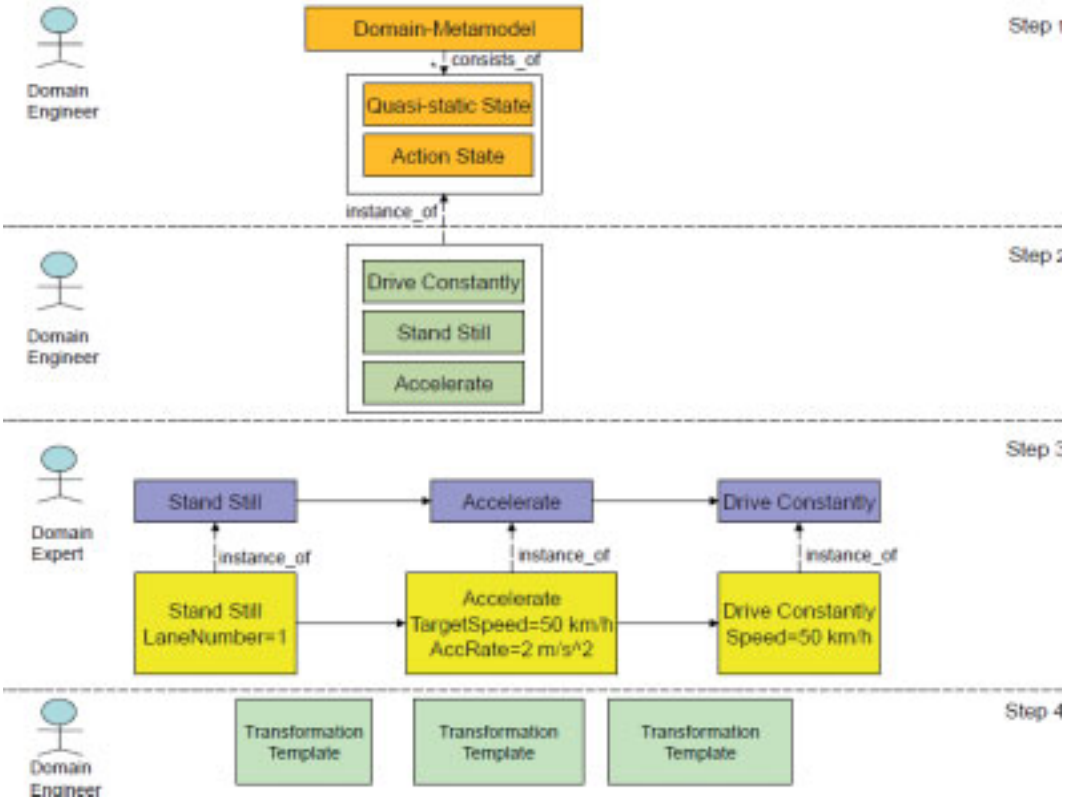


Bild 4.14: Zusammenhang zwischen Metamodell, Modell und Instanz: [ESK+09]

spezifikation (In der Abbildung 4.14 gelb markiert). Schließlich muss ein Domäneningenieur im vierten und letzten Step Templates definieren, damit die Transformation in bestimmte Zielformate geschehen kann.

Partielle Instanziierung des Metamodells ist kein einmaliger Vorgang, sondern ein sich zyklisch wiederholender, denn:

- Es können jederzeit neue Projekte und Subprojekte entstehen, welche am Software-Entwicklungsvorgang innerhalb der Domäne teilnehmen und somit auch diverse Testverfahren im Entwicklungsteam einsetzen.

- Durch ständige Pflege der Beschreibungsattribute sämtlicher Beschreibungselemente einer Methode kann festgestellt werden, dass bestimmte Attribute in allen Projekten und Subprojekten eingesetzt werden. Somit können sie unter Umständen als generische Attribute verwaltet werden. Damit ist allerdings ein Verwaltungsaufwand verbunden, denn falls die Testspezifikationsverwaltung in einer relationalen Datenbank geschieht, müssen beispielsweise serverseitige Funktionen geschrieben werden, welche ein bestimmtes pro-

jektspezifisches Attribut auf der M1-Ebene als ein generisches anlegen und alle bisher angelegten M0 projektspezifischen Instanzen als generische deklarieren. Der Automatisierungsgrad einer solchen Funktionalität ist in diesem konkreten Beispiel relativ hoch, denn solche Funktionen können in einen Datenbank-Trigger eingebunden sein, der jedesmal feuert, wenn ein neues generisches Attribut angelegt wird.

- Es können jederzeit neue M1-Instanzen des Domänen-Metamodells für alle Beschreibungselemente kommen, sodass möglicherweise neue Kategorien für diese notwendig sind. Das Anlegen sowie das Neuzuordnen von M1-Beschreibungselementen zu den neuen Kategorien ist ebenfalls mit einem Verwaltungsaufwand verbunden. Der Automatisierungsgrad ist in diesem Fall niedriger als im vorigen Punkt, denn die eventuelle Neuzuordnung der Beschreibungselemente zu den Kategorien geschieht zum größten Teil manuell.

Aus den obigen Ausführungen wird ersichtlich, dass eine ständige Pflege der partiellen Metamodell-Instanzen durch den Domäneningenieur notwendig ist. Es empfiehlt sich außerdem innerhalb jeder beteiligter Abteilung einen Domänenexperten als Ansprechpartner festzulegen, welcher in regelmäßigen Zeitabständen eventuelle neue Projekte, Subprojekte sowie projektspezifische Änderungswünsche an den Beschreibungen einzelner bereits angelegter Beschreibungselemente (M1) an den Domänenexperten kommuniziert. Idealerweise eignet sich ein Serientermin zwischen dem Domänenexperten und den jeweiligen Ansprechpartnern zur Pflege des partiellen M1-Modells. Im weiteren werden nun die einzelnen Schritte erläutert, welche zur partiellen Instanziierung des Metamodells notwendig sind (Abbildung 4.15).

4.4.1 Anlegen der Unternehmensstruktur

In diesem Schritt müssen Instanzen (M1) des Elements „Extension Category“(M2) angelegt werden. Diese Instanzen entsprechen der im Kapitel 1 ermittelten Unternehmensstruktur in Bezug auf Software-Entwicklung innerhalb einer bestimmten Domäne.

4.4.2 Erarbeitung der Benennungen für Beschreibungselemente

Hier muss der Domäneningenieur in Zusammenarbeit mit den Ansprechpartnern passende Benennungen für die ermittelten Beschreibungselemente der formalen Methode finden. Falls für ein Element keine einheitliche Benennung gefunden wird, dann muss eine Mehrheitsabstimmung als Entscheidungsgrundlage dienen. Die projekt- beziehungsweise subprojektspezifischen Benen-

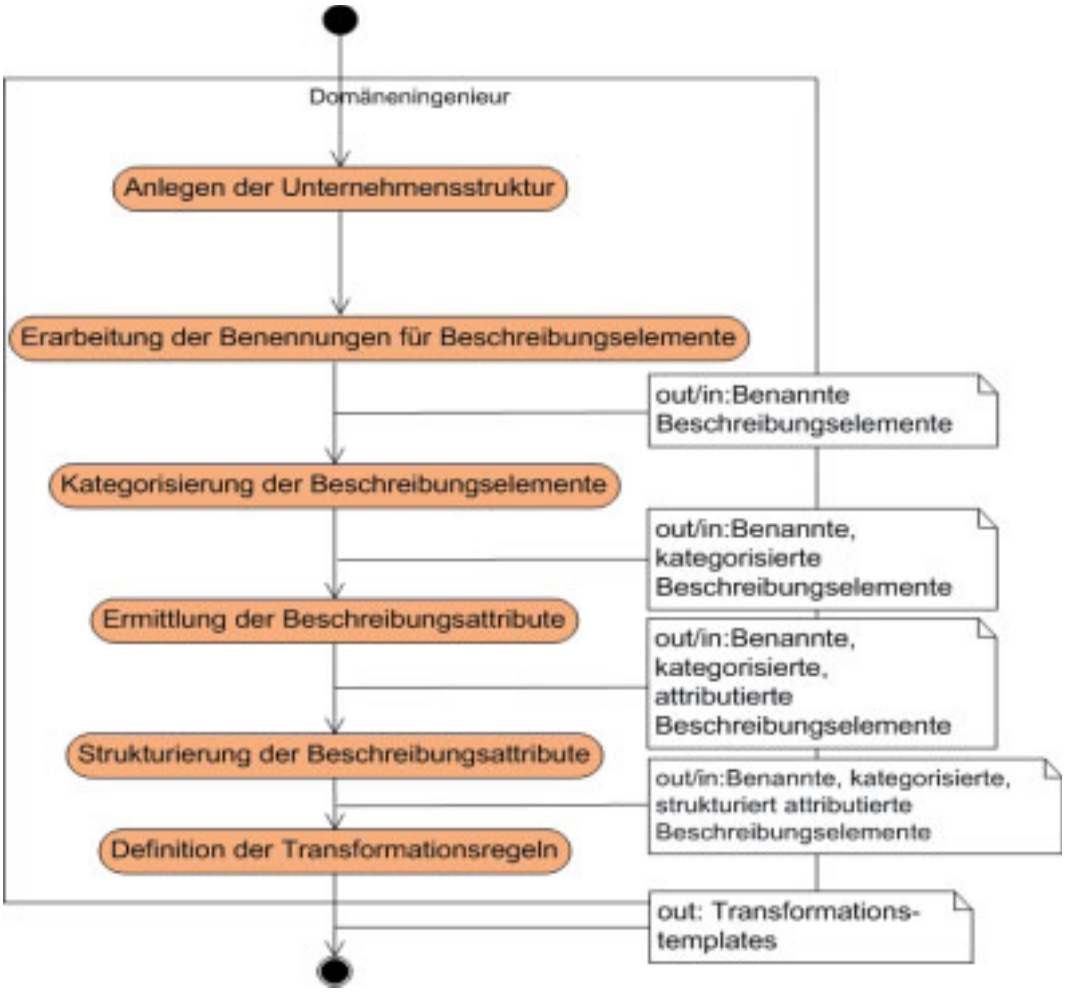


Bild 4.15: Partielle Instanziierung des Metamodells

nungen können als projektspezifische Attribute in den jeweiligen Erweiterungen als Synonyme verwaltet werden.

4.4.3 Kategorisierung der Beschreibungselemente

Zur übersichtlichen Strukturierung der Darstellung muss nun der Domäneningenieur anhand der Analyse ermittelter Beschreibungselemente eine sinnvolle Kategorisierung für diese vornehmen. Selbstverständlich gibt es für jede Gruppe von Beschreibungselementen eigene Kategorisierungshierarchien.

Letztlich muss der Domäneningenieur die einzelnen Beschreibungselemente im System anlegen und diese dann der jeweiligen (Sub)Kategorie zuordnen.

4.4.4 Ermittlung der Beschreibungsattribute

Nachdem nun die Kategorisierungshierarchien feststehen und diesen die ermittelten Beschreibungselemente zugeordnet sind, ist es die Aufgabe des Domäneningenieurs, in Zusammenarbeit mit den Ansprechpartnern mögliche Attribute für die Beschreibungselemente pro Subprojekt und die dort eingesetzte Testmethode zu ermitteln. In diesem Schritt geht es lediglich darum, diese undifferenziert zu sammeln. Neben den Attributnamen sollen, soweit sinnvoll, auch die Wertebereiche für die Attribute gesammelt werden. Eine wichtige Information stellt ebenfalls die Messeinheit dar, soweit die durch sie beschriebenen Größen messbar sind. Dies ermöglicht dem Domänenexperten eine sehr komfortable Nutzung des durch partielle Instanziierung entstandenen Wörterbuchs. Selbstverständlich können pro Attribut mehrere Messeinheiten definiert werden. Beispielsweise kann die Längsgeschwindigkeit sowohl in Meter pro Sekunde als auch in Kilometer pro Stunde ausgedrückt sein.

4.4.5 Strukturierung der Beschreibungsattribute

Hierbei geht es darum, dass der Domäneningeneiur die im vorhergehenden Schritt ermittelten Attribute pro Beschreibungselement in generische und projektspezifische aufteilt und diese anschließend im System anlegt. Für die projektspezifischen Attribute muss logischerweise pro Subprojekt und Testtechnik zuvor eine Erweiterung erzeugt werden, welcher sie zugeordnet werden müssen.

4.4.6 Definition der Transformationsregeln

Auch dieser Vorgang ist nicht einmalig, sondern er muss wiederholt werden bei jedem neu hinzugekommenen Subprojekt beziehungsweise einer dort verwendeten Testmethode. Je nach dem gewählten Transformationsansatz können solche Transformationsregeln in Form von Templates in einer dedizierten Sprache spezifiziert werden. Die Definition von Transformationsregeln ist die Aufgabe des Domäneningenieurs. Zum Schluss sollen alle ermittelten Kategorien, Beschreibungselemente, deren generische und projektspezifische Attribute im System eingetragen werden. Ebenfalls sollen die Transformationstemplates im System eingetragen werden.

4.5 Definition der Testspezifikation

Die Definition der Testspezifikation durch den Domänenexperten findet auf den Ebenen M1 sowie M0 statt (siehe Abbildung 4.6). Zur Erstellung der Testspezifikation wird folgendes Rahmenwerk definiert (Abbildung 4.16). Dieses besteht aus fünf Ebenen, wobei die auf jeder Ebene definierten

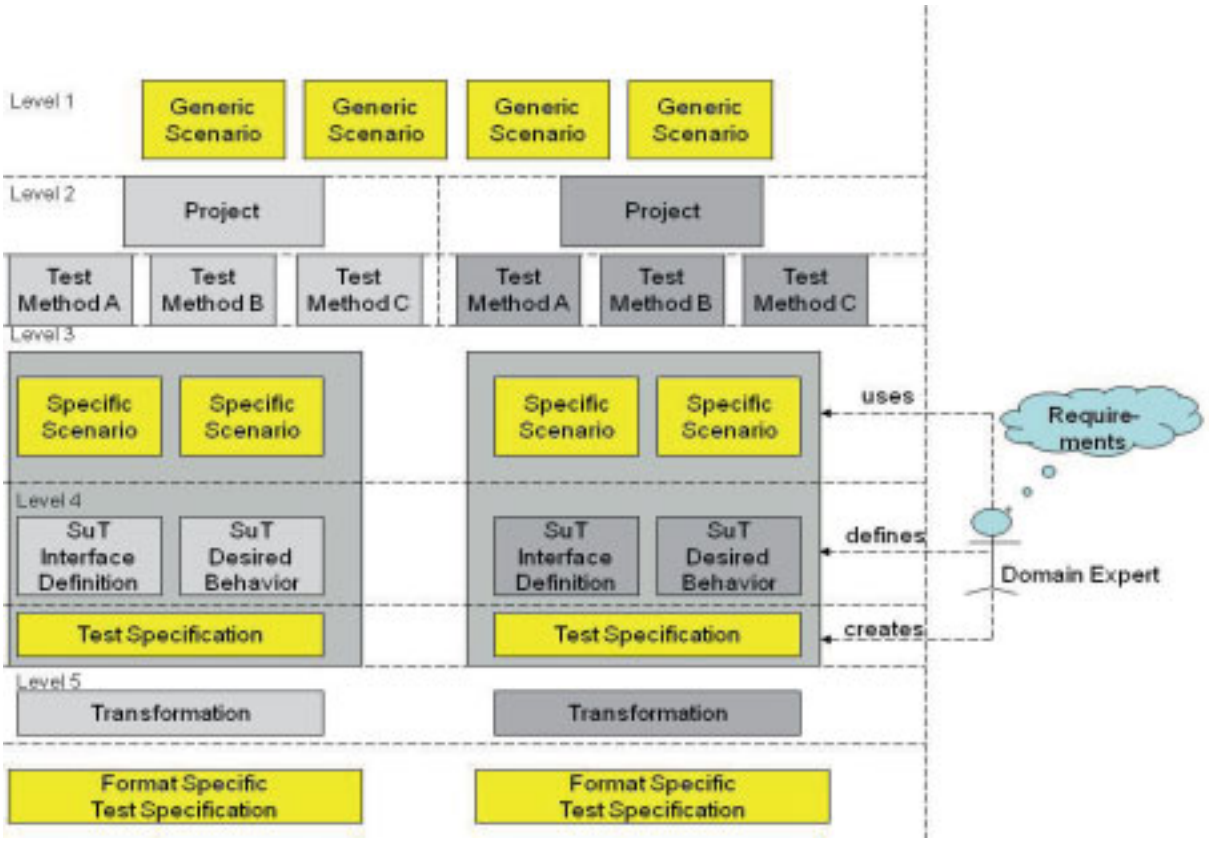


Bild 4.16: Rahmenwerk zur Definition von Testspezifikation durch den Domänenexperten [ESK+09]

Artefakte durch dieselbe Modellierungsmethode beschrieben sind. Jedes auf einer Ebene erzeugte Artefakt wird auf den nachfolgenden Ebenen weiter verfeinert. Die einzelnen Artefakte werden, wie bereits erwähnt, entweder als M1-Modelle oder als M0-Modellinstanzen dargestellt. Die einzige Ebene, auf welcher die vom Domänenexperten definierten Artefakte ihre Darstellungsform ändern, ist die Ebene 5, auf welcher die Transformation in ein plattformspezifisches Format geschieht. Im Weiteren werden nun die einzelnen Ebenen im Rahmenwerk genauer betrachtet.

- Ebene 1: Auf dieser Ebene findet die generische Szenariendefinition statt. Generisch ist sie deshalb, weil sie mit Hilfe einer Menge generischer Attribute geschieht. Solche Definitionen

sind projekt- und testartunabhängig. Was deren Semantik betrifft, so beschreiben sie, wie bereits dargestellt, entweder das Interaktionsverhalten oder das Systemverhalten des SUT. Solche generischen Szenarien beschreiben keineswegs das gewünschte SUT-Verhalten auf dieser Ebene. Dies kann in einem späteren Verfeinerungsschritt geschehen. Denn würde man schon auf generischer Ebene das gewünschte Systemverhalten beschreiben wollen, dann wären solche Szenariendefinitionen nicht mehr projekt- und testartunabhängig und somit wäre der Wiederverwendbarkeitsgrad sehr gering. Ein Domänenexperte kann entweder eine komplette Neudefinition eines solchen Szenarios vornehmen oder er wählt eines aus einem bereits bestehenden Pool aus Szenarien und verfeinert dann das gewählte Szenario gemäß den Anforderungen seines Subprojekts und eines spezifischen dort eingesetzten Testverfahrens. Als Ausgangsbasis für die Szenarienbeschreibung dient meistens das Wissen der Domänenexperten. Beispielsweise könnte man für einen Spurerkennungsalgorithmus die Fahrszenarienbeschreibung um solches Attribut wie „Spur-Markierungsfarbe“ ergänzen. Andererseits kann man ein bereits bestehendes Szenario modifizieren. So kommt es oft vor, dass man während der HiL-Tests ein und dasselbe Abbremsmanöver mit unterschiedlicher Startgeschwindigkeit abprüft. Somit würde man bei einem bereits angelegten Fahrszenario lediglich den Attributwert „Startgeschwindigkeit“ ändern und das Fahrszenario als eine neue Instanz (M0) abspeichern. Wichtig nochmals zu erwähnen ist, dass Szenarien auf dieser Ebene unabhängig von den funktionalen Anforderungen eines bestimmten Projekts beziehungsweise eines bestimmten zu testenden Moduls (FUT) sind.

- Ebene 2: Nachdem nun das generische Szenario entweder neu erstellt oder aus einer bereits bestehenden Menge ausgewählt wurde, muss der Domänenexperte sich für ein Projekt oder Subprojekt beziehungsweise eines in diesem Rahmen eingesetztes Testverfahren entscheiden. In Abhängigkeit von der getroffenen Entscheidung wählt er aus dem Wörterbuch die Menge entsprechender subprojekt- beziehungsweise testverfahrenabhängiger Beschreibungsattribute und zwar für jedes Element des auf der Ebene 1 definierten generischen Szenarios. Diese Entscheidung trifft ein Domänenexperte ausgehend von den funktionalen Anforderungen an das zu testende Modul.

- Ebene 3: Auf dieser Ebene belegt der Domänenexperte die aus dem Wörterbuch gewählten subprojekt- und testverfahrenspezifischen Attribute eines Szenarios mit konkreten Werten. Um ein Beispiel aus der Anwendungsdomäne zu geben, werden zwei Funktionen betrachtet. Eine beschäftigt sich im Bereich der bildbasierten Mustererkennung damit, Spurmarkierungen zu erkennen, während eine andere sich mit der Unterstützung des Fahrers beim

Spurwechsel des EGO-Fahrzeugs befasst. Die erste Funktion benötigt als projektspezifisches Attribut beispielsweise die Spurmarkierungsfarbe oder Lichtzustand, während die zweite zum Beispiel die Querbeschleunigung beim Fahrmanöver „Spurwechsel“ benötigt. Was die Semantik eines subprojekt- bzw. testartspezifischen Szenarios betrifft, so kann dieses in Abhängigkeit vom zu testenden Modul das Soll-Verhalten beschreiben oder aber es können daraus über einen Transformationsschritt (siehe Ebene 5) beispielsweise Testdaten generiert werden, mit denen ein SiL-Vorgang gestartet werden kann.

- Ebene 4: In Abhängigkeit davon, ob das Szenario das gewünschte Verhalten des zu testenden Moduls beschreibt oder nicht, muss dieses noch um Referenzwerte ergänzt werden (Im Fall, dass das Szenario kein gewünschtes Modulverhalten beschreibt). Die Referenzwerte werden in Form von subprojektspezifischen Attributen beschrieben. Die letzte Aktivität auf dieser Ebene betrifft die Interface-Spezifikation des zu testenden Moduls. Diese dient dazu, dass die Testumgebung, welche beispielsweise den automatisierten Testablauf steuert, das zu testende Modul ansprechen kann.

- Ebene 5: Auf dieser Ebene geschieht die eigentliche Transformation aller bisher erzeugten Artefakte in die plattformspezifische Notation. Die erzeugten Formate können zum einen dazu dienen, einen automatischen Testablauf zu starten oder aber einen manuellen wie zum Beispiel eine Testfahrt auf dem Prüfgelände. Im letzteren Fall kann beispielsweise während der Transformation ein (elektronisches) Dokument mit den Richtlinien für den Testfahrer erzeugt werden. Neben den eigentlichen Bestandteilen einer Testspezifikation können ebenfalls zusätzliche Informationen generiert werden, welche beispielsweise einen Zugriff auf eine Testdaten-Datenbank erlauben oder aber die Parametrisierung für die Entwicklungsumgebung beinhalten, in der das zu testende Modul entwickelt worden ist. Solche Informationen ändern sich relativ selten, deshalb können sie einfach beispielsweise in einem Transformationstemplate festkodiert werden.

Kapitel 5

Konzeptanwendung

In diesem Kapitel erfolgt die Anwendung des im Kapitel 4 definierten Vorgehensmodells auf die FAS-Domäne. Die Anwendung erfolgte für eine Vorentwicklungsabteilung im Bereich Fahrwerk und Elektronik, Entwicklung von Fahrer-Assistenzsystemen. Der Kapitelaufbau richtet sich nach dem definierten Vorgehensmodell.

5.1 Wahl der Szenarienmodellierungsmethode für die Domäne

5.1.1 Analyse bestehender Testtechniken

Auf die Darstellung des ersten Schrittes im Vorgehensmodell, Analyse bestehender Testtechniken, wird hier verzichtet, da sie bereits im Kapitel 1 und 2 erfolgte und die dort beschriebenen Testverfahren bis auf MiL ebenfalls in der untersuchten Abteilung zum Einsatz kommen. Im folgenden wird also auf die Wahl der Modellierungsmethode eingegangen.

5.1.2 Wahl der Modellierungsmethode

Bevor die eigentliche Auswahl geschieht, muss die Struktur eines Fahrszenarios erläutert werden sowie Anforderungen an die formale Szenarienmodellierungsmethode sowohl aus Entwicklersicht als auch aus informatischer Sicht.

Allgemeine Struktur eines Fahrszenarios

Im Kontext der Entwicklung von Fahrer-Assistenzsystemen wird in unterschiedlichsten Testverfahren mit den Fahrszenarien gearbeitet. Ein Fahrszenario ist zunächst die Beschreibung der Interaktion des SUT mit der Umwelt. Durch dessen Ergänzung mit projekt- und testartspezifischen Attributen gewinnt man daraus die Interaktionsbeschreibung der FUT (engl. Functionality Under Test) mit der Umwelt. Der Aufbau eines Fahrszenarios lässt sich grob in drei Kategorien unterteilen: EGO-Fahrzeug, andere Verkehrsteilnehmer und die Umgebung. Bei virtuellen und realen Testverfahren mit den Fahrszenarien wird zunächst die Umgebung beschrieben. Dazu gehört sicherlich die Beschreibung der Strecke, wie beispielsweise Anzahl der Spuren, Art der Strecke, also etwa Autobahn oder Nebenstraße. Bei virtuellen Testverfahren kann die Strecke sehr genau durch Angabe von Krümmungen und Steigungen/Senkungen beschrieben werden. Außerdem gehört zur Umgebung die Angabe von statischen Objekten wie Brücken, Verkehrszeichen oder aber Gebäuden. Sicherlich müssen auch noch die Wetterzustände angegeben werden, welche beispielsweise aus Angabe der Beleuchtung, des möglichen Niederschlags, des Spurzustands bestehen können. Bei der Beschreibung des EGO-Fahrzeugs spielen beispielsweise solche Größen eine Rolle wie die momentane Geschwindigkeit, Position in der Spur, Status der zu testenden Assistenzfunktion (FUT) (aktiv/inaktiv). Bei virtuellen Testverfahren kann man ebenfalls das Fahrerverhalten bestimmen, also ob der Fahrer beispielsweise die Verkehrszeichen beachten soll. Bei Fremdfahrzeugen wird die Klasse angegeben (PKW, LKW), Ablage in der Spur sowie Geschwindigkeit. Bei virtuellen Vorgängen werden die Verkehrsteilnehmer grundsätzlich wesentlich genauer beschrieben als bei realen Messfahrten. So spielen dort solche Eigenschaften eine Rolle wie zum Beispiel der Raddiameter oder die Motorleistung.

Da es sich um Fahrszenarien handelt, muss sicherlich in einer Fahrszenarienbeschreibung eine Menge von Ereignissen angegeben werden, die von den Verkehrsteilnehmern, dem EGO-Fahrzeugfahrer, der Assistenzfunktion, der Umgebung während des Szenarios ausgelöst werden. Im Folgenden werden nun die genauen Anforderungen an eine Fahrszenarienbeschreibung beschrieben. Abschnitt 5.1.2 „Anforderungen aus der Entwicklersicht“ betrachtet die Anforderungen aus Sicht des Funktionsentwicklers, welche sich bei Gesprächen mit diesen herauskristallisiert haben. Sie sind aus diesem Grund, zumindestens aus informatischer Sicht, noch unscharf. Deshalb bildet der Abschnitt „Anforderungen aus informatischer Sicht“ 5.1.2 die in „Anforderungen aus Entwicklersicht“ beschriebenen Anforderungen auf die Begriffe der Informatik ab.

Anforderungen aus der Entwicklersicht

Grundsätzlich handelt es sich bei der Darstellung von Fahrszenarien um dynamische Abläufe, die sowohl parallel als auch sequentiell ausgeführt werden können. Man unterscheidet generell zwischen Ereignissen und quasistatischen Zuständen. Der Begriff „quasistatischer Zustand“ kommt aus der Luftfahrt. Dort befindet sich ein Flugzeug in einem quasistatischen Zustand, falls es eine längere Zeit keine seinen Zustand maßgeblich ändernden Manöver ausführt. Das klassische Beispiel ist das Fliegen mit dem Autopilot. Quasistatischen Zustände sind solche, bei denen eine längere Zeit keine Testfall-relevanten, das Fahrzeug betreffenden Ereignisse eintreten wie beispielsweise: „Fahren in der linken Autobahnspur mit einer konstanten Geschwindigkeit“ oder „Fahrzeug ist im Stillstand“. Ereignisse führen von einem quasistatischen Zustand in den anderen. Die Ereignisse können sowohl das EGO-Fahrzeug betreffen als auch die Fremdfahrzeuge. Es gibt ebenfalls Ereignisse, die als Voraussetzung erfüllt sein müssen, damit das Ereignis ausgelöst wird, das von einem quasistatischen Zustand in den anderen führt. So kann beispielsweise das Ereignis „Regen“ zum Eintritt des Ereignis „Scheibenwischer einschalten“ führen. Da es sich um Testen handelt, soll es die Möglichkeit geben, FUT-abhängige (!) Soll-Werte für die Variablen zu definieren, die den Fahrzeugzustand nach dem Eintreten des Ereignisses beschreiben. Dazu wird die Kategorie der quasistatischen Zustände in zwei Unterkategorien unterteilt: Setup-Zustände, Soll-Zustände. Die Soll-Zustände beschreiben den erwarteten Fahrzeugzustand nach dem Eintreten eines bestimmten Ereignisses beim Testen einer spezifischen Assistenzfunktion. Setup-Zustände beschreiben den Fahrzeugzustand vor dem Eintreten des Ereignisses. Ein Ereignis führt von einem Setup-Zustand in einen Soll-Zustand.

Anforderungen aus informatischer Sicht

Aus informatischer Sicht muss zunächst eine klare Trennung zwischen Ereignissen und Zuständen geschaffen werden. Die Trennung in Ereignisse, die als Voraussetzung für das Eintreten anderer Ereignisse dienen (siehe vorigen Abschnitt), ist unpräzise. Deshalb muss die Kategorie der Zustände um die so genannten Aktionszustände erweitert werden. Auf die Aktionszustände werden die Ereignisse aus Abschnitt „Anforderungen aus Entwicklersicht“ 5.1.2 abgebildet, die von einem quasistatischen Zustand in den anderen führen. Die Ereignisse, die als Voraussetzung dafür dienen, werden als Stimuli oder auch als Events bezeichnet. Die Aktionszustände bringen eine dynamische für den Testfall relevante Änderung des Setup-Zustands zum Ausdruck. Während sich das System im Aktionszustand befindet, wird, wie der Name schon vermuten lässt, eine bestimmte Aktion ausgeführt. Die Stimuli lösen somit den Übergang von einem quasistatischen Zustand in einen Aktionszustand aus. Ein Beispiel für den Stimulus kann z.B. „das Eintreten eines

bestimmten Zeitpunkts“ oder das „Erreichen einer bestimmten Position auf der Fahrspur“ sein. Aktionszustände beziehungsweise Aktionen, die beim Betreten des Aktionszustands ausgeführt werden, betreffen sowohl Eigenfahrzeug als auch Fremdfahrzeuge (z.B. EGO-Fahrzeug bremst, Fremdfahrzeug schert aus etc.). Es muss ebenfalls die Möglichkeit geben, für eine bestimmte Menge von Aktionen, die während des Befindens im Aktionszustand ausgeführt werden, Sachverhalte zu beschreiben, die sich während ihrer Ausführung nicht ändern, wie z.B. Wetterzustände, Fahrbahnzustände u.ä.) Dazu muss es möglich sein, die Aktionszustände in einem Testschritt zusammenzufassen. Für jeden Testschritt wird ein globaler Zustand definiert. Die Variablen, die diesen beschreiben, ändern sich während der gesamten Testschritt-Ausführung nicht. Zwecks der Übersichtlichkeit muss es möglich sein, für Testschritte Untertestschritte zu definieren. D.h. jeder Testschritt besteht aus einer Menge von Untertestschritten. Somit muss die Anforderung der Hierarchisierung erfüllt sein. Einzelne Testschritte können sowohl parallel als auch sequentiell ausgeführt werden.

Da es sich bei Fahrszenarien um die Modellierung hoch dynamischer Vorgänge handelt, soll es die Möglichkeit geben, globale Startzeitpunkte für die Aktionszustände bzw. ihre Aktionen zu ermitteln. Bestimmte Aktionszustände können auch mit einer Verzögerung erreicht werden. Besonders für virtuelle automatisierte Testvorgänge soll es die Möglichkeit geben, die Dauer anzugeben, während welcher sich das System in dem Aktionszustand befindet. Die Stimuli können absolut global sein, das heißt, ihre Auslösung geschieht beim Erreichen eines bestimmten globalen Zeitpunkts oder beim Erreichen einer bestimmten Position auf der Fahrbahn. Stimuli können auch relativ zu bestimmten Aktionszuständen ausgelöst werden. So kann ein Stimulus beispielsweise ausgelöst werden, wenn ein bestimmter Aktionszustand verlassen wird ist oder wenn er betreten wird. Ein Stimulus kann auch abstandsbezogen sein, d.h. er wird ausgelöst, wenn ein bestimmter Abstand zwischen dem EGO-Fahrzeug und z.B. dem Vorderfahrzeug erreicht ist. In bestimmten Fällen sind die Setup- oder Soll-Zustände für den Entwickler nicht von Interesse: Er möchte nur einen bestimmten Aktionszustand modellieren. Besonders für die Hardware-in-the-Loop-Vorgänge muss die Möglichkeit berücksichtigt werden, dass bestimmte Abläufe mehrmals wiederholt werden können.

Nun werden ausgehend von den im vorigen Abschnitt bestimmten Anforderungen und den im Kapitel drei aufgestellten Bewertungskriterien einzelne Modellierungsmethoden untersucht und bewertet.

5.1.2.1 Vorauswahl

Endliche Automaten

Endliche Automaten erlauben es, die wesentlichen aufgestellten Anforderungen zu erfüllen. Die Setup- und quasistatischen Zustände sowie Stimuli und Aktionen, die während des Befindens im Aktionszustand ausgeführt werden, lassen sich mit dieser Methode darstellen. Was problematisch zu modellieren ist, ist zum einen die globale Zeit, da es in der klassischen Definition kein „Clock"-Konstrukt gibt. Zum anderen sieht die Definition keine Zeitangaben für eine Aktion vor. Eine Erweiterung für endliche Automaten namens „zeitbehaftete Automaten" [AD94] erlaubt zwar die Berücksichtigung der Auftrittzeitpunkte/Intervalle einzelner Stimuli, allerdings werden keine Angaben zur Dauer einzelner Aktionen gemacht. Außerdem ist keine hierarchische Gliederung zeitbehafteter Automaten möglich. Die Automaten sind durch ein mathematisches Modell beschrieben, sie sind korrekt und vollständig, allerdings dürfte speziell bei zeitbehafteten Automaten die Verständlichkeit dem Entwickler Probleme bereiten.

Petri-Netze

Klassische Petri-Netze erlauben [Rei85] die Modellierung solcher Anforderungen wie Parallelität oder Darstellung von Zyklen. Außer Acht gelassen werden die Hierarchisierung und die Berücksichtigung von zeitlichen Aspekten. In der Literatur wurde eine große Menge an Erweiterungen für Petri-Netze gefunden, die die Zeit in Betracht ziehen. Besonders geeignet würde der Ansatz von [AHR00] erscheinen, der für jede Aktion im Netz die Angabe eines zeitlichen Intervalls ermöglicht. Der Zeitpunkt aus diesem Intervall stellt eine Verzögerung dar, die zwischen dem Zeitpunkt des Feuerns einer Aktion und dem der Erzeugung eines Ausgangs-Tokens vergeht. Es ist ebenfalls möglich, mit den zeitbehafteten Petri-Netzen Verzögerungen nicht nur für Aktionen zu definieren, sondern auch für Tokens: Ein Token kann also nicht sofort zum Feuern einer Aktion führen, sondern es kann erst eine definierte Zeit (ebenfalls eine Verzögerung genannt) in einer Stelle verweilen, bevor die Aktion gefeuert wird. Die Petri-Netze sind ebenfalls korrekt und vollständig, allerdings wurde die Möglichkeit der Hierarchisierung nicht gefunden, somit wird auch die Wiederverwendbarkeit von bestimmten Teilen der Testfallbeschreibung schwierig. Ebenfalls würde sich die Modellierung von globalen Zuständen als mühsam gestalten. Auch die Übersichtlichkeit der Darstellung und somit auch Verständlichkeit leiden daran. Nichtsdestoweniger gibt es einige an Petri-Netze angelehnte Notationen, von denen eine im nächsten Abschnitt betrachtet wird.

Aktivitätsdiagramme

„Aktivitätsdiagramme sind in UML 2.0, semantisch gesehen, sehr stark an Petri-Netze angenähert worden“ [Mü07]. Sie erlauben die Modellierung einer Abfolge von Aktivitäten und des Objektflusses zwischen ihnen. Die Aktivitäten können aus mehreren Aktionen bestehen. In der Spezifikation ist erwähnt, dass sowohl für Aktivitäten als auch für Flüsse Bedingungen definiert werden können, die spezifizieren, wann z.B. ein Objekt (Token) von der empfangenden Aktivität angenommen werden darf, wann ein Objekt über den Fluss verschickt werden darf oder wann überhaupt eine Aktivität starten darf. Problematischer erscheint die Modellierung von Zuständen, wie sie in den oben dargelegten Anforderungen erwünscht ist, denn mit den Aktivitätsdiagrammen ist keine explizite Zustandsmodellierung möglich, was beispielsweise die Darstellung von Soll-Größen einer FUT erschwert. Die Modellierung von Zeit (Aktivitätsdauer, Verzögerungen, globale Uhr) ist bei Aktivitätsdiagrammen ohne spezielle Profile nicht möglich. Positiv ist hingegen die Einführung von Events in UML 2.0 zu bewerten. So ist es möglich, das Senden und das Empfangen eines Ereignisses darzustellen. Mit diesem Konstrukt kann beispielsweise die Ausführungsabhängigkeit einer Aktion von einer anderen sehr gut modelliert werden. Das graphische Beispiel dazu findet sich in der Abbildung 5.1. Bei der formalen Definition von Fahrszenen geht es vor allem darum,

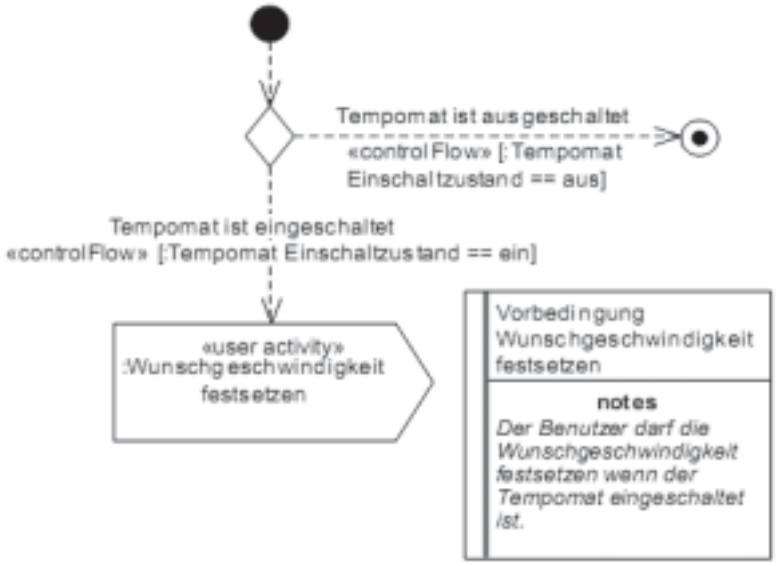


Bild 5.1: Aktivitätsdiagramm [Alt]

Ereignis-abhängige Änderungen von den Systemzuständen der Verkehrsteilnehmer zu modellieren, während die Aktivitätsmodellierung bei Aktivitätsdiagrammen „emphasizes the sequence

and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models.“ [Obj09]. Aus Sicht der Fahrszenenmodellierung ist die Darstellung des Objektflußes unwichtig, denn zwischen den einzelnen Systemzuständen fließen keine Objekte. Somit ist aus Sicht der Fahrszenenmodellierung bei den Aktivitätsdiagrammen ein redundantes Modellierungselement (Objektfluß) vorhanden, das bei der Metamodelldefinition ausgeschlossen werden müsste. Es sei an dieser Stelle anzumerken, dass die Modellierung von Ereignissen erst ab UML-Version 2.0 gekommen ist. Dies zeugt auch davon, dass bei Aktivitätsdiagrammen die Modellierung von Ereignissen eine geringere Rolle spielt als beispielsweise bei den Statecharts. Ein anderes Argument gegen die Anwendung der Aktivitätsdiagrammen ist die Tatsache, dass diese den Entwicklern weitgehend unbekannt sind. Die Aktivitätsdiagramme sind korrekt, da ihnen ein Metamodell zugrunde liegt [Obj09], das es erlaubt, die modellierten Fahrszenarien zu validieren. Sie sind ebenfalls vollständig, da in der umfangreichen Spezifikation von UML 2.0 sowohl syntaktische als auch semantische Beschreibungen für jedes Konstrukt vorliegen. Eine Basis-Erweiterbarkeit ist ebenfalls durch den Profil-Mechanismus sichergestellt, allerdings lässt sich das UML-Metamodell von OMG um die im Vorgehensmodell (siehe Basis-Metamodell, Kapitel 4) definierten Konstrukte ohne Verletzung der Spezifikation nicht erweitern. Was die Benutzer-bezogenen Aspekte betrifft, so besitzen die Aktivitätsdiagramme eine relativ intuitive graphische Darstellung und sind somit auch schnell zu erlernen. Wichtig ist natürlich, das modellierte Fahrszenario nicht sofort mit Einschränkungen zu überladen, denn sonst geht die Lesbarkeit schnell verloren. Die Einschränkungen sollten während des Abbildungsvorgangs sukzessive der Beschreibung hinzugefügt werden [Mü07].

Sequenzdiagramme

Sequenzdiagramme dienen zur Modellierung von Interaktionen. Eine Interaktion spezifiziert die Art und Weise, „wie Nachrichten und Daten über die Zeit hinweg zwischen verschiedenen Interaktionspartnern in einem bestimmten Kontext ausgetauscht werden, um eine bestimmte Aufgabe zu erfüllen“ [HKKR05]. Sequenzdiagramme sind eng mit Message Sequence Charts verwandt [ITU04]. Im Rahmen der Untersuchung wurde vor allem Wert auf die Möglichkeit echt-parallele Abläufe zu modellieren gelegt, denn gerade bei den Fahrszenen findet man häufig echte Parallelität vor. Als Beispiel soll ein ganz einfacher Sachverhalt dienen: “Drei Fahrzeuge fahren mit der konstanten Geschwindigkeit von 40 km/h“. Die Sequenzdiagramme erlauben die Möglichkeit, nebenläufige Prozesse abzubilden. Die Nebenläufigkeit ist nach [Tan08] folgendermaßen definiert. „Wenn zwei Ereignisse in zwei unterschiedlichen Prozessen auftreten, die keine Nachrichten austauschen (nichteinmal indirekt über Dritte), dann sagt man, dass diese Ereignisse nebenläufig

sind, was einfach bedeutet, dass nichts darüber ausgesagt werden kann (oder muss), wann die Ereignisse aufgetreten sind oder welches Ereignis zuerst aufgetreten ist“. Zur Darstellung der Nebenläufigkeit wird bei UML-Sequenzdiagrammen in Version 2.0 das kombinierte Fragment „par“ verwendet. Zunächst würde man zur Darstellung des oben beschriebenen Sachverhalts folgendes Sequenzdiagramm konzipieren (Abbildung 5.2). Semantisch würde das dargestellte Diagramm

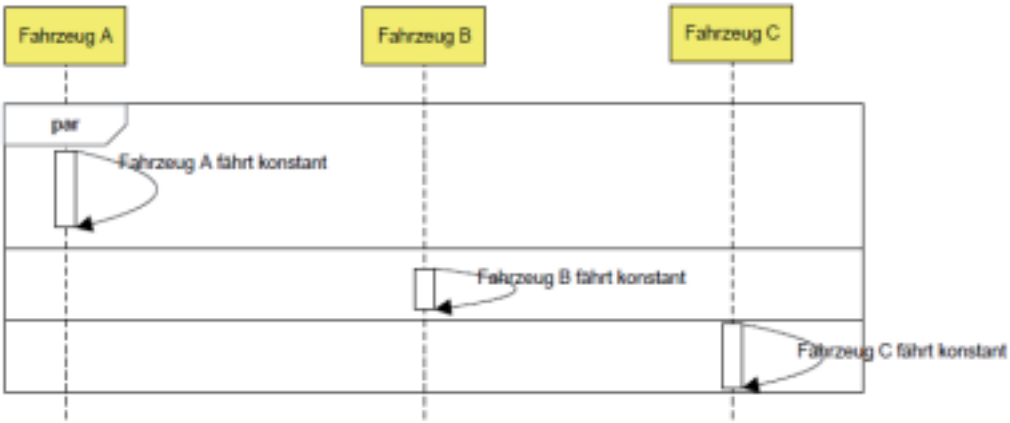


Bild 5.2: Modellierung der Fahrszenen mit Sequenzdiagrammen: Beispiel

folgendes aussagen: „Das Ereignis (a) „Fahrzeug A fährt konstant“ kann als erstes passieren, Das Ereignis (b) „Das Fahrzeug B fährt konstant“ als zweites, Das Ereignis (c) „Fahrzeug C fährt konstant“ als drittes, oder das Ereignis b als erstes, c als zweites, a als drittes, oder c als erstes, b als zweites a als drittes usw. Die echte Parallelität im Sinne einer absolut gleichzeitigen Ausführung wird damit aber semantisch gesehen nicht zum Audruck gebracht. Das heißt, „die Beliebigkeit der Reihenfolge ist maßgebend, eine echte Parallelität im Sinne eines gleichzeitigen Ablaufs der Operanden wird durch dieses Konstrukt per se nicht gefordert, wie das Kürzel des Operators „par“ vermuten ließe“ [HKKR05]. Somit ist die Verwendung von Sequenzdiagrammen zur Darstellung von Fahrszenen ohne Einführung weiterer Einschränkungen oder zusätzlicher Modellierungskonstrukte nicht möglich.

Weitere untersuchten Arten von Sequenzdiagrammen wie beispielsweise „Live Sequence Charts“ (LSC) [DH98] tragen ebenfalls wenig zur Lösung des gestellten Problems bei. Der Hauptvorteil von LSC, nämlich die Möglichkeit, Alternativen und die nicht erlaubten Folgen von Ereignissen zu modellieren, kann bei der Modellierung von Fahrszenenbeschreibungen nicht sinnvoll angewandt werden: Die Entwickler wissen ganz genau, wie die Fahrszene aussieht, und sie wird entweder komplett gefahren oder gar nicht. Alternativen gibt es nicht.

Statecharts

Statecharts [Har87] wurden von Professor Harel (Weizmann-Institut, Rehovot) im Jahr 1987 erfunden. Sie stellen eine Erweiterung gewöhnlicher endlicher Automaten um solche Konzepte wie Nebenläufigkeit, Hierarchisierung und Kommunikation dar. Der grundlegende Aufbau eines Statecharts ist in der Abbildung 5.3 zu sehen. Das zu modellierende System besteht aus mehreren

Bild 5.3: Statechart: Grundsätzlicher Aufbau

Zuständen. Die Übergänge zwischen diesen Zuständen werden als Transitionen bezeichnet, deren Dauer auf 0 gesetzt wird. Jede Transition befolgt die Regel „Event [Guard] /Action“. Das heißt, sobald ein Ereignis eintritt und die Bedingung (engl. „guard“) erfüllt ist, ist die Transition schaltbereit. Während des Schaltens kann (muss aber nicht) eine bestimmte Aktion ausgelöst werden. Es sei an dieser Stelle ausdrücklich erwähnt,dass die Angabe von Event, Condition und Action optional ist.

Somit werden Stimuli auf Events abgebildet. Sind bei einer Transition weder „Events“ noch „Guards“ angegeben, schaltet die Transition sofort ohne zu warten. Interessant ist es, den Aufbau eines Zustands bei Statecharts zu betrachten (Abbildung 5.4). Entry-Actions werden beim Betreten

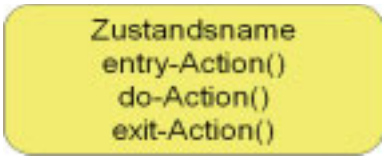


Bild 5.4: Statechart: Zustandsaufbau

des Zustands ausgelöst. Exit-Actions werden beim Verlassen des Zustands ausgelöst. Die Exit-Action wird vor der Action ausgelöst, die beim Schalten der Transition auftritt (Abb. 5.3). Die Entry-Action wird nach der Action ausgeführt, die während des Schaltens einer Transition ausgeführt werden kann. Aus Sicht der Fahrszenarienmodellierung sind vor allem die do-Actions interessant, denn sie bringen die Aktionen zum Ausdruck, welche ausgeführt werden, während

sich das System im Zustand befindet. Die do-Action wird nur einmal ausgelöst und ihre Dauer entspricht der Dauer, während welcher sich das System in diesem Zustand befindet. Somit können die Aktionszustände aus Abschnitt 5.1.2 „Anforderungen aus informatischer Sicht“ auf Statechart-Zustände abgebildet werden, wobei jedes Mal eine do-Action spezifiziert werden soll, welche die auszuführende Aktion realisiert (Abb. 5.5). Do-Actions werden entweder durch einen Präfix „do:“ gekennzeichnet oder sie werden durch den Namen eines Zustands zum Ausdruck gebracht. Die in den Anforderungen spezifizierten Setup- und Sollzustände werden auf die entsprechend benannten Zustände in den Statecharts abgebildet (Abbildung 5.5). Die Ausdrucksmächtigkeit

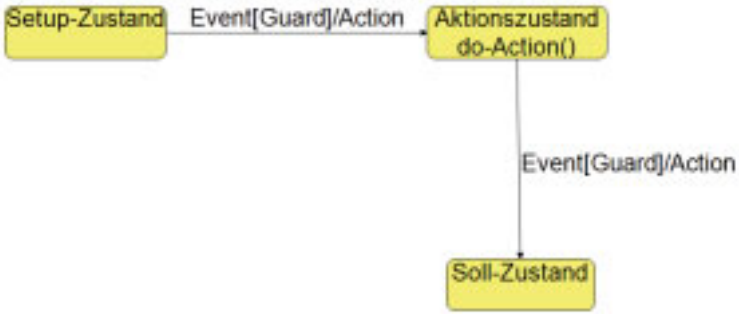


Bild 5.5: Statechart: domänenspezifische Zustandsarten

der Statecharts erlaubt die Modellierung einer Großzahl der in den Anforderungen dargelegten Sachverhalte. Das einzige, was noch unberücksichtigt bleibt, ist die globale Zeit. Ein Vorschlag zur graphischen Angabe von globalen Zeitpunkten wird im Abschnitt „Darstellung von Aktionen“ gemacht. Die formale Semantik von Statecharts ist in [HPSR87] zu finden.

Ebenfalls haben die Statecharts im Rahmen von UML Verbreitung gefunden. Deren Semantik ist also in einem UML-Metamodell abgebildet, wodurch die Validierung von modellierten Testfällen gegenüber dem Metamodell möglich wird. Statecharts sind sowohl in [Har87] als auch in der entsprechenden UML-Spezifikation vollständig beschrieben [Obj09]. Da das vorliegende Statechart-Metamodell um die in Kapitel 4 beschriebenen Modellierungselemente erweitert werden muss, wird der enstprechende OMG-Standard für UML verletzt. Daher soll bei der Implementierung auf andere Sprachen zugegriffen werden. Gleichzeitig muss betont werden, dass die Methode (also die Statecharts) und ihre Semantik zum größten Teil beibehalten werden sollen. Gleichwohl kann das vom OMG definierte Metamodell als Ausgangsbasis für die Erweiterung dienen. Im Übrigen ist die Wiederverwendbarkeit einzelner Teile der Testfälle durch ein Hierarchisierungskonzept gegeben. Statecharts sind für die Domainexperten relativ schnell zu erlernen, da ihnen das Automatenkonzept bereits aus anderen Modellierungsumgebungen wie z.B.

Matlab bekannt ist. Im nächsten Abschnitt werden nun die einzelnen an die Domäne angepassten Sprachkonstrukte vorgestellt.

Sprachkonstrukte

Darstellung von Zuständen

Jeder Zustand wird charakterisiert durch:

- Zustandskategorien.
- Die Menge an beschreibenden Attributen.
- Eine „State-Guard"-Bedingung, die bestimmt, wann das am Test beteiligte System sich in diesem Zustand befinden darf. Die „State-Guard"-Bedingung wird beim Schalten der entsprechenden Transition überprüft. Falls kein „State Guard" angegeben ist, schaltet die Transition ohne Überprüfung und somit wird der Zustand betreten. Falls eine State-Guard-Bedingung angegeben aber nicht erfüllt ist, wird solange gewartet, bis sie erfüllt ist und die Transition schalten kann, denn sonst handelt es sich um einen Modellierungsfehler und die Fahrszene kann überhaupt nicht stattfinden.
- Eine „Entry Action", die beim Betreten des Zustands ausgeführt wird. Falls keine angegeben ist, wird keine Aktion beim Betreten des Zustands ausgeführt. Eine Entry Action kann beispielsweise der Variablendefinition oder Variableninitialisierung dienen.
- Eine „do-Action" ist eine Aktion, die während des gesamten Verbleibens des am Test beteiligten Systems in einem bestimmten Zustand einmal ausgeführt wird.
- Eine „Exit Action" ist eine Aktion, die beim Verlassen des Zustands ausgeführt wird. Sollte keine „Exit Action" angegeben werden, dann wird der Zustand ohne deren Ausführung verlassen. Eine „Exit Action" könnte beispielsweise eine Log-Funktion aufrufen oder ein bestimmtes Ereignis auslösen.

Es wird grundsätzlich zwischen fünf Zustandsarten unterschieden (globale Zustände, Setup-Zustände, Soll-Zustände, Unterschritte sowie Aktionen):

- Globale Zustände: Sie werden auch als Testschritte bezeichnet. Diese werden durch eine Menge an Attributen beschrieben, bei denen davon ausgegangen wird, dass sie sich während des Testschritts nicht ändern werden. Solche Attribute können beispielsweise Straßenzustände, Anzahl der Spuren oder auch Wetterzustände beschreiben. In der prototypischen

Implementierung wurden beispielsweise drei globale Zustände verwendet: Vorszene, Szene sowie Nachszene. Ein globaler Zustand wird dann betreten, wenn eine Transition schaltet. Ein Fahrszenario wird auf der obersten Ebene durch eine geordnete Menge an Testschritten beschrieben.

- Setup-Zustände: In einem Setup-Zustand werden alle Variablen aufgelistet, die vor dem Übergang in den Aktionszustand vorhanden sein und bestimmte Werte aufweisen müssen.
- Soll-Zustände: In einem Soll-Zustand werden Soll-Werte für die zu überprüfenden Systemvariablen angegeben. Als „Do-Action“ kann beispielsweise eine oder mehrere Auswertungsfunktionen angegeben werden. Soll-Variablen hängen sehr stark vom jeweiligen Subprojekt ab, sie sollen deshalb subprojektspezifisch definiert werden.
- Unterschritte: Unterschritte sind komposite Zustände, die durch keine Attribute beschrieben werden. Sie dienen lediglich der strukturierten Darstellung von Testfällen. Theoretisch können Unterschritte in weitere Unterschritte strukturiert werden, praktisch sollten aber gewisse Grenzen beachtet werden, damit die Übersichtlichkeit bewahrt wird.

Darstellung von Aktionen

Wie oben erwähnt, werden Aktionen, die unmittelbar von den Verkehrsteilnehmern ausgeführt werden, durch Aktionszustände bzw. als deren do-Actions dargestellt. Im Weiteren werden einzelne „Konstellationen“ von diesen erläutert.

- Absolut globale Aktionen: Die absolut globalen Aktionen werden als „do-Actions“ der Aktionszustände modelliert. Diese werden dann aktiviert, wenn der entsprechende Zustand betreten wird. Dies geschieht wiederum, wenn die in den Zustand führende Transition schaltet. Um zeitbezogene absolut globale Aktionen darstellen zu können, muss das Konzept einer globalen Uhr eingeführt werden. Diese wird mit „clock“ bezeichnet. Für jede Uhr müssen die Genauigkeit sowie der Schritt bestimmt werden. Die Uhr wird als ein Ereignis an der Transition modelliert. Dabei wird die Anzahl der „ticks“ angegeben, die seit Startzeitpunkt der Fahrszene vergangen sind (Abb. 5.6). Der in der Abbildung 5.6

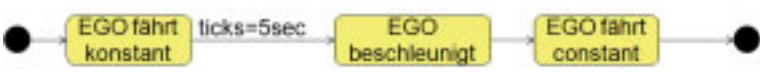


Bild 5.6: Statechart: Das Uhrkonzept

dargestellte Sachverhalt hat folgende Bedeutung: Exakt nachdem 5 Sekunden seit dem Start der Fahrszene vergangen sind, muss die konstante Fahrt beendet worden sein. Dadurch, dass eine Transition 0 Sekunden dauert, wird der Zustand „EGO beschleunigt“ 5 Sekunden nach dem Beginn der Fahrszene betreten.

Für die örtlichen absolut globalen Aktionen muss für die in den entsprechenden Aktionszustand führende Transition ein auslösendes Ereignis „position(x, y) ist erreicht“ eingeführt werden (abgekürzt als position(x,y) gekennzeichnet, Abb. 5.7). Was die Angabe der Akti-

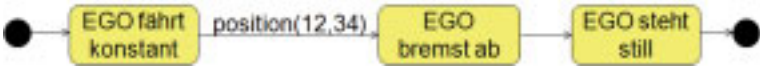


Bild 5.7: Beispiel für positionsabhängige Ereignisse

onsdauer (do-Action) betrifft, so wird diese durch einen Timer spezifiziert. Der Timer wird durch ein Ereignis modelliert (siehe Abbildung 5.8). Durch die Semantik der Statecharts ist eine explizite Zuordnung des Timers einem Zustand nicht nötig. Beim Betreten des Zustands wird er durch eine „entry-Action“ ausgelöst und beim Verlassen durch eine „exit-Action“ gestoppt und es wird gleichzeitig ein Event produziert „Timer=5sec“. Damit kann die Transition schalten. In der Abbildung 5.8 wird folgender Sachverhalt zum Ausdruck

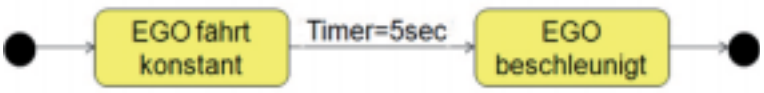


Bild 5.8: Beispiel für die Angabe der Aktionsdauer

gebracht: Nachdem das EGO-Fahrzeug 5 Sekunden konstant gefahren ist, beschleunigt es. Schließlich muss erwähnt werden, dass das Uhr-Konzept die relative Zeit (zum Beginn der Fahrszene) zum Ausdruck bringt, während ein Timer zur Darstellung absoluter Zeit dient. Ein Timer wird, wie bereits erwähnt, durch eine Entry-Action ausgelöst und durch eine Exit-Action beendet.

- Aktionen, die relativ zu anderen Aktionen geschehen: Zur Modellierung von Aktionen, die von anderen Aktionen abhängen, ist es notwendig, das Konzept einer Verzögerung einzuführen. [Har87] schlägt dazu das Konzept eines „Timeouts“ vor. Dieser besitzt die Form „timeout(event, number)“, wobei „event“ den Stimulus bezeichnet, der dann eintritt,

wenn seit dem Eintreten des Stimulus eine bestimmte Anzahl („number“) an „ticks“ vergangen ist. In der Terminologie der Statecharts ist „timeout“ ebenfalls ein „event“. Mit dem Konzept des „timeout“ ist man also imstande, Zustände darzustellen, welche zeitversetzt relativ zum Betreten eines anderen Zustands betreten werden (Abb. 5.9). Die Aktionen, die

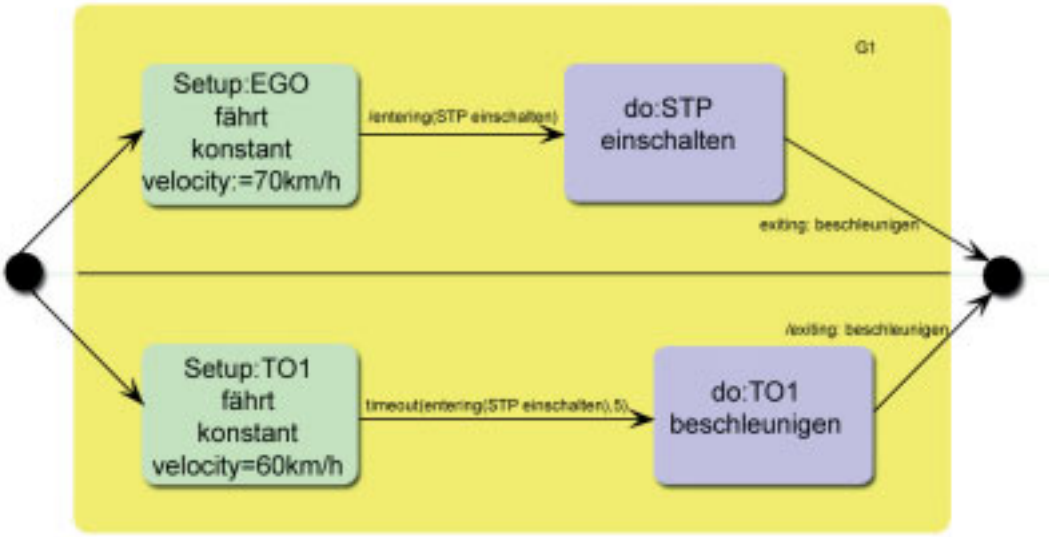


Bild 5.9: Statechart: Timeout-Konzept

nach dem Ende einer anderen Aktion beginnen, mit oder ohne Verzögerung, sind jeweils in den Abbildungen 5.10 und 5.11 zu sehen. In der Abbildung 5.10 ist folgendes dargestellt. Sobald das Ereignis „exiting(EGO schert ein)“ eintritt, wird ein „Timeout“ für 5 Sekunden ausgelöst. Dieses verzögert das Betreten des Zustands „EGO beschleunigt“ um 5 Sekunden. Mit anderen Worten wird die schaltende Transition verzögert. Das Ereignis „exiting(EGO schert ein)“ wird beim Verlassen des Zustands „EGO schert ein“ durch die „exit-Action“ produziert.

Um echte Nebenläufigkeit darzustellen, schlägt [Har87] die in Abbildung 5.12 dargestellte Notation vor. Daraus ist ersichtlich, dass, sobald die in einen Aktionszustand führende Transition die („benachrichtigende“) Aktion „entering(STP einschalten)“ ausführt, sofort die zweite Transition schaltet, die auf das gleichnamige Ereignis „entering(STP einschalten)“ wartet. Die Aktion „entering(STP einschalten)“ produziert in diesem Fall das gleichnamige Ereignis, dessen Eintreten die zweite Transition auslöst. Der Aktionszustand „STP einschalten“ führt dann die eigentliche do-Einschaltfunktion aus.

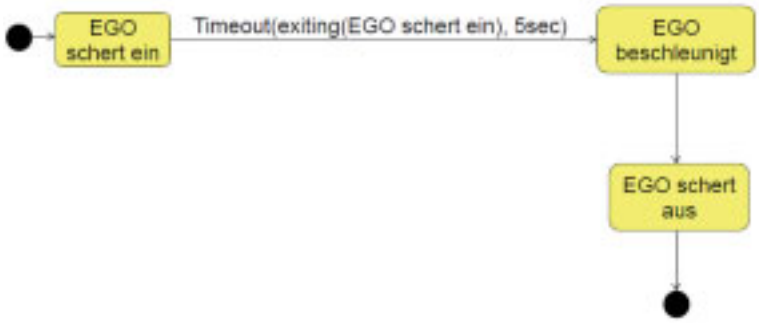


Bild 5.10: Verzögerte sequentielle Ausführung von Aktionen

Bild 5.11: Sequentielle Ausführung von Aktionen

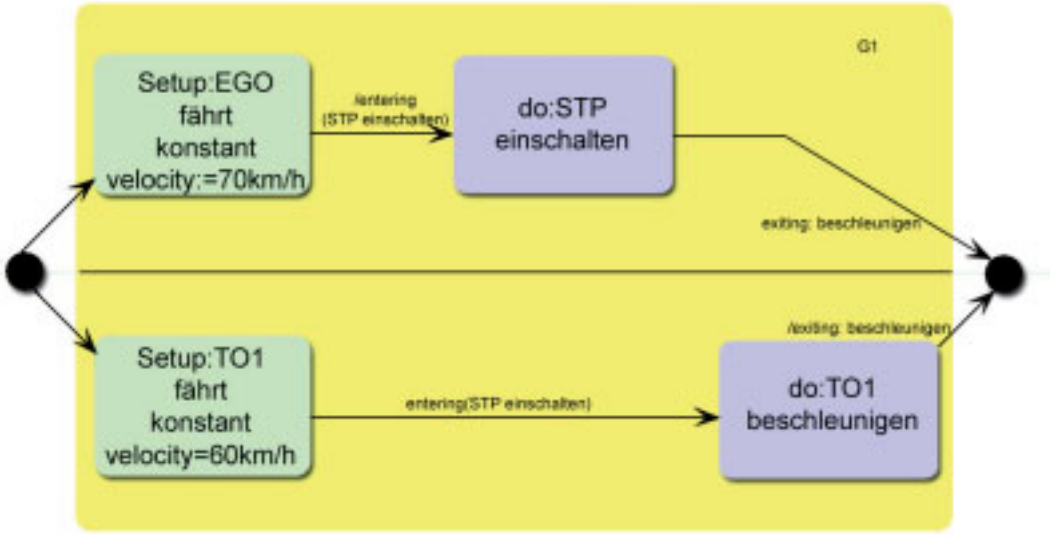


Bild 5.12: Darstellung echter Nebenläufigkeit

- Aktionen, die auf den Abstand relativ zum anderen Fahrzeug bezogen sind: Zur Modellierung dieses Sachverhalts soll das Ereignis „Abstand zum Fahrzeug X ist erreicht worden“ eingeführt werden (Abbildung 5.13).

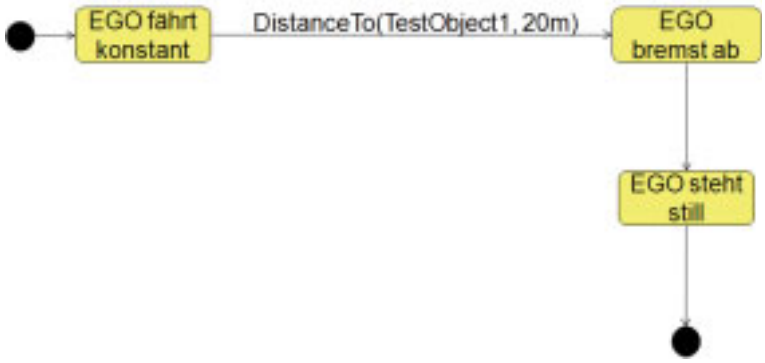


Bild 5.13: Abstandsbezogene Darstellung von Aktionen

- Modellierung von zyklischen Vorgängen: Um die zyklischen Vorgänge modellieren zu können, bieten die Statecharts das Konstrukt „ChoicePoint“, also Auswahlpunkt, für welchen eine Bedingung definiert werden kann. Somit kann man beispielsweise While-Schleifen definieren.

Zusammenfassend lässt sich feststellen, dass die Statecharts als formale Modellierungsmethode nach der Vorauswahl am Besten zur formalen Modellierung von Fahrszenen geeignet zu sein scheinen. Nun muss das Ergebnis der Vorauswahl anhand einer Fallstudie bestätigt werden.

5.1.2.2 Die Fallstudie

Für die Domäne von Fahrer-Assistenzsystemen ist eine Fallstudie durchgeführt worden. Dazu wird folgende Vorgehensweise vorgeschlagen. Es wird der Bereich der realen Messfahrten betrachtet. Für diesen Bereich wurden bei Gesprächen mit den Entwicklern zwei Haupt-Anwendungsfälle ermittelt (Abb. 5.14). Bei einem geht man davon aus, dass das komplette Fahrszenario vor der Messfahrt modelliert wird. Dazu gehören nicht nur die eigentlichen Testaktionen (Fahrmanöver), sondern auch die Beschreibung der Strecke, der Soll-Werte, des Testequipments (z.B. Pilonen und andere stehende Objekte) und der Fahrzeugkonfiguration. Ein solcher Anwendungsfall tritt dann ein, wenn das Fahrszenario beispielsweise auf dem Prüfgelände stattfindet und somit alle zur Beschreibung des Fahrszenarios notwendigen Informationen von Vornherein bekannt sind.

Der zweite Anwendungsfall tritt dann ein, wenn bestimmte Informationen, wie Testaktionen, Streckenverläufe etc. von Vornherein nicht bekannt sind. Das ist dann der Fall, wenn beispielsweise der Entwickler gezielt Stausituationen im Autobahnverkehr aufsucht. Dabei kann er natürlich nicht schon vor der Fahrt wissen, wie der Stau aussehen wird. Vielmehr setzt er sich ins Versuchsfahrzeug und fährt gezielt in die Orte auf der Autobahn, wo er einen Stau vermutet. Beispielsweise ist die Wahrscheinlichkeit hoch, dass man freitags ab 16:00 in München stadtauswärts in den Stau

kommt. Bei diesem Anwendungsfall nimmt der Fahrer die Fahrszenarien auf, stellt sie anschließend in die Szenariendatenbank ein und erst dann beschreibt er die interessanten Fahrszenen mit den Statecharts. Dabei orientiert er sich an den Videodaten, die zu Dokumentationszwecken in der Datenbank abgelegt werden. Diese Szenenbeschreibungen sind im Vergleich zum ersten Anwendungsfall wesentlich kompakter, da dort wirklich nur das beschrieben wird, was der Entwickler im Videobild sieht, also zum Beispiel Testaktionen und der Streckenverlauf.

Wichtig zu erwähnen ist es, dass beim zweiten Anwendungsfall zunächst keine Soll-Werte beschrieben werden, sondern nur die beobachteten Fahrszenen. Die Definition der Soll-Werte soll in diesem Fall in einem separaten Schritt geschehen, also bei der Verfeinerung der Fahrszene für ein bestimmtes Subprojekt. Zusammenfassend sind die beiden gerade beschriebenen Anwendungsfälle in der Abbildung 5.14 dargestellt. Wichtig zu erwähnen ist, dass alle behandelten

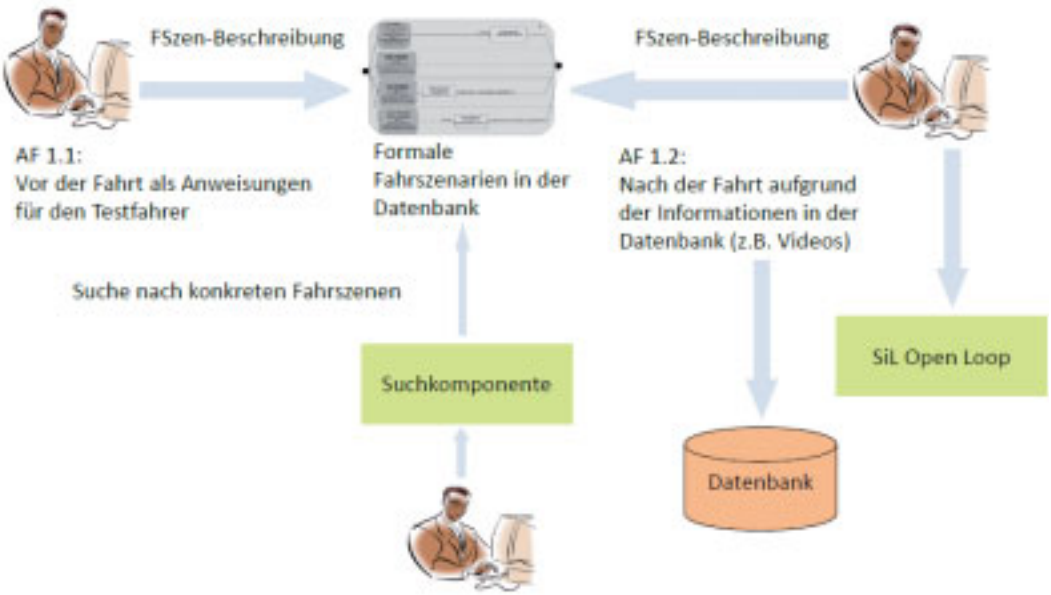


Bild 5.14: Fallstudie: Haupt-Anwendungsfälle

Evaluierungsbeispiele von den Entwicklern selbst bei Gesprächen vorgeschlagen wurden. Sie stellen somit die realen Sachverhalte dar und sind keineswegs erdacht!

Beschreibung der Fahrszenen vor der Fahrt im Bereich der Umfeldwahrnehmung

Bei diesem Fahrszenario wird die Funktionalität diverser verbauter Sensoren überprüft. Dabei fährt das EGO-Fahrzeug mit einer konstanten Geschwindigkeit von 20 km/h 10 Meter hinter dem Vorderfahrzeug. Das Vorderfahrzeug (Synonym zu Fremdfahrzeug oder abgekürzt FF) fährt

ebenfalls mit einer konstanten Geschwindigkeit. Das EGO-Fahrzeug fängt an, „Schlangenlinie“ zu fahren, und zwar in regelmäßigen Zeitabständen, z.B. 5 sec Schlangenlinie fahren, dann normal fahren, dann wieder 5 sec Schlangenlinie. Der Vorgang soll insgesamt 5 mal wiederholt werden. Überprüft werden muss, ob die aufgezeichneten Daten komplett sind. In der Abbildung 5.15 ist das Szenario mit der Statechart-Methode dargestellt. Als Voraussetzung zur Ausführung des Fahrs-

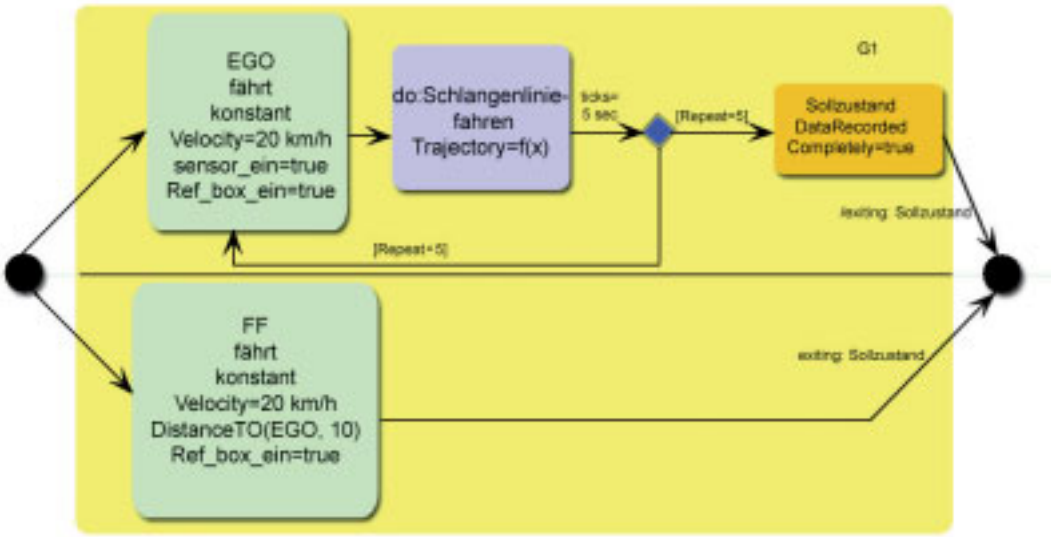


Bild 5.15: das Szenario Umfeldwahrnehmung

zenarios ist es notwendig, dass beim EGO-Fahrzeug die Sensoren eingeschaltet sind und auch die Referenzbox, die die GPS Daten zum späteren Soll-Ist Vergleich aufnimmt. Selbstverständlich muss das EGO-Fahrzeug mit einer bestimmten Geschwindigkeit fahren, bevor die eigentliche Testaktion (=do-Aktion) beginnt. Für das Fremdfahrzeug muss natürlich die Geschwindigkeit angegeben werden können, die Distanz zum EGO-Fahrzeug sowie die Angaben darüber, ob die Referenzbox eingeschaltet wurde. Die eigentliche Testaktion besitzt eine bestimmte Dauer, die Anzahl der Wiederholungen sowie eine bestimmte Trajektorie. Die Trajektorie kann sicherlich präzise mittels einer mathematischen Funktion beschrieben werden aber auch durch einen bestimmten Begriff, was gerade für einen Testfahrer am Verständlichsten ist. Bei den Soll-Zuständen ist jeweils eine boolesche Variable abgebildet, die angibt, dass die Daten komplett aufgezeichnet werden müssen. Es hat sich bei der Diskussion herausgestellt, dass ebenfalls globale Attribute für die komplette Beschreibung des Fahrszenarios angegeben werden müssen. In diesem konkreten Fall gehören zu diesen solche Attribute wie Koordinatensystem, oder auch die aufzuzeichnenden Daten (z.B. CAN-Daten und GPS-Daten). Ebenfalls können mit den globalen Attributen solche

Informationen abgebildet werden wie Anzahl der Pilonen, die zur Testdurchführung notwendig ist, Streckenverlauf, Fahrbahnzustände, verbaute Sensorik u.ä. Generell ist anzumerken, dass aus allen Evaluierungsbeispielen des Anwendungsfalls 1.1 (Abb. 5.14) normalerweise elektronische Dokumente erzeugt werden sollen, die der Testfahrer als Anweisung zu dem durchzuführenden Szenario bekommt. Ein beispielhaftes Excel-Stylesheet wird im Abschnitt über die Adaptive Cruise Control (ACC) Funktion dargestellt.

Warnfunktion

Bei diesem Fahrszenario, das zu AF 1.1 Abb. 5.14 gehört und in der Abbildung 5.16 dargestellt ist, wird Warnfunktionalität einer bestimmten Funktion überprüft. Hierbei fährt das EGO-Fahrzeug konstant mit v=140 km/h. Das Vorderfahrzeug fährt ebenfalls konstant mit v=100 km/h. Das EGO-Fahrzeug kommt zu nah an das Vorderfahrzeug (:=Target Object:=TO). EGO-Fahrzeug gibt eine Warnung an den Fahrer. Es soll getestet werden, ob die Fahrerwarnung erfolgt ist. Die Abkürzung „Event A“ steht für „EGO fährt zu nah an das Vorderfahrzeug (Time To Collision =1.4 sec)“. Auch in diesem Evaluierungsbeispiel müssen die beteiligten Fahrzeuge erst eine bestimmte Geschwindigkeit erreicht haben bevor die eigentliche Testaktion stattfinden darf. So muss das EGO-Fahrzeug mit 140 km/h fahren und das Testobjekt mit 100 km/h. Sobald eine bestimmte Zeit bis zur Kollision erreicht wird, findet die eigentliche Testaktion statt: Die Fahrerwarnung. Im Sollzustand wird das Bit definiert, das gesetzt sein muss, wenn die Fahrerwarnung erfolgt ist. Um die Korrektheit der Notation zu bewahren, soll noch spezifiziert werden, wann der Zustand „TO fährt konstant“ verlassen werden soll. Die geschieht dann, wenn der Zustand „Sollzustand“ verlassen wird. Die gewählte Notation entspricht der, die vom Professor Harel im Artikel über die Statecharts vorgeschlagen wurde [Har87].

Adaptive Cruise Control

Die zu modellierende Fahrszene gehört zum AF 1.1 Abb. 5.14 und ist in der Abbildung 5.17 zu sehen. Das Ergebnis der Modellierung für die Fahrszene „ACC“ ist in der Abbildung 5.18 dargestellt. Das EGO-Fahrzeug fährt in der zweiten Spur geregelt ACC mit einer Setz- Geschwindigkeit von 140 km/h. Das Einscherfahrzeug fährt in der Spur 1 ebenfalls mit einer Geschwindigkeit von 140 km/h. Die Zeitlücke zum Zielobjekt (In der Abbildung 5.18 mit TO gekennzeichnet) beträgt 1.7 Sekunden. Das Zielobjekt fährt in der zweiten Spur vor dem EGO-Fahrzeug (RelativePosition=5) mit einer Geschwindigkeit von 100 km/h. Schließlich ist mit G 1 der globale Zustand gekennzeichnet, während mit G 1.1 ein kompositer Subzustand von G 1 notiert ist. Um die Klarheit über die Bezeichnungen zu schaffen, ist in der Abbildung 5.19 die Notation zur

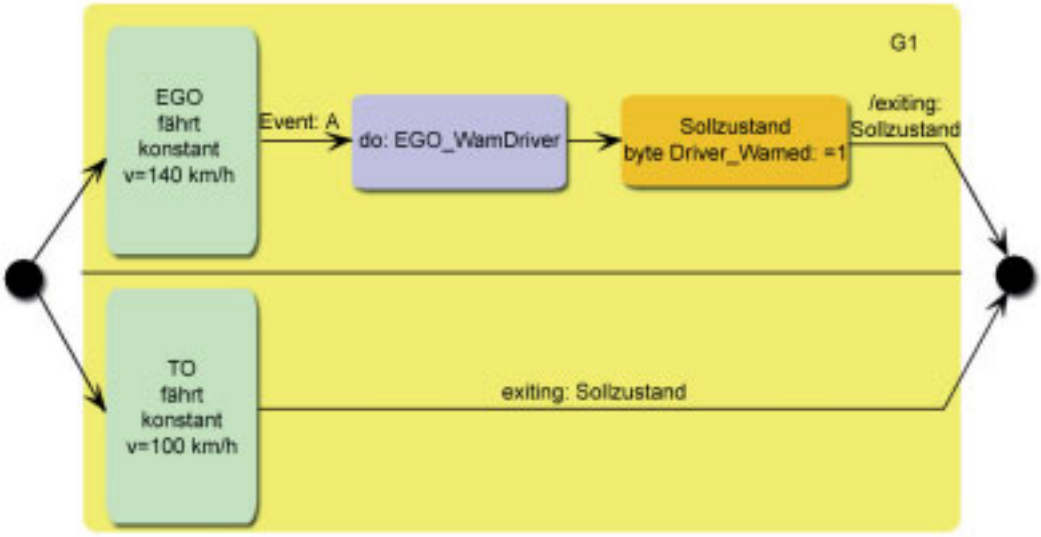


Bild 5.16: das Szenario Warnfunktion

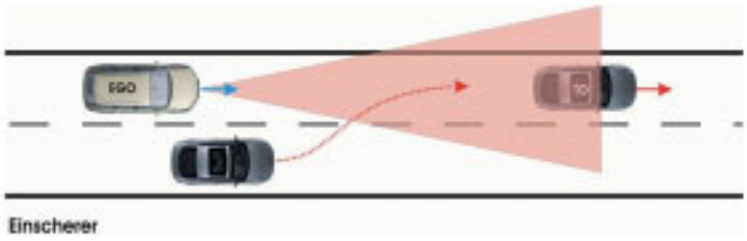


Bild 5.17: das Szenario Adaptive Cruise Control

Darstellung relativer Positionen der Fremdfahrzeuge zum EGO-Fahrzeug dargestellt. Ebenfalls ist dort die Semantik des kompositen Attributes „DistanceTo“ erklärt. Sobald die Y-Distanz des Einscherfahrzeugs zum EGO 1.5 Meter erreicht wird (RelativePositionToVehicle1), beschleunigt dieser auf 150 km/h und schert gleichzeitig ein auf die relative Position 5. Zur Beschreibung der Einscher-Trajektorie kann man eine Funktion angeben. Das Ereignis „RelativePositionToVehicle1“ besitzt folgende Attribute: („Distance_Y“, RelativePosition“). Das Attribut „RelativePosition“ wird dazu benötigt, um das Fahrzeug anzugeben, relativ zu welchem die Y-Distanz erreicht wird. Sobald die Distanz vom EGO-Fahrzeug zum eingescherten Fahrzeug auf der Position 5 10 Meter erreicht, bremst das erstere ab auf 100 km/h und der Negativbeschleunigung von $2m/s^2$. Das Ereignis „RelativePositionToVehicle“ wird durch die Attribute „Distance_X=10“ und „RelativePosition“=5 beschrieben. Schließlich wird auch der Soll-Zustand des EGO-Fahrzeugs

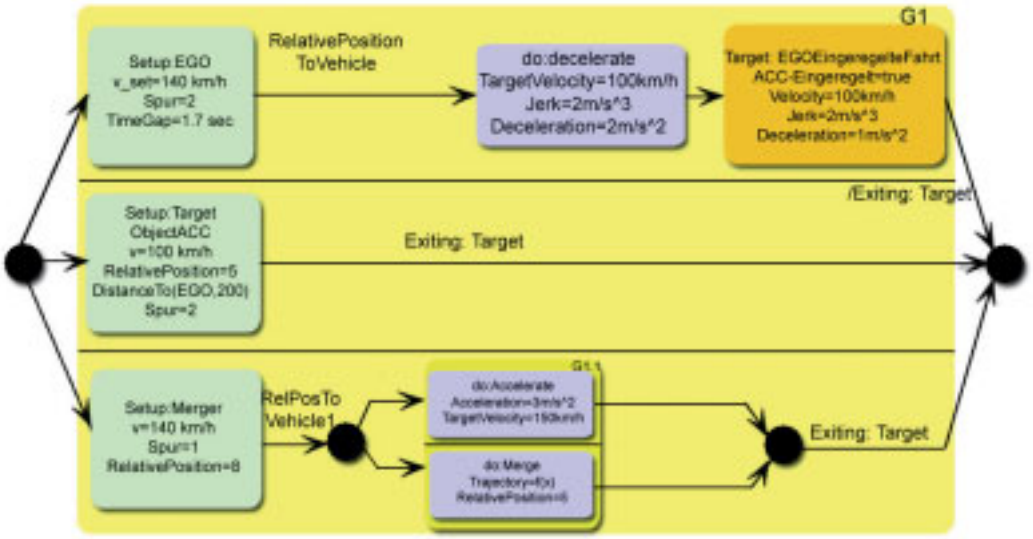


Bild 5.18: das Szenario Adaptive Cruise Control (ACC)

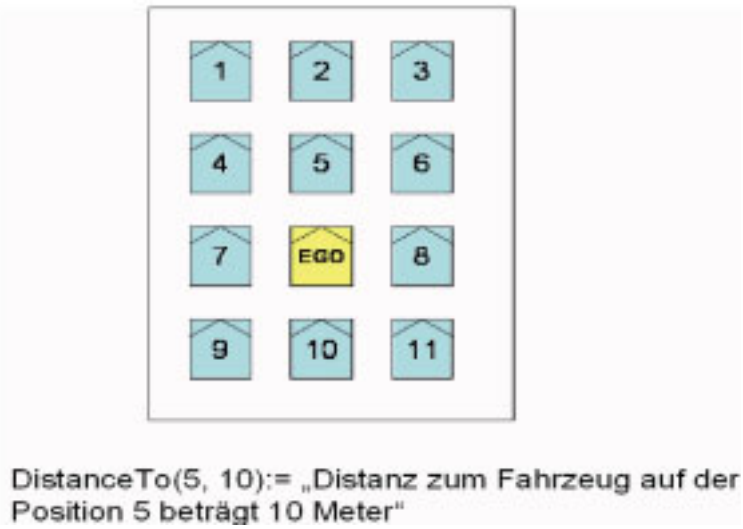


Bild 5.19: Notation von relativen Fahrzeugpositionen

beschrieben. Dort sind die Soll-Werte für die zu überprüfenden Größen dagestellt (Abbildung 5.18). Abschließend ist in der Abbildung 5.20 der Entwurf eines EXCEL-Stylesheets für das vorliegende Evaluierungsbeispiel dargestellt, welches aus der formalen Beschreibung generiert werden soll. Dieser Entwurf diente als Grundlage zur Besprechung mit dem Testfahrer.

Ausgangssituation	Spalte1	Spalte2
EGO		
Zustandsbeschreibung	fährt eingeregelt ACC	
SET-Geschwindigkeit	140 km/h	
Zeitlücke(Fahrstufe=3)	1.7 sec	
Spurbezeichnung	Mittlere Spur	
Zielobjekt		
Zustandsbeschreibung	fährt konstant	
Geschwindigkeit	100 km/h	
Position relativ zum EGO	Vor dem EGO	
X-Distanz zum EGO	200m	
Einscherer		
Zustandsbeschreibung	fährt konstant	
Geschwindigkeit	140 km/h	
Position relativ zum EGO	Rechte Nebenspur, vorn.	
Bedingung	**Aktion**	
Die X-Distanz von Einscherer zum Zielobjekt erreicht 10 meter	Aktionsbeschreibung	Einscherer beschleunigt
	Beschleunigung	3m/s^2
	Zielgeschwindigkeit von Einscherer	150 km/h
	Aktionsbeschreibung	Einscherer schert nach links ein
Die Y-Distanz vom Einscherer zu EGO erreicht 1.5 Meter	Aktionsbeschreibung	EGO bremst ab
	Zielgeschwindigkeit	100 km/h
	Beschleunigung	2m/s^2

Bild 5.20: Mögliche Tabellenstruktur für ACC

Dichtes Auffahren von Hinten

Bei dieser Fahrszene, welche zum AF 1.2 Abb. 5.14 gehört, fährt das EGO in der rechten Spur mit einer Geschwindigkeit von 80 km/h. Das Vorderfahrzeug fährt mit einer Geschwindigkeit von 60 km/h. Das EGO-Fahrzeug fährt dicht auf und bremst anschließend in den Stillstand ab. Im nächsten Schritt bremst das Vorderfahrzeug ebenfalls in den Stillstand ab. Die mit Statechart modellierte Fahrszene ist in der Abbidlung 5.21 dargestellt. Die Abkürzung VF in der Abbildung 5.21 steht für Vorderfahrzeug. Das Attribut RelativePosition=2 bedeutet, dass das Vorderfahrzeug vor dem EGO-Fahrzeug fährt. Mögliche globale Beschreibungsattribute wären in diesem Fall: Streckentyp, Spurenanzahl, Spurmarkierung, Streckenverlauf.

Einscherer von links

In diesem Fall (gehört zu AF 1.2 Abb. 5.14) fahren EGO und Vorderfahrzeug in der Mittelspur (beide 60 km/h). Einscherer fährt auf der linken Spur. Einscherer bremst ab und schert gleichzeitig vor dem EGO-Fahrzeug von links ein (Abbildung 5.22).

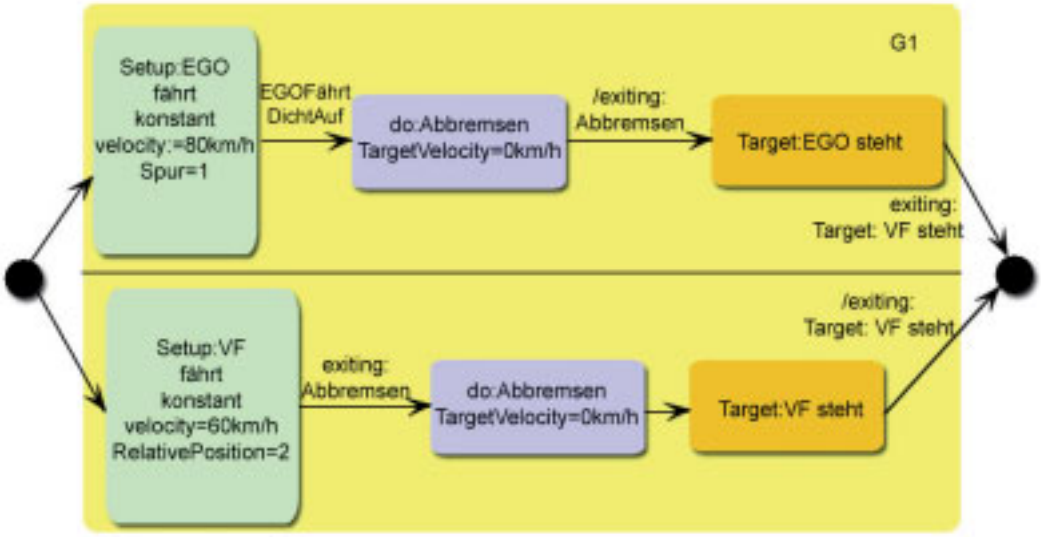


Bild 5.21: das Szenario dichtes Auffahren von hinten

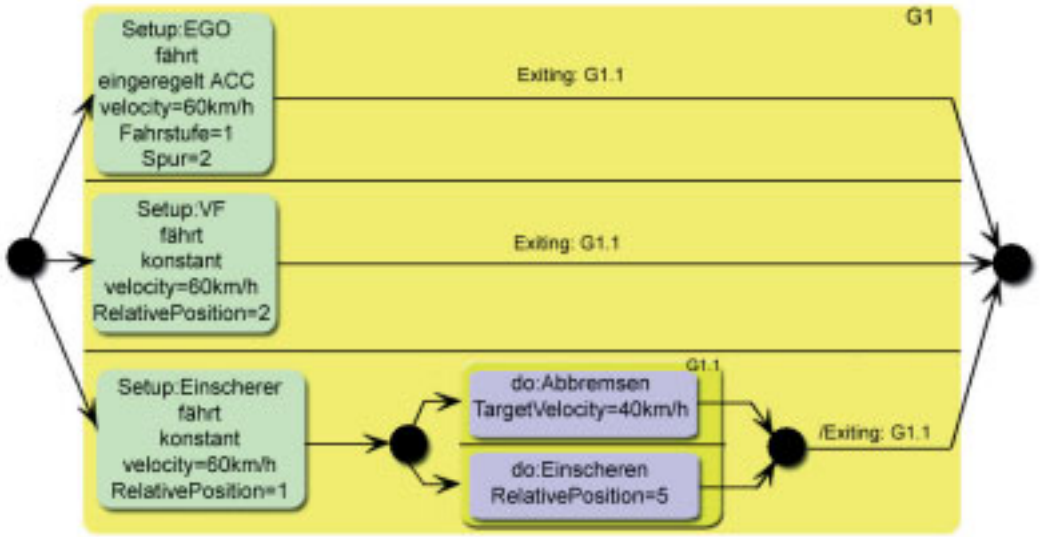


Bild 5.22: das Szenario Einscherer von links

Baustellenbeginn

In dieser Fahrszene, welche zum AF 1.2 Abb. 5.14 gehört, findet der Übergang von der normalen Autobahnstrecke zum Baustellenbeginn statt. Der Übergang selbst wird als ein Ereignis modelliert (Abbildung 5.23). Folgende globale Attribute können hier von Interesse sein: Streckentyp, Spurenanzahl, Spurmarkierung, Wetterzustand, Fahrbahnzustand, Streckenabschnitt-Verlauf.

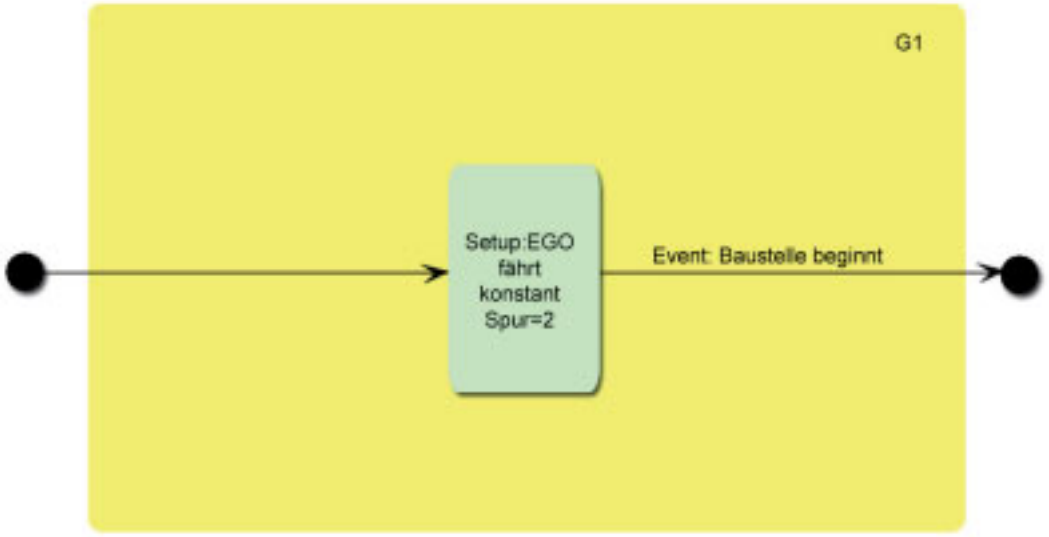


Bild 5.23: das Szenario Baustellenbeginn

Zusammenfassend lässt sich feststellen, dass die im Nachhinein beschriebenen Fahrszenen eine geringere Komplexität im Vergleich zu den vor der Messfahrt beschriebenen aufweisen. Unter Komplexität ist hier die Anzahl an einzugebenden Attributen zu verstehen. Gleichzeitig kann man die im Nachhinein beschriebenen Fahrszenen wesentlich verkomplizieren, indem eine längere Abfolge von Fahrmanövern beschrieben wird. So könnte man zum Beispiel die Fahrszene aus Abbildung 5.22 erweitern, indem man den nach dem Einscheren folgenden Bremsvorgang des EGO-Fahrzeugs modelliert.

5.1.2.3 Prototyp

Zur ersten Implementierung wurden die Windows Forms (WF) [Mic] von Microsoft verwendet mit einer Zusatzbibliothek namens „AddFlow“ [Las10], welche die Programmierung von Flussdiagrammen, insbesondere von Statecharts erlaubt. Alle Beschreibungselemente wurden festkodiert, d.h. es wurde eine fixe Menge an möglichen Zuständen und ihren Beschreibungsattributen gewählt. Es geschah ebenfalls noch keine Unterscheidung in projektspezifische und generische Attribute. Was die Persistierung betrifft, so wurde die Möglichkeit eines einfachen XML-Exports geschaffen, sodass bereits erzeugte Fahrszenenbeschreibung leicht wieder importiert werden können. Ein Screenshot der graphischen Oberfläche ist in der Abbildung 5.24 zu sehen.

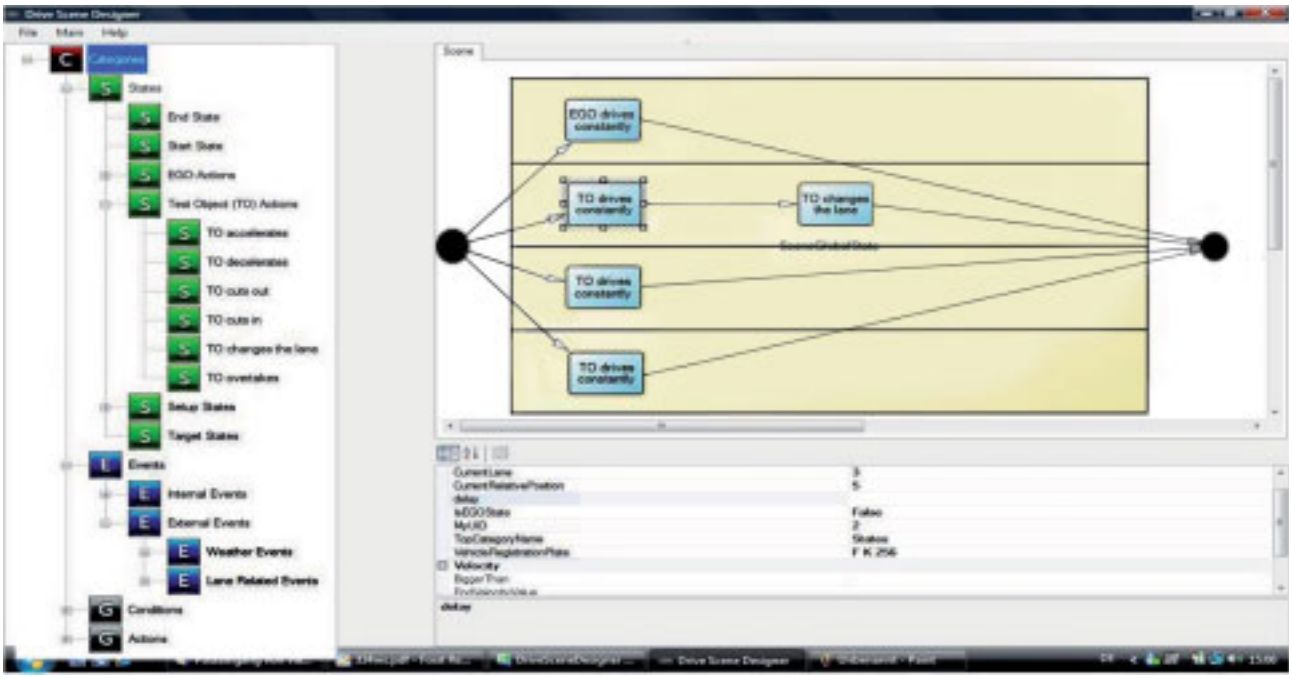


Bild 5.24: prototypische Oberflächenimplementierung

5.1.3 Methodenbewertung

Das erste Feedback von den Entwicklern hat folgende Ergebnisse gebracht:

- Keine gravierenden Modellierungsfehler: Es können alle für die Darstellung von Fahrszenarien relevanten Sachverhalte mit Hilfe von Statechart modelliert werden. Die Entwickler haben sich zum einen für die Möglichkeit interessiert, dass ein Fahrzeug während eines Fahrmanövers mehrere Aktionen gleichzeitig ausführen kann wie zum Beispiel „Spurwechsel“ und „Beschleunigung“. Dies kann mit einem kompositen Subzustand modelliert werden. In diesem werden dann die parallel stattfindenden Aktionen modelliert (Abbildung 5.22). Eine andere Anmerkung betraf die Möglichkeit Trajektorien eines Fahrzeugs darstellen zu können. Dies ist ebenfalls möglich durch die Angabe eines Beschreibungsattributs „Trajectory“ im jeweiligen Fahrzeugzustand, der ein Fahrmanöver beschreibt. Die Trajektorie selbst kann mit Hilfe einer mathematischen Funktion beschrieben werden.

- Damit Missverständnisse und Analogien zwischen kompositen Subzuständen und Fahrspuren vermieden werden, hat sich die Notwendigkeit herausgestellt, in einem kompositen Subzustand nur die Aktionen zu modellieren, die einen einzigen Verkehrsteilnehmer betreffen. Es muss also vom Werkzeug ein Assistent zur Verfügung gestellt werden, welcher

zur Erstellungszeit die Richtigkeit der erstellten Fahrszenenbeschreibung überprüft und den Entwickler über eventuelle Modellierungsfehler informiert.

- Es wurden gemäß dem Vorgehensmodell zwei Iterationen benötigt, denn beim Feedback in der ersten Iteration wurde festgestellt, dass im Wörterbuch eindeutig festgelegt werden muss, was mit Hilfe von Bedingungen (engl. „Guards“) modelliert werden soll. Dabei wurde zunächst folgender Vorschlag erarbeitet: Alles, was die Auswahl bei den „ChoicePoints“ also bei den Auswahlpunkten betrifft, wird durch Guards modelliert. Alles Andere wird mit Hilfe von Ereignissen modelliert.

- Was das allgemeine Feedback betrifft, so haben sich die Entwickler noch die Angabe der Zeitachse in der graphischen Oberfläche gewünscht sowie die Benennung der kompositen Subzustände nach Verkehrsteilnehmern (also gewöhnlich Fahrzeugen). Dies wurde durch Erweiterung der graphischen Oberfläche gelöst.

Abschließend lässt sich feststellen, dass sowohl die Ergebnisse der Fallstudie als auch die erfolgte Methodenbewertung keine Sachverhalte haben ermitteln können, die den Einsatz von Statecharts unmöglich machen würden. Somit werden Statecharts endgültig als Szenarienmodellierungsmethode für die FAS-Domäne festgelegt. Nun kann man zum nächsten Schritt im Vorgehensmodell übergehen.

5.2 Definition eines geeigneten Metamodells

Zur Erinnerung ist in der Abbildung 5.25 der allgemeine Teil des Metamodells (Basis-Metamodell) zur Definition von Testspezifikationen gezeigt. Dieser Teil dient als Ausgangsbasis für jegliche szenariobasierte domänenspezifische testspezifikationsbezogene Metamodelldefinition. Dies bedeutet insbesondere, dass die dort definierten Elemente eine minimal notwendige Basis bilden, sollte man sich zum Einsatz des in dieser Arbeit vorgestellten Vorgehensmodells entscheiden. Von diesen Basiselementen müssen auf M2-Ebene domänenspezifische Modellierungselemente abgeleitet werden. Somit wird die Allgemeingültigkeit des Konzepts nochmals unterstrichen, denn das gezeigte Basis-Metamodell kann in mehreren komplett unterschiedlichen Domänen angewandt werden. Im Folgenden wird nun anhand des im Abbildung 5.26 vorgestellten Schemas das Metamodell für die Statechart-basierte Testspezifikationsbeschreibung für die FAS-Domäne definiert (Domänen-Metamodell) sowie das zugehörige partielle Modell. Betrachtet werden nicht alle Beschreibungselemente, sondern exemplarisch nur Zustände sowie der allgemeine Aufbau der Testspezifikation, denn alle sonstigen Beschreibungselemente wie Aktionen, Transitionen,

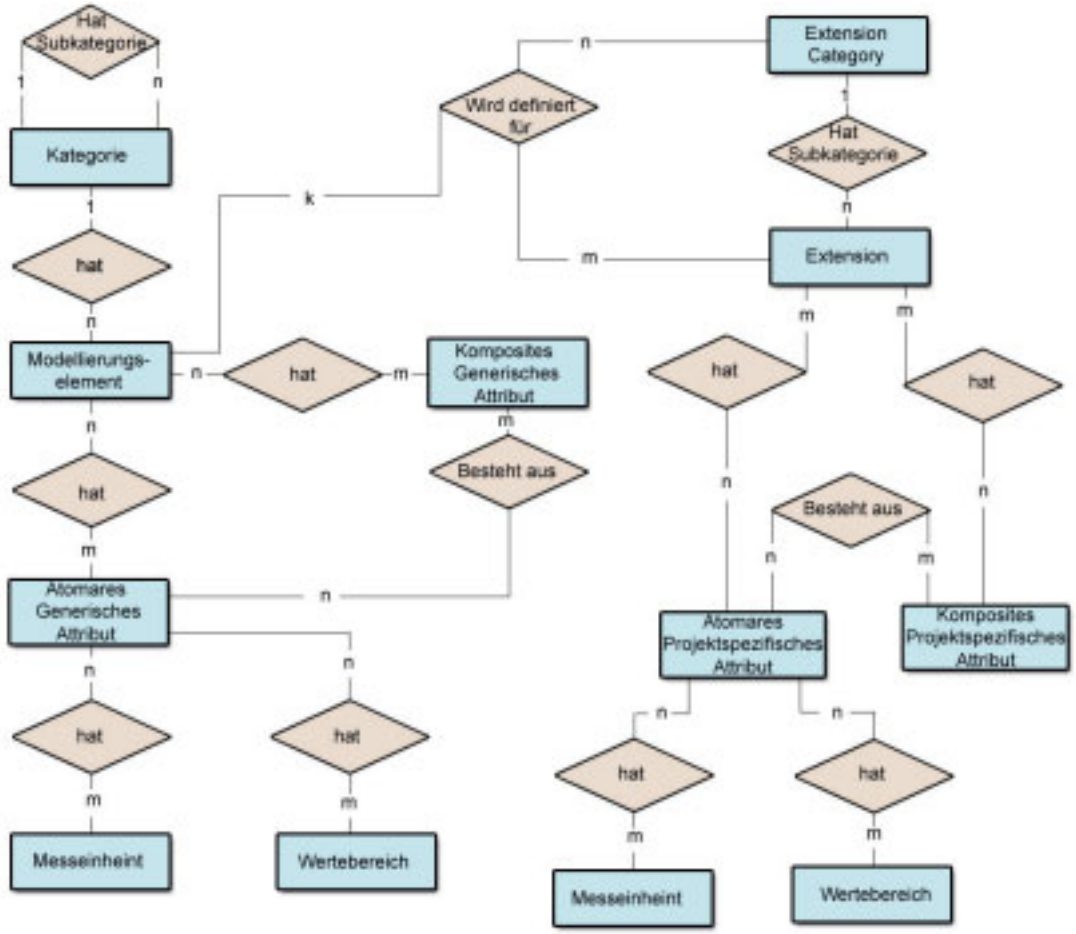


Bild 5.25: Der allgemeingültige Teil des Metamodells zur Definition von Testspezifikationen

Ereignisse und Bedingungen haben einen zu den Zuständen ähnlichen Aufbau. Die farbliche Kodierung von Beschreibungselementen aus Abb. 5.26 wird für die nachfolgenden Abbildungen übernommen.

5.2.1 Definition von Modellierungselementen

Darstellung von Zuständen

In der Abbildung 5.27 ist die Zustandshierarchie zu sehen. Das Element „Zustand“ ist ein Modellierungselement, deshalb wird es von der Klasse „Modellierungselement“ abgeleitet. Zustandskategorie ist eine Kategorie. Aus diesem Grund wird sie von der Klasse „Kategorie“ abgeleitet.

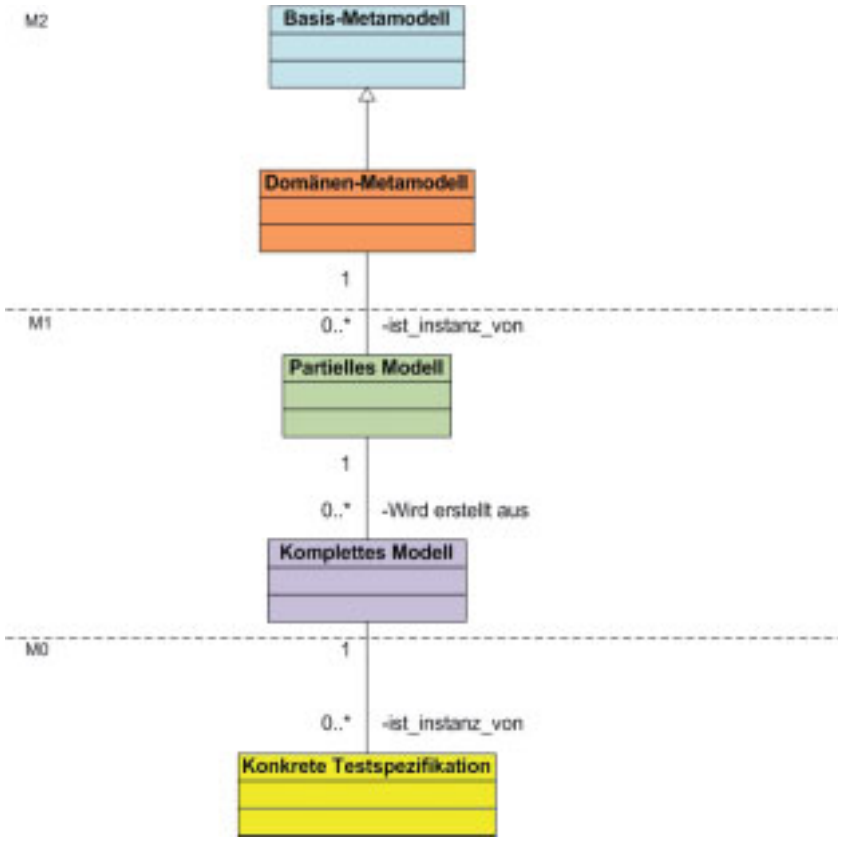


Bild 5.26: Erinnerung: Gesamte Modellierungshierarchie

Das sind die einzigen Metaklassen, welche speziell für das Element „Zustand“ abgeleitet werden müssen. Was die Elemente „Erweiterungskategorie“, „Erweiterung“, „Atomares Generisches Attribut“, „Komposites Generisches Attribut“, „Atomares Projektspezifisches Attribut“ und schließlich „Komposites Projektspezifisches Attribut“ (Abbildungen 5.28 und 5.29) betrifft, so werden diese hier nur einmal für die FAS-Domäne abgeleitet. Denn man geht davon aus, dass beispielsweise die Attribute in der vorliegenden Domäne für sämtliche Beschreibungselemente den gleichen Aufbau haben werden: Name, Messeinheit und Wertebereich. Grundsätzlich werden Zustände in technische und so genannte „NonTechnical“ unterteilt. Zu den technischen gehören laut Spezifikation von Statecharts [Obj09] [Har87] Anfangs- und Endzustände, Auswahl -und Verbindungszustände. Die Verbindungszustände dienen dazu, die Transitionen zusammenzuführen oder sie zu verzweigen (engl. branch). Die „NonTechnical“ unterteilen sich in atomare und zusammengesetzte Zustände. Was die Beziehungen zwischen den abgeleiteten Elementen betrifft, so werden diese von Basis-Metamodell geerbt. Sie werden aus diesem Grund nicht nocheinmal in der Abbildung eingezeichnet. Die atomaren Zustände, wie der Name schon sagt, gliedern sich in Aktionszustände

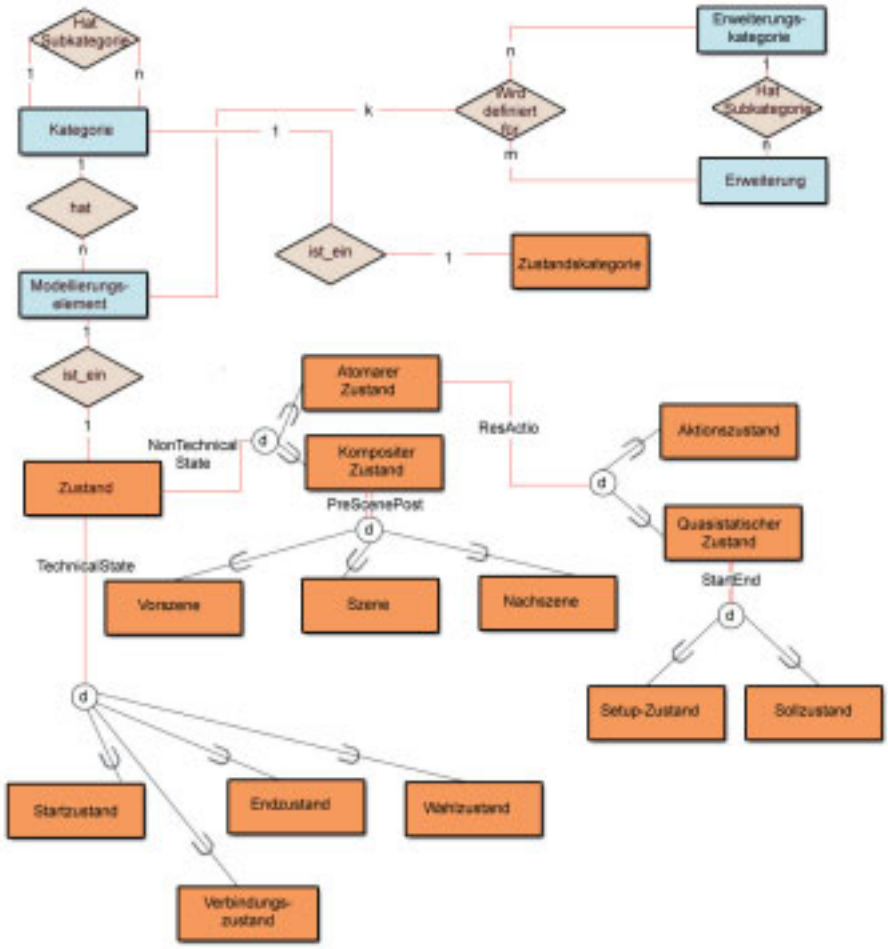


Bild 5.27: Zustandsmodellierung: Arten von Zuständen

und Quasistatische Zustände. Die Quasistatischen Zustände unterteilen sich in der aktuellen Version in Setup- und Sollzustände. Genauere Semantik dieser Zustände wurde in [Ent09b] sowie in Abschnitt „Anforderungen aus informatischer Sicht“ 5.1.2 erklärt. Jeder Zustand muss auch einer Kategorie zugeordnet sein. Die Kategorien können selbstverständlich in Unterkategorien gegliedert sein. Die Modellierung von generischen Variablen ist in der Abbildung 5.28 zu sehen. Der Grund, aus welchem es jeweils zwei Entitätstypen für Zustand, atomares generisches sowie für komposites generisches Attribut gibt, ist folgender. Die hellgrünen Entitätstypen bezeichnen die M1-Instanzen des Domänen-Metamodells. Später werden sie unter anderem auf Tabellen abgebildet, in welchen die konkreten M0-Instanzen verwaltet werden. Nimmt man als Beispiel das atomare generische Attribut „Geschwindigkeit“, so wird auf M2-Ebene (in der Abbildung 5.28

orangenfarben) lediglich sein Name verwaltet, während auf M1-Ebene (hellgrün) die M0-Werte dieses Attributs abgelegt werden. Daher besitzt beispielsweise der M1-Entitätstyp "Atomares Generisches Attribut" (in der Abbildung 5.28 hellgrün) ein Attribut namens „Value". Die M0-Werte werden bei der Abbildung der M1-Entitätstypen auf Tabellen als Einträge in diesen verwaltet. Das hier erklärte Prinzip der Aufteilung in M2-, M1-, beziehungsweise M0-Modellierungsebenen gilt für sämtliche im weiteren Verlauf dieser Arbeit modellierten Bestandteile der Testspezifikation. Was die projektspezifischen Variablen betrifft (Abbildung 5.29), so werden diese in den

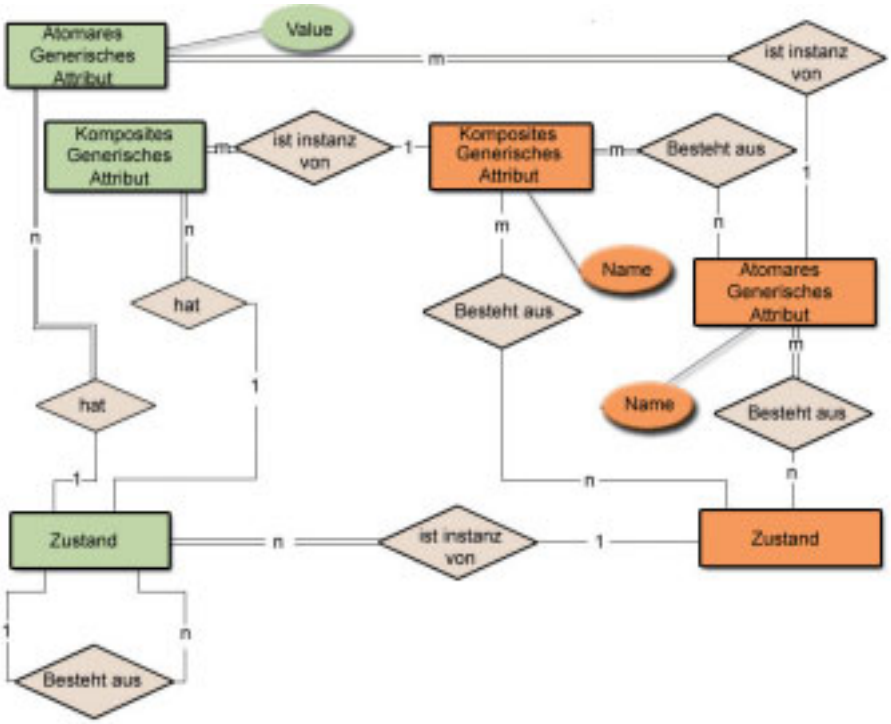


Bild 5.28: Zustandsmodellierung: Generische Variablen

so genannten Erweiterungen „Extensions"(siehe Kapitel 4) zusammengefasst. Somit kann man (sub)projektspezifische Anpassungen für die einzelnen Zustände vornehmen. Die Extensions selbst besitzen eine ähnliche Struktur wie die Zustände: Sie werden ebenfalls sowohl auf M2 als auch auf M1 modelliert. Auf der M2-Ebene kann jede Erweiterung mehrere projektspezifische atomare und komposite Variablen besitzen. Dementsprechend kann auch jede projektspezifische Variable, atomar und komposit, zu mehreren Extensions gehören. Auf der M1-Ebene können zu jeder M2-Extension mehrere M1-Instanzen gehören. Jede M1-Instanz gehört dabei eindeutig zu genau einer M2-Extension. Jede M1-Extension ist eindeutig genau einem M1-Zustand zugeordnet. Jeder M1-Zustand kann wiederum mehrere M1-Extensions besitzen. Beispielhaft könnte ein kom-

positer Zustand für den Bereich realer Testfahrten bei der untersuchten Abteilung durch solche Erweiterungen wie Straßenzustand und Wetterzustand ergänzt werden. Von zentraler Bedeutung

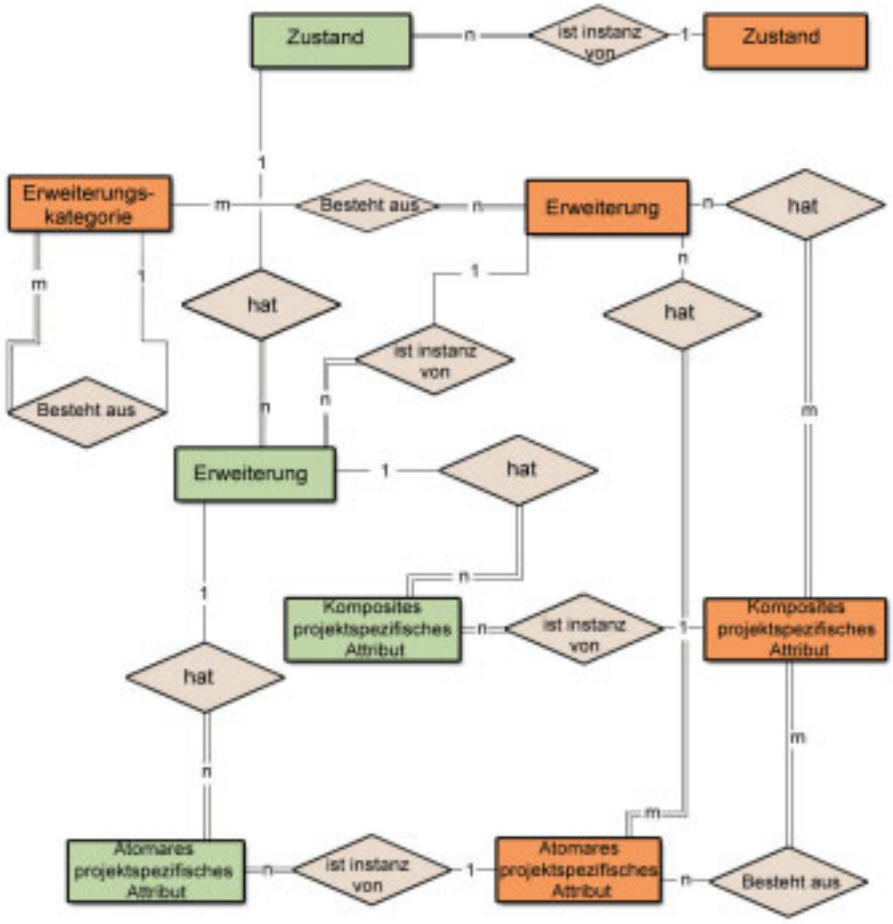


Bild 5.29: Zustandsmodellierung: Erweiterungen

ist der Entitätstyp „Erweiterungskategorie“. Diese hat die in der Abbildung (Abbildung 1.6 Kapitel 1) dargestellte Struktur. Somit ist jede Extension einer bestimmten Testtechnik im Rahmen eines bestimmten Subprojekts zugeordnet. Um den Zusammenhang zwischen Subprojekt, Zustand und Erweiterung auf M2-Ebene herzustellen, ist die Einführung Relationship-Typs „ist_definiert_für“ notwendig (Abbildung 5.30). Dieser Beziehungstyp erlaubt es beispielsweise, einem Entwickler spezifisch für sein Subprojekt und sogar für die Testtechnik für einen bestimmten Zustand die zugehörigen Erweiterungen anzuzeigen.

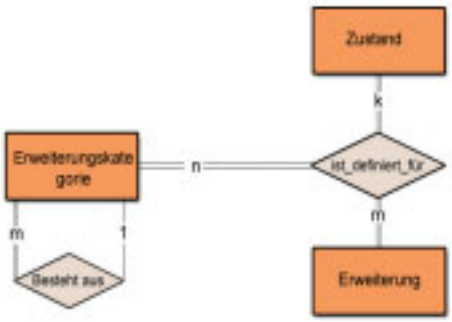


Bild 5.30: Erinnerung: Zuordnung von Zustandserweiterungen den Subprojekten

Allgemeiner Aufbau der Testspezifikation

Vom Typ „Modellierungselement“ müssen für den allgemeinen Aufbau der Testspezifikation folgende Elemente abgeleitet werden: „TestStep“(dt. Testschritt), „TestData“(dt. Testdaten) sowie „TestCase“(dt.Testfall). Der Zusammenhang zwischen dem Testfall und den Testdaten ist in der Abbildung 5.31 zu sehen. Der Testfall selbst wird auf M2, M1 und M0 modelliert. Der Grund ist folgender: Die Testfälle können für unterschiedliche Subprojekte durch unterschiedliche Attribute beschrieben werden. Auf M1-Ebene werden deshalb konkrete subprojektspezifische Attribute definiert während auf M0-Ebene diese Attribute mit konkreten für diesen Testfall relevanten Werten belegt werden. Ähnlicher Sachverhalt gilt auch für Testdaten, also Fahrszenen, in diesem Fall: Unterscheiden sich zwei Fahrszenen nur durch Attributbelegungen, so sind sie als Instanzen (M0) eines und desselben Fahrszenentyps zu betrachten. Zwei Fahrszenentypen (M1) unterscheiden sich dann, wenn sie beispielsweise einen unterschiedlichen Ablauf (Anordnung und Typ von den Zuständen) aufweisen. Die Notwendigkeit der Trennung in M2, M1 und M0 ist ebenfalls dadurch begründet, dass der Entwickler oftmals nach bereits bestehenden Testfällen/Testdaten suchen wird. Anschließend wird er lediglich bestimmte Parameterbelegungen ändern und diese als Instanzen eines bestehenden Testfalls abspeichern. Er wird sich sicherlich auch zu einem bestehenden Testdatentyp (M1) alle bereits angelegten Instanzen anzeigen lassen wollen. Jeder Testdaten-Entitätstyp besteht aus einer Reihenfolge von Schritten (Entitätstyp: Testschritt) (Abb. 5.31). Der Entitätstyp „Testdaten“ ist über einen 1:1 Beziehungstyp mit dem Entitätstyp „Testschritt“ verbunden. Dabei handelt es sich um den ersten Testschritt, aus welchem ein Testdaten-Satz besteht. Die nachfolgenden Testschritte sind mit ihren jeweiligen Vorgänger verbunden (Beziehungstyp: „ist_Nachfolger_von“). Jeder Schritt wird durch einen kompositen Zustand beschrieben. Alle in der Abbildung 5.31 dargestellten Entitätstypen („Testfall“, „Testdaten“, „Zustand“) werden

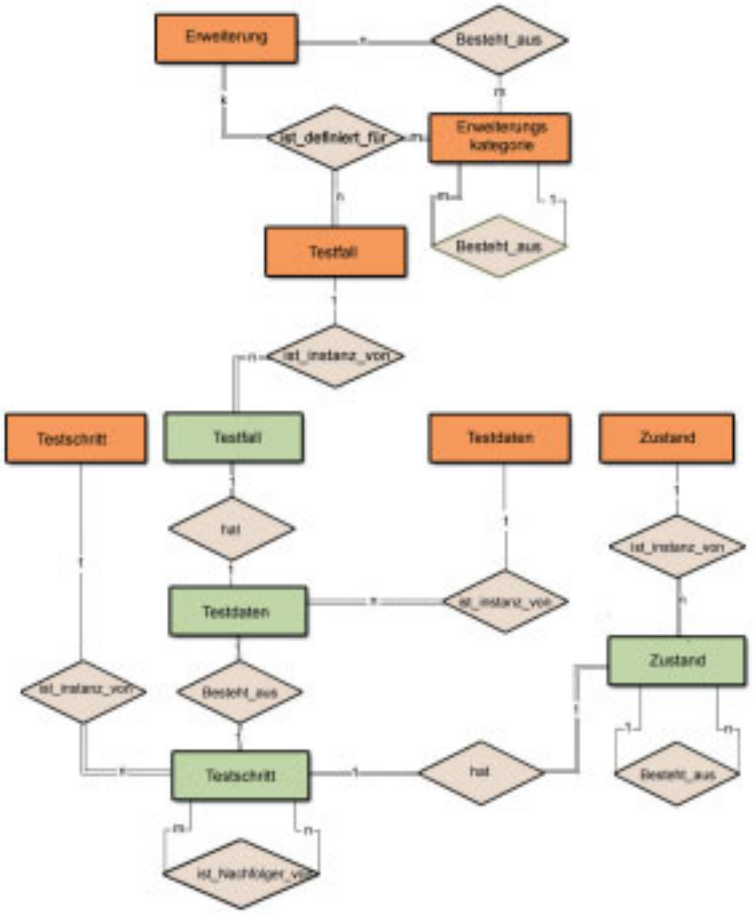


Bild 5.31: Struktur der Testspezifikation

sowohl durch atomare als auch komposite generischen Variablen beschrieben. Zusätzlich können Testfälle und Testdaten durch (sub)projektspezifische Variablen beschrieben werden, welche genauso wie bei Zustandsmodellierung zu Erweiterungen (Engl. Extensions) zusammengefasst sind. Aus Übersichtlichkeitsgründen wurden sämtliche Attributsarten in der Abbildung 5.31 nicht eingezeichnet. Auch die Erweiterung wurde nur für den Entitätstyp „Testfall“ eingezeichnet. Bei den restlichen Entitätstypen („Testschritt“, „Testdaten“, „Zustand“) wird diese auf dieselbe Art und Weise definiert.

Nachdem die Modellierung von Testspezifikationsdaten dargestellt wurde, wird nun auf die Modellierung von den SUT-Data (engl. System Under Test Data) eingegangen. Im Bereich der Fahrerassistenzsysteme wird der Begriff des zu testenden Systems durch Funktionen, aber auch

durch Einzelmodule dieser Funktionen charakterisiert. Die Modellierung selbst geschieht auf genau dieselbe Art und Weise wie Zustände, Ereignisse, Aktionen und Bedingungen. Im ersten Schritt wird das Element „System Under Test“ vom „Modellierungselement“ (Bild 5.25) im Basismodell abgeleitet. Die Parametrisierung für das zu testende System wird in generische und subprojektspezifische unterteilt (5.32). Ein Parametersatz für den Entitätstyp “System Under Test“ besteht aus atomaren sowie kompositen Variablen. subprojektspezifische Variablen werden in Erweiterungen zusammengefasst (in der Abb. 5.32 aus Platzgründen ausgelassen). Je nach dem Subprojekt können die SUT-Parametersätze unterschiedlich aussehen (dreifacher

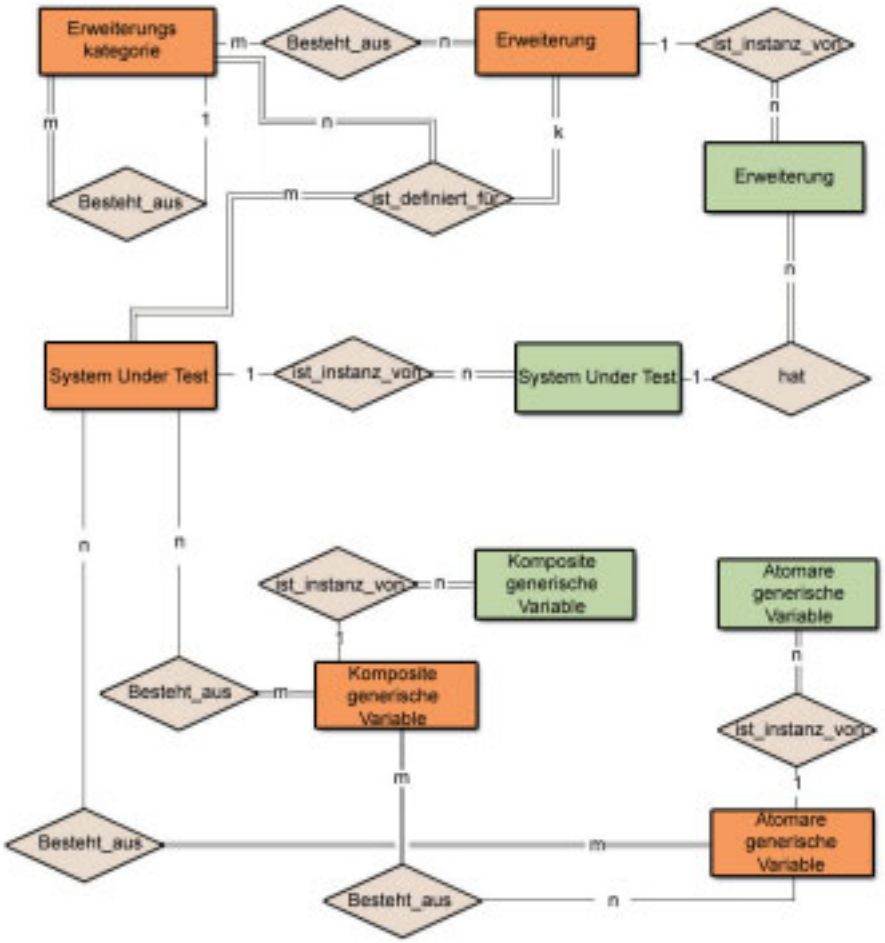


Bild 5.32: Das zu testende System

Beziehungstyp „ist_definiert_für“ zwischen „System Under Test“, „Erweiterungskategorie“ und „Erweiterung“). Aufgrund der Tatsache, dass „System Under Test“ vom „Modellierungselement“

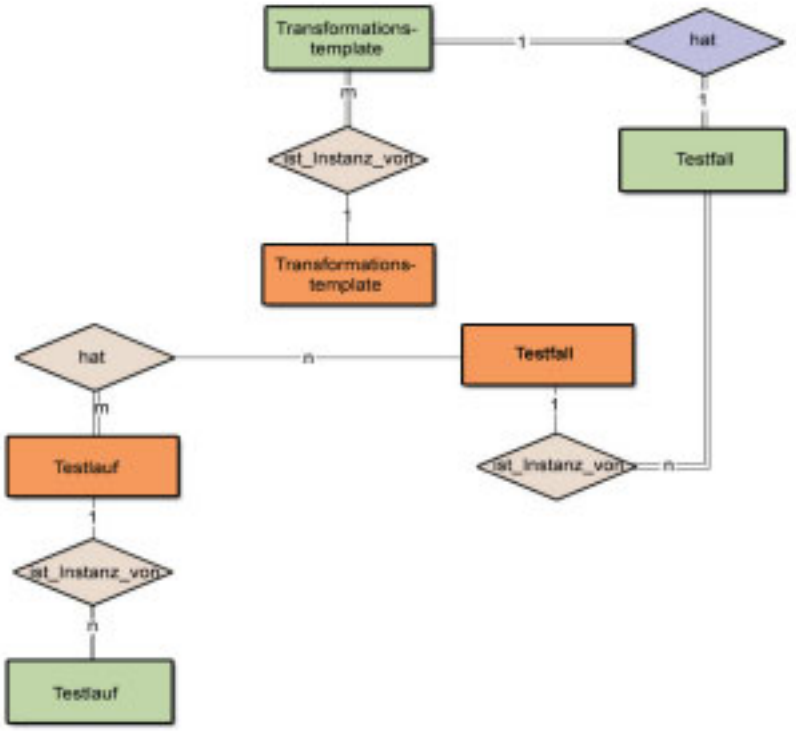


Bild 5.33: Testlauf-Definition

im Basis-Metamodell abgeleitet wird, erbt es die Beziehungen zu den generischen Attributen sowie zu den projektspezifischen Erweiterungen. „System Under Test“, Erweiterungen und Variablen können auf der M1-Ebene mehrere Instanzen besitzen.

Auf die Modellierung von Soll-Werten (engl.TargetData) sowie von den Setup-Daten (engl. „SetupData“) wir hier aus Platzgründen verzichtet. Die Darstellung ist aber absolut identisch zu der von „System Under Test“. Zum Schluss bleiben lediglich zwei wichtige Aspekte zu modellieren: Zuordnung der Testfälle zu den Testläufen sowie die Benutzerverwaltung (Abb. 5.33 und 5.34). Grundsätzlich wurde auch bei der Modellierung der Testläufe eine Trennung in M2 und M1-Modellierungsebenen beibehalten. Der Grund dafür ist, dass je nach der Abteilung oder eventuell auch je nach Subprojekt die Attribute zur Beschreibung einer Testspezifikation sich unterscheiden können. Diese Möglichkeit ist zurzeit zwar als theoretisch zu betrachten, nichtsdestoweniger soll das vorliegende Schema die maximale Flexibilität bieten und außerdem soll die Trennung in verschiedene Modellierungsebenen konsequent angewandt werden. Demnach kann ein auf M2-Ebene modellierter Testlauf (Abb. 5.33) durch eine Menge von generischen Attributen beschrieben werden. Ebenfalls können projektspezifische Erweiterungen definiert werden. Das „Testlauf“-Element

wurde ebenfalls von „Modellierungselement“ im Basismodell abgeleitet. Aus diesem Grund werden sämtliche Beziehungen geerbt. Damit in der graphischen Oberfläche für jede Testspezifikation

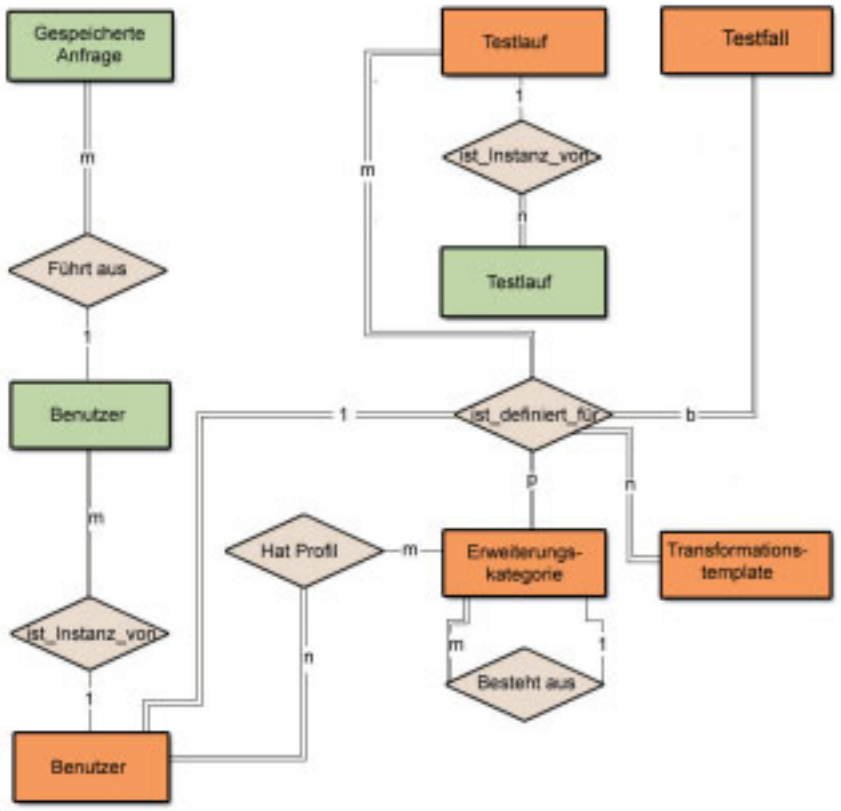


Bild 5.34: Benutzerverwaltung

projektspezifische Attribute angezeigt werden, muss ein dreifacher Relationstyp zwischen den folgenden Entitätstypen definiert werden: „Testlauf“, „Erweiterung“, „Erweiterungskategorie“ (In der Abbildung 5.33 ausgelassen). Auf der M1-Ebene kann ein Testlauf mehrere Instanzen besitzen. Wichtig ist in diesem Zusammenhang der fünffache Relationstyp „ist_definiert_für“ (Bild 5.34), denn aus diesem wird ersichtlich, welche Testfälle(„Testfall“) zu welchen Testläufen(„Testlauf“) gehören und welchen (Sub)Projekten („Erweiterungskategorie“) sind die Testspezifikationen zugeordnet. Ebenfalls wird einer bestimmten Testspezifikation (die wiederum für ein bestimmtes Projekt definiert wurde) ein Transformationstemplate („Transformationtemplate“) zugeordnet. Selbstverständlich können alle (außer „Erweiterungskategorie“) obengenannten M2-Elemente auch Instanzen auf M1-Ebene besitzen. Was die Benutzerverwaltung betrifft, so ist sie auf identische Art und Weise wie ein Testlauf modelliert worden (Abb. 5.34): Auf M2-Ebene kann ein

Benutzer („Benutzer“) durch mehrere generische und projektspezifische Variablen beschrieben werden, die zu den Erweiterungen (engl. Extensions) zusammengefasst sind. Damit in der graphischen Oberfläche alle einem Benutzer zugehörigen projektspezifischen Attributen angezeigt werden können, ist die Definition eines dreifachen Relationstyps „ist_definiert_für“ zwischen folgenden Entitätstypen notwendig: “Benutzer“, „Erweiterung“, „Erweiterungskategorie“ (In der Abbildung 5.34 ausgelassen). Ein Benutzer kann für mehrere (Sub)Projekte arbeiten und mehrere Benutzer können an einem (Sub)Projekt arbeiten (zweifacher Relationstyp „Hat Profil“). Auf der M1-Ebene haben die Benutzer mehrere Instanzen. Jeder M1-Instanz können dabei mehrere abgespeicherte Suchanfragen („Gespeicherte Suchanfrage“) zugeordnet werden, welche die Suche nach einzelnen Fahrszenen oder deren Sequenzen ermöglichen.

5.2.2 Wahl der Modellierungssprachen

Wie im Konzeptkapitel erwähnt, gibt es nicht eine einzige Modellierungssprache, die die Konzepte der Operationalisierbarkeit, der zentralisierten Verwaltung sowie der graphischen Darstellung gleich stark berücksichtigt. Aus diesem Grund müssen für die Domäne der Fahrerassistenzsysteme so genannte Modellierungsräume bestimmt werden (siehe Kapitel 4). Zunächst wird der konzeptuelle Modellierungsraum zur graphischen Darstellung des Domänen-Metamodells gewählt. Im Fall der vorliegenden Anwendungsdomäne wurden die erweiterten Entity-Relationship-Diagramme (EER) [EN02] gewählt. Die Wahl lässt sich dadurch begründen, dass dem Domäneningenieur sowohl UML als auch EER bekannt sind. EER wurde aus dem Grund gewählt, dass daraus automatisch die Modellierungshierarchie in der Sprache des technischen Modellierungsraums zur Berücksichtigung des Verwaltungsaspekts „SQL DDL“ erzeugt werden kann. Die automatische Generierung ist im Rahmen dieser Arbeit durch das Vorhandensein des kommerziellen EER-Modellierungstools Toad [Toa] bedingt. Würde die konzeptuelle Modellierung in UML erfolgen, müsste eine manuelle Übertragung der Modellierungshierarchie in EER erfolgen um daraus automatisch mit Hilfe von Toad SQL-DDL zu erzeugen (siehe Abbildung 5.35). Da die Modellierung gleich im Modellierungsraum zur Erzeugung verwaltungsfähiger Modellierungshierarchien erfolgte, wurden die verwaltungstechnischen Aspekte wie zum Beispiel Benutzerverwaltung, Organisation der Testfälle zu den Testläufen oder aber die Verwaltung von Transformationstemplates bereits berücksichtigt (siehe oben). Der Grund, aus welchem SQL DDL als technischer Modellierungsraum gewählt wurde, ist, dass zum einen in der analysierten Abteilung bereits eine SQL-Datenbank zur Verwaltung von großen Mengen an Messdaten realisiert wurde [Ent06], sodass sich Implementierung der Modellierungshierarchie in SQL DDL insofern lohnen würde, als man auf das Schema zur Verwaltung real eingefahrener Messungen vom so erzeugten Schema

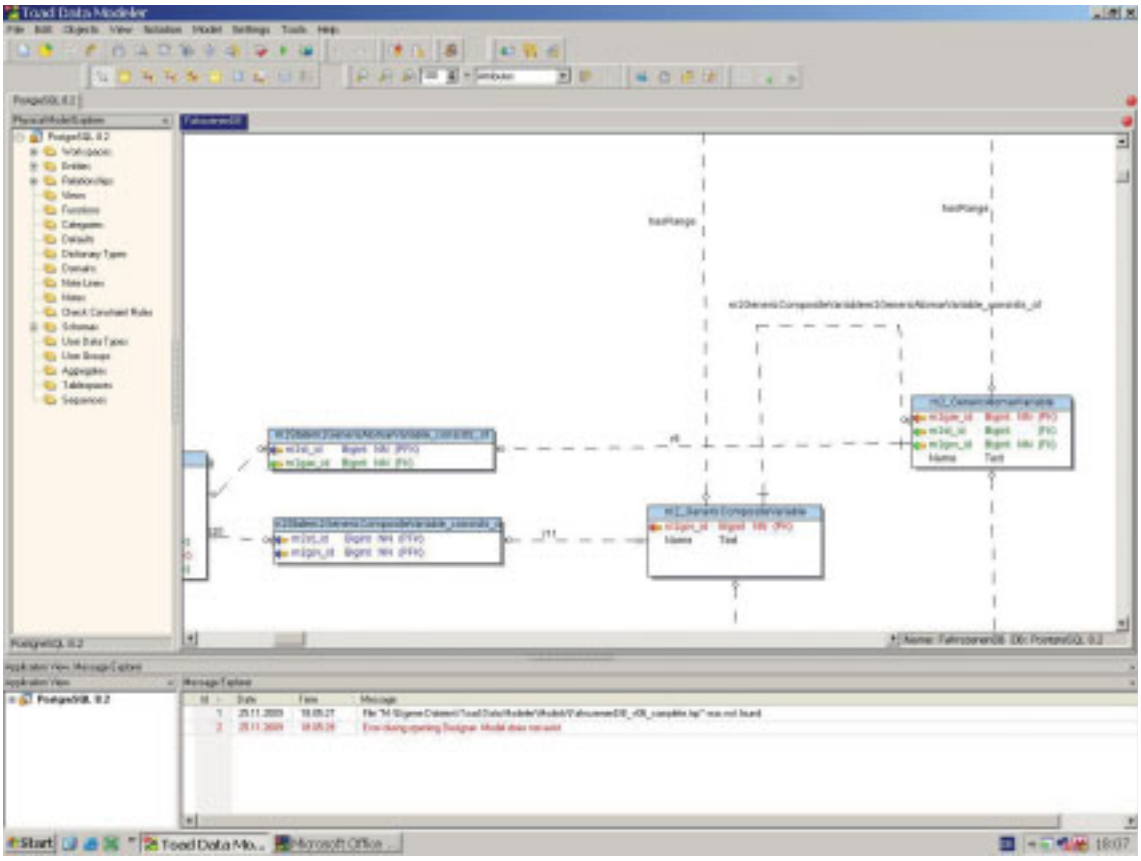


Bild 5.35: Toad: Screenshot

zur Verwaltung formaler Testspezifikationen für FAS problemlos verweisen kann. Zum anderen ist SQL eine stabile und ausgereifte Sprache zur Definition von Datenbanksystemen, mit der sehr viel Erfahrung gesammelt wurde und entsprechend viel Dokumentation vorhanden ist. Was die Wahl des konzeptuellen Modellierungsraums zur Erzeugung transformationsfähiger Testspezifikationen betrifft, so fiel hier die Wahl relativ schnell auf Ecore. Ecore ist das Metametamodell (M3) von Eclipse Modelling Framework [Ecl]. Ursprünglich wurde MOF von OMG in Betracht gezogen als Metametamodell für EMF, dann wurde darauf verzichtet, da MOF zu komplex ist. Aus diesem Grund wurde Ecore entwickelt, welches im Vergleich zu Meta Object Facility einen wesentlich geringeren Grad an Komplexität aufweist. Im Verlauf der Zeit wurde OMG klar, dass es einer Vereinfachung des MOF bedarf und es wurde deshalb parallel zu Ecore das EMOF-Standard [Obj10a] entwickelt als eine vereinfachte Version von MOF. „Dadurch, dass es einen ständigen Ideenaustausch zwischen den Entwicklern von Ecore und EMOF gab, ist EMOF stark auf Ecore ausgerichtet, sodass EMF die in EMOF beschriebenen Metamodelle lesen, schreiben und manipu-

lieren kann genauso wie die in Ecore geschriebenen“ [Kar07]. Die Struktur von Ecore ist in der Abbildung 5.36 gezeigt. Ecore wird durch Ecore selbst beschrieben (M3). Neben Ecore wurden

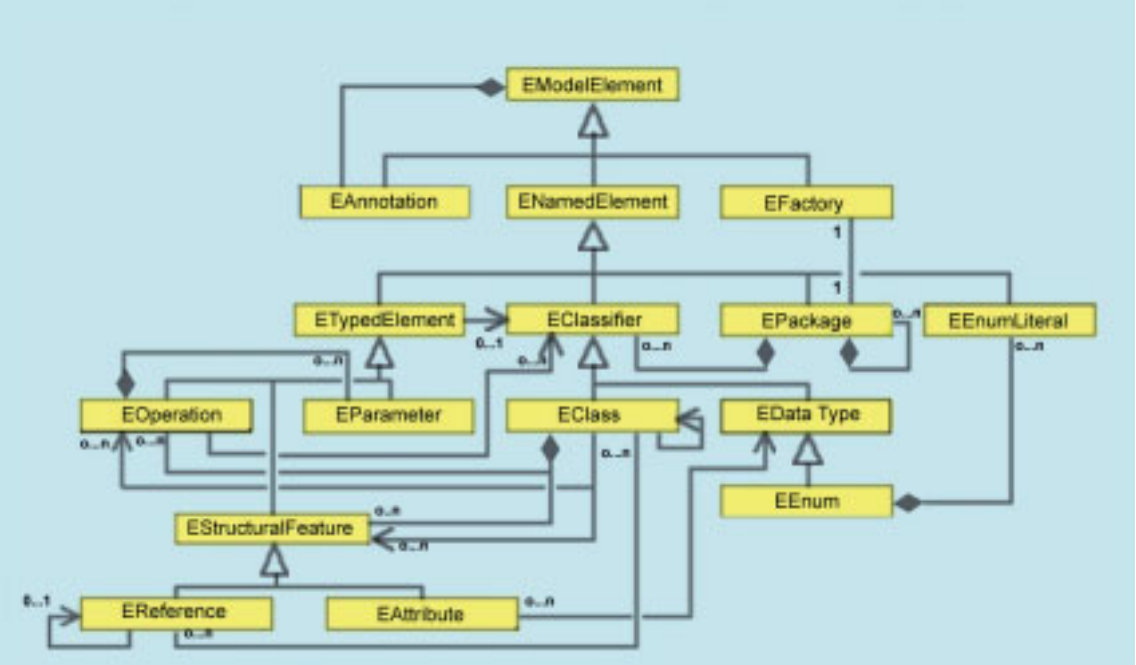


Bild 5.36: Das Metametamodell von Ecore

solche Ansätze aus dem Bereich modellgetriebener Software-Entwicklung untersucht wie Java Emitter Templates [Ecl04] und Atlas Model Management Architecture [Sch06]. Die Wahl von Ecore ist primär dadurch begründet, dass es die Definition domänenspezifischer Metamodelle erlaubt. Somit kann ein bereits bestehendes Metamodell von einem Statechart um zusätzliche in dieser Arbeit vorgestellte Modellierungselemente problemlos erweitert werden. Eine solche Erweiterung beispielsweise bei UML vorzunehmen, würde zur Verletzung des Standards führen. Es können die UML-Metamodelle sehr wohl als Ausgangsbasis (!) genutzt werden. Ein gewichtiges Argument ist natürlich die Tatsache, dass seit mehreren Jahren auf dem Markt freiverfügbare Frameworks existieren, die Ecore implementieren und die Modell-zu-Text Transformation ermöglichen. Zum anderen wurde auch innerhalb des Konzerns genügend Erfahrung mit einigen von solchen Ecore-basierten Frameworks gesammelt wie zum Beispiel Open Architecture Ware [Ecl10b]. Es ist also genügend Know-How sowohl in der Community als auch im Unternehmen vorhanden. Dementsprechend fiel auch die Wahl des technischen Modellierungsraums relativ schnell auf EMF-Modelle sowie auf XPand als Transformationssprache. Als Transformationsfra-

mework wurde ein auf Eclipse Modelling Framework aufbauendes Transformationsframework namens „Open Architecture Ware“ gewählt, denn nach [Mer07]:

- „Lassen sich damit beliebige Metamodelle verwenden. Somit kann für das Projekt die Art der Modellierung gewählt werden, die zum Anwendungsfall passt und man ist nicht an Restriktionen einer bestimmten Modellierungsprache, wie etwa in UML, gebunden. Voraussetzung ist, dass das Metamodell in Java implementiert werden kann und ein Parser für OAW zur Verfügung steht.

- Kann auf der Ausgabeseite jeglicher textbasierter Output erzeugt werden. Dazu zählen Quelltext in unterschiedlichsten Programmiersprachen, Dokumentation und Konfigurationsdateien. Dabei ist zu beachten, dass nur generiert werden kann, was vorher modelliert worden ist oder sich aus dem Modell ableiten lässt.

- Können Aufgaben, die sich nicht mit eingebauten Mitteln lösen lassen, durch die Integration von Javaprogrammen in den Generierungsworkflow nachgerüstet werden.

- Haben die Sprachkonstrukte und Komponenten für die Outputgenerierung eine gut erlernbare Syntax und Semantik. Man bedenke, dass XPand als Transformationssprache lediglich ca. 15 Schlüsselwörter besitzt. Außerdem ist die Unterstützung durch Eclipse-Editoren für die Template- und Extensionerstellung durch Syntaxhervorhebung und das Vorblenden von verfügbaren Befehlen hilfreich.“

Der generelle Aufbau von OAW ist in der Abbildung 5.37 dargestellt [Tho05]. Zunächst wird ein

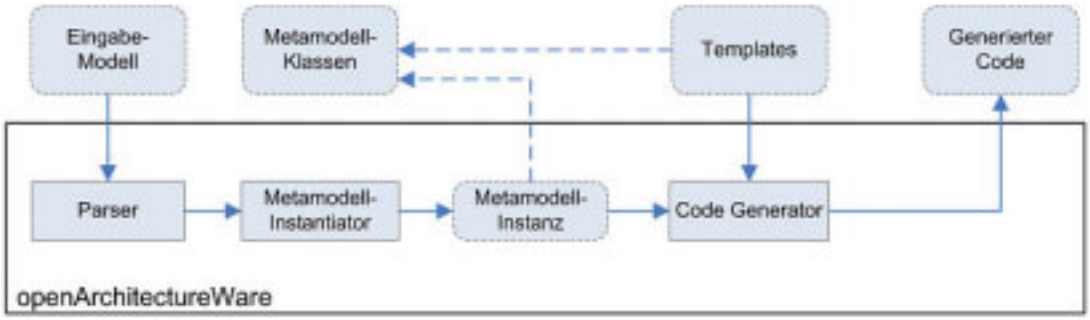


Bild 5.37: Funktionsweise von Open Architecture Ware [Tho05]

Modell instanziiert. Ein XPand-Transformationstemplate ist für die Erzeugung der Zielformate zuständig. Er hat Zugriff auf die Metamodell-Klassen und kann damit Transformationsregeln

definieren. Zur Laufzeit werden die Transformationsregeln auf die Metamodell-Instanz angewandt. In OAW werden die Metamodell-Instanzen im speziellen .emf Format persistiert. Für eine detaillierte Funktionsbeschreibung sei auf [Ecl10b] verwiesen.

Nachdem nun sämtliche Modellierungsräume für die FAS-Domäne gewählt worden sind, ergibt sich folgende Systemarchitektur für den zu realisierenden Prototypen (Abb. 5.38).

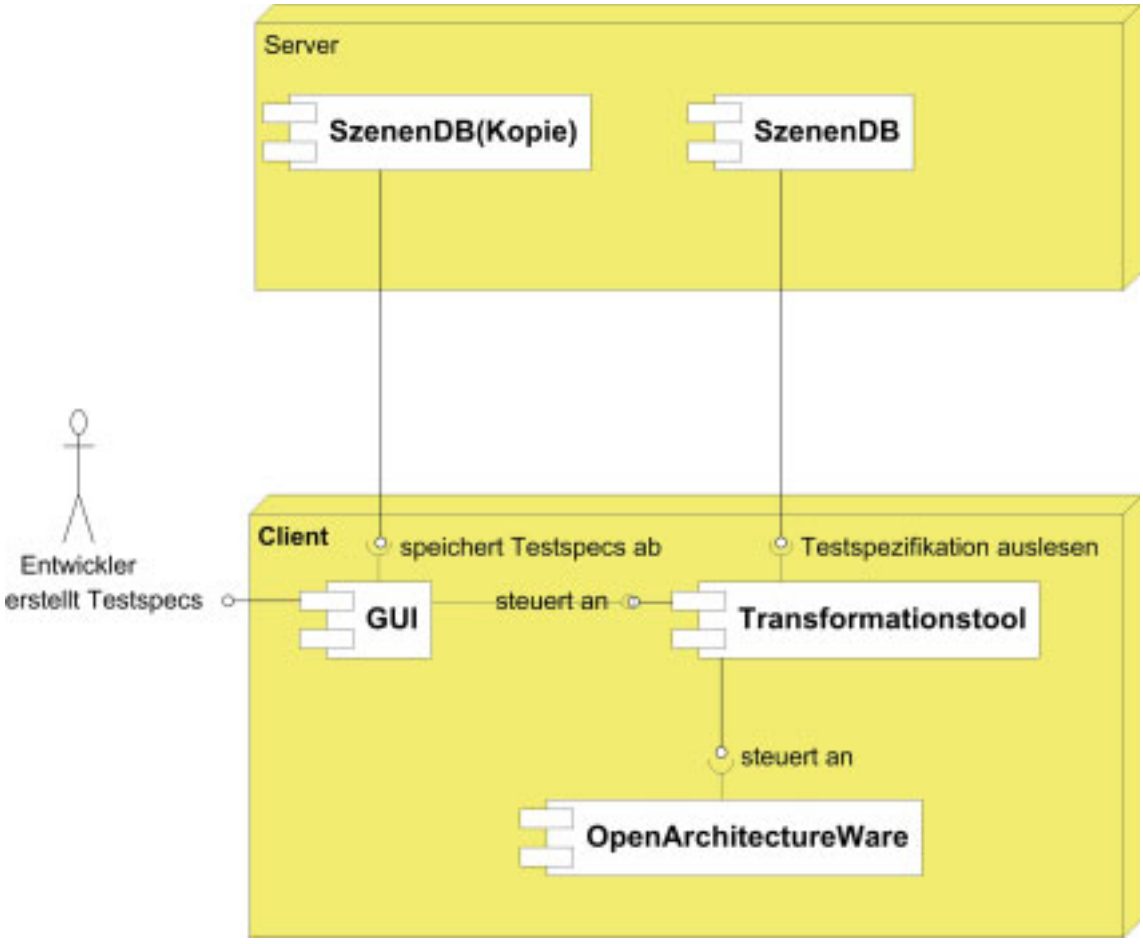


Bild 5.38: Systemarchitektur

Zugrunde gelegt wurde die klassische Client-Server Architektur, bei welcher die gesamte Modellierungshierarchie in einer PostgreSQL-Datenbank (Version 8.3) [SRH90] zentral auf dem Abteilungsserver verwaltet wird („SzenenDB“). Die Datenbank besteht aktuell aus 114 Tabellen. Client-seitig wird eine graphische Oberfläche in C-Sharp (Windows Forms) installiert, welche unter Verwendung einer zusätzlichen Bibliothek zur Programmierung von Flussdiagrammen [Las10] geschrieben wurde.

Ebenfalls Client-seitig wird ein entsprechendes Transformationstool [Cha10] installiert. Dieses Tool ist dafür zuständig, im ersten Schritt die Transformation aus dem SQL-DDL Modellierungsraum in Ecore-EMF zu schaffen (M0-Ebene). Anschließend wird das vom Entwickler gewählte XPand-Transformationstemplate gewählt und OAW mit diesem Template ausgeführt.

Ein Screenshot der graphischen Oberfläche ist in der Abbildung 5.39 zu sehen. Im Gegenteil zum Prototypen wurde nun die Anbindung der Oberfläche an die Datenbank (Abb. 5.38) geschaffen. Ebenfalls wurde das Transformationstool (Abb. 5.38) integriert. Im linken Teil befindet sich das besagte Wörterbuch. Aus diesem werden einzelne Systemzustände ausgewählt und auf der Modellierungsfläche per „Drag And Drop“ platziert. Je nach dem Anwendungsfall wird jeder Zustand durch eine Menge nur generischer oder generischer und projektspezifischer Attribute beschrieben. Diese können im unteren Teil der graphischen Oberfläche ausgefüllt werden. Zustände können per einfachem Mausclick miteinander verbunden werden. Dabei wird automatisch ein Menü zur Angabe von Ereignissen, Bedingungen und Aktionen angezeigt. Vor der Transformation muss eine solche Beschreibung in der Datenbank abgespeichert werden. Die Transformation wird per einfachem Mausclick angesteuert, dabei muss im ersten Schritt ein entsprechendes XPand-Template ausgewählt werden, dann muss noch die Ausführungsdatei für das Transformationstool gewählt werden. Nach der Transformation wird das erzeugte Zielformat lokal auf dem Client-Rechner abgelegt. Ein Beispiel für die Template-Definition in XPand ist im folgenden Codeabschnitt zu sehen.

```
 «DEFINE ext_action_state FOR Extension»
«FOREACH PrjSpecAtVars AS psav»
«addRowToDocument("ActionTable", psav.Name, psav.Value)»
«ENDFOREACH»
«FOREACH PrjSpecCompVars AS pscv»
«addRowToDocument("ActionTable", pscv.Name, " ")»
«FOREACH pscv.PrjSpecAtVars AS pscvav»
«addRowToDocument("ActionTable", " " + pscvav.Name,
                                    pscvav.Value)»
«ENDFOREACH»
«ENDFOREACH»
```

In dem oben dargestellten Code-Ausschnitt wird ein Template namens „ext_action_state“ für das Metamodell-Element namens „Extension“ (Erweiterung) definiert. Innerhalb dieses Templates werden in einer Schleife („FOREACH“) alle projektspezifischen Variablen einer PDF-Tabelle hinzugefügt („ActionTable“). In der zweiten Schleife werden auf ähnliche Art und Weise die

kompositen Variablen in die „ActionTable“ eingetragen. Das Element „addRowToDocument(...)“ ist ein in Java definierter und über den XPand-Erweiterungsmechanismus eingebundener Operator, der unmittelbar für die Erzeugung des Tabelleneintrags zuständig ist.

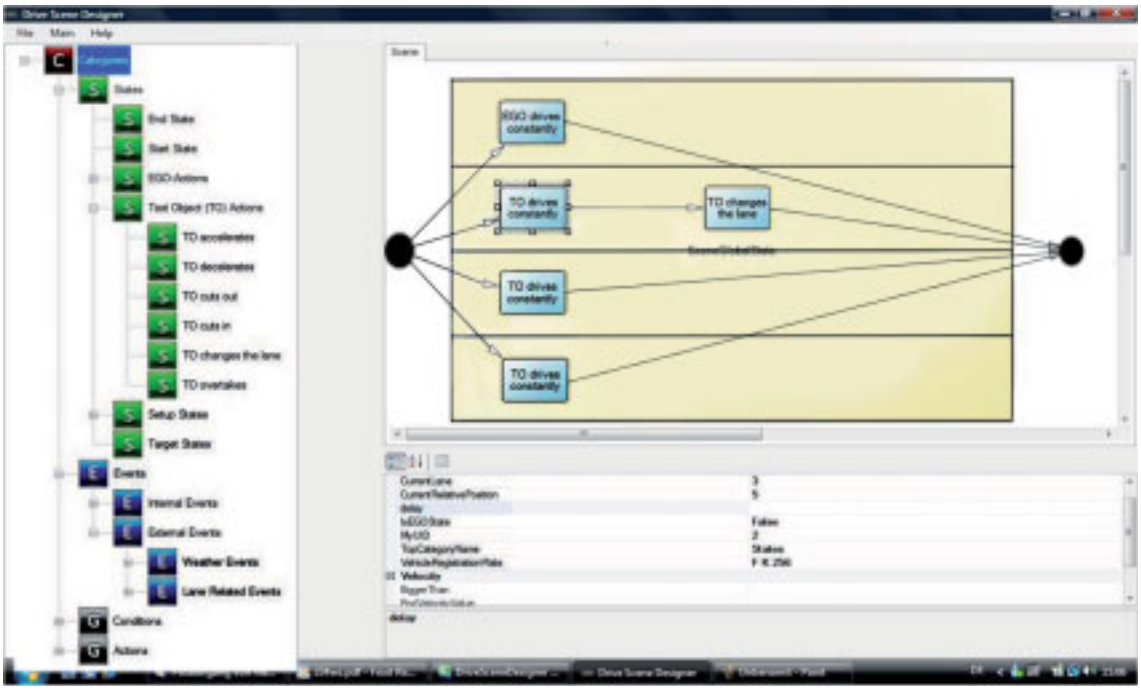


Bild 5.39: Ein Screenshot der graphischen Benutzeroberfläche

5.3 Partielle Instanziierung des Metamodells

Nach der Implementierung erfolgte gemäß dem definierten Vorgehensmodell die partielle Instanziierung des Metamodells. Zu diesem Zweck wurden in einem ersten Schritt zum einen Gespräche mit den Entwicklern durchgeführt, um Fahrzeugzustände und die entsprechenden Beschreibungsattribute für den Bereich „reale Messfahrten“zu ermitteln. Zum anderen wurden XML-Formate analysiert zur Beschreibung eines automatisierten SiL-Testlaufs. Die Ergebnisse wurden in Form von Kategorien von Zuständen, Ereignisse, Bedingungen und Aktionen in einem separaten Dokument erfasst [Ent09a]. In der Abbildung 5.40 ist beispielhaft ein Ausschnitt aus der ermittelten Zustände-Hierarchie aufgezeigt. Da der Prozess der Definition der Testspezifikation sehr von den ermittelten und evaluierten Anwendungsfällen abhängt, wird er im Kapitel „Evaluierung“ erfolgen.

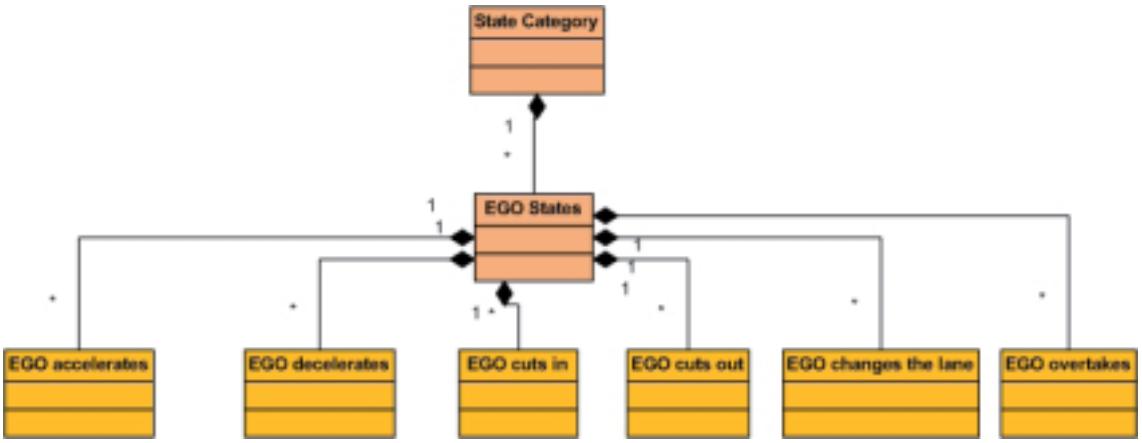


Bild 5.40: Beispielhafte Kategorisierung von Zuständen

Kapitel 6

Evaluierung des Ansatzes

In diesem Kapitel wird die Evaluierung des in dieser Arbeit vorgestellten Ansatzes vorgenommen. Dazu werden in den Abschnitten 6.1.1 und 6.1.2 zunächst die ermittelten Anwendungsfälle für die im Kapitel 5 behandelte Anwendung des Vorgehensmodells innerhalb der Fahrerassistenz-Domäne vorgestellt. Die Anwendungsfälle wurden innerhalb einer Abteilung im Bereich Fahrwerk Vorentwicklung von FAS ermittelt. Anschließend wird gemäß dem im Kapitel 4 spezifizierten Framework der Prozess der Definition einer Testspezifikation mit dem entwickelten Toolset aufgezeigt. Um eine präzisere Idee über die Konzeptanwendbarkeit zu geben, wird abschließend im Abschnitt 6.1.3 ein konkretes Projekt aus der Abteilung vorgestellt, in welchem der Ansatz zur Anwendung kommen kann. Im Abschnitt 6.2 geht es um die Interviews, welche zur weiteren Ansatzevaluierung mit den Entwicklern durchgeführt wurden. Zunächst wird der Aufbau eines solchen Interviews aufgezeigt und anschließend dessen Ergebnisse betrachtet. Abschließend wird im Abschnitt 6.3 die Erfüllung der am Anfang dieser Arbeit gesetzten Ziele analysiert.

6.1 Ermittelte Anwendungsfälle

In diesem Abschnitt werden die ermittelten Anwendungsfälle aufgezeigt, durch welche eine reibungslose Integration des Konzepts in den täglichen Entwicklungsablauf einer Abteilung verdeutlicht wird.

6.1.1 Anwendungsfall: Erstellung einer Testspezifikation vor der Messfahrt

Speziell in den frühen Entwicklungsphasen von FAS haben die Entwickler bestimmte Fahrszenenbeschreibungen im Bewusstsein. In bestimmten Fällen, falls z.B. Vorgängerprojekte durchgeführt wurden, sind diese in den bereits erwähnten Fahrszenarien-Katalogen zusammengefasst. Im Bereich realer Messfahrten besteht folgende Infrastruktur zur Verwaltung der auf einem Prüfgelände oder im freien Verkehr eingefahrenen Messungen (Abbildung 6.1). Aktuell werden die Messun-

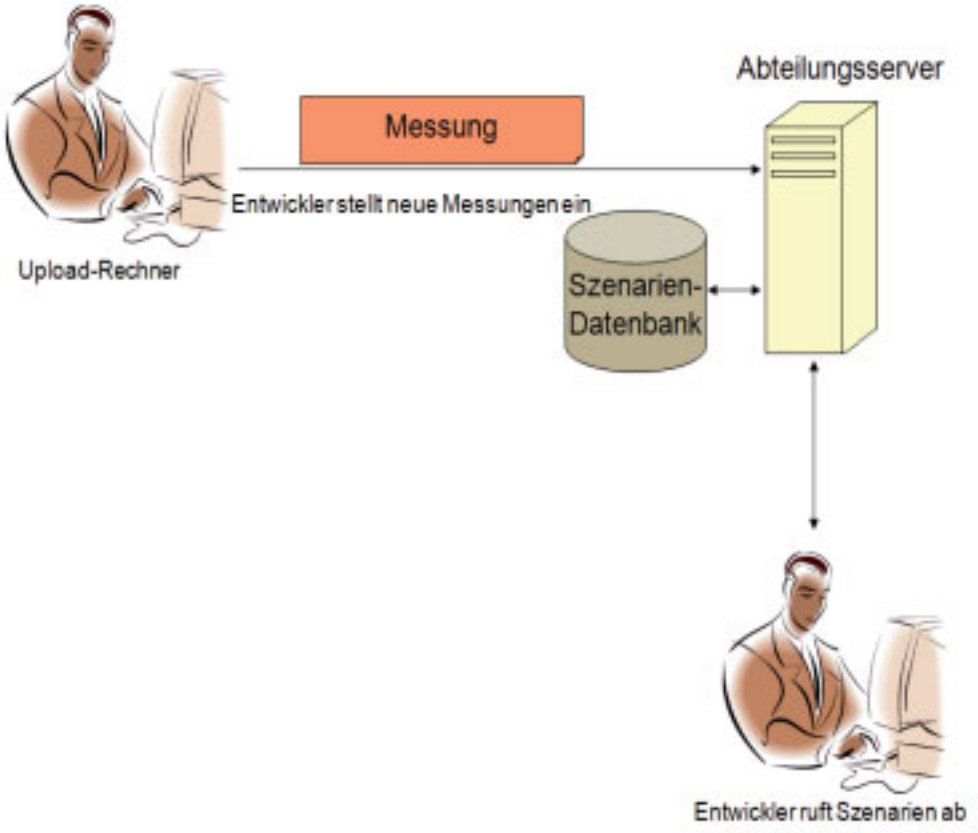


Bild 6.1: Aktuelle Infrastruktur zur Verwaltung von Messungen

gen mit einer minimalen Menge an Attributen nach dem Hochladen auf den Abteilungsserver beschrieben. Aufgrund der Größe (80 GB pro Stunde Fahrt) kann die komplette Messung nicht in der Datenbank verwaltet werden. Es werden deshalb lediglich Metainformationen verwaltet sowie möglicherweise bestimmte extrahierte Größen wie beispielsweise die GPS-Positionen des Fahrzeugs während einer Messfahrt. Auf die eigentlichen Messungen wird lediglich per Dateilink

verwiesen, welches in einer Datenbanktabelle abgelegt wird. Die Messungen selbst werden im binären Format verwaltet.

Der hier beschriebene Anwendungsfall [EGZ09] zur formalen Definition von Testspezifikationen ist in der Abbildung 6.2 dargestellt. Vor einer Messfahrt hat ein Entwickler die Möglichkeit

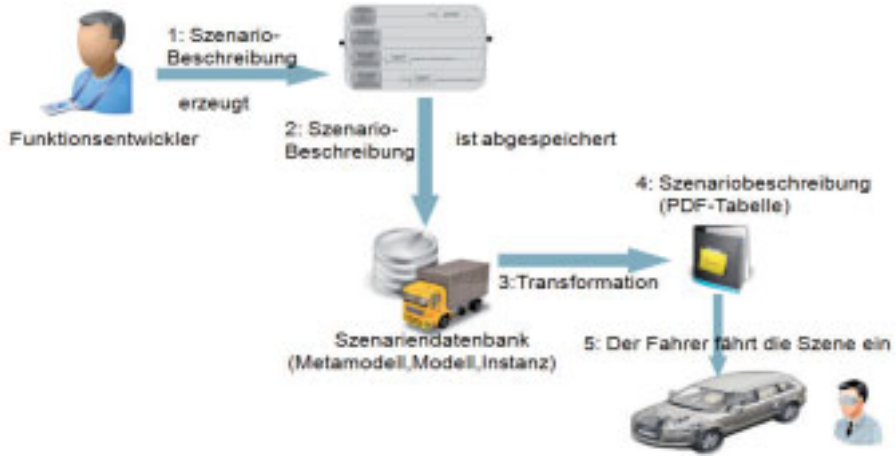


Bild 6.2: Fahrszenenbeschreibung vor der Fahrt

eine generische formale Fahrszenenbeschreibung entweder aus einer bereits bestehenden Menge zu wählen oder sie selbst anzulegen. Dazu wählt er in der graphischen Oberfläche die Option „Create Generic Drive Scene Description“ und ihm werden für jedes Beschreibungselement der Statechart-Methode zunächst generische Attribute angezeigt. Anschließend speichert er die Be-

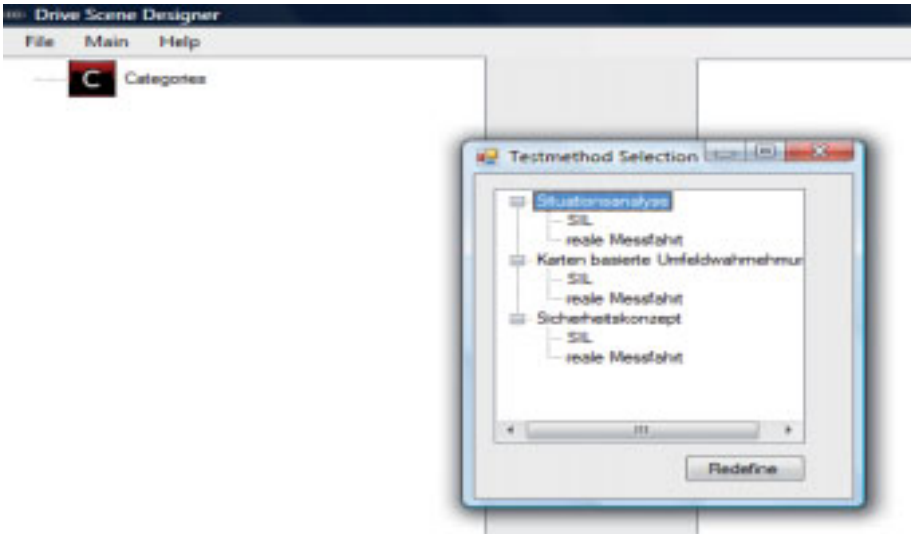


Bild 6.3: Erzeugung generischer Fahrszene

Ausgangssituation	
Zustandbeschreibung	EGO drives constantly
Spurbezeichnung	1
Geschwindigkeit	
Equals	40 km/h
Kfz-Kennzeichen	BB-123
Distanz zum	

Bedingung	Aktion	
	Zustandbeschreibung	EGO accelerates
	Zielgeschwindigkeit	60 km/h
	Quellgeschwindigkeit	40 km/h
	Beschleunigungssatz	2 m/s^2

Bild 6.4: Ausschnitt aus der PDF-Tabelle mit der Beschreibung einer Fahrszene

schreibung im Datenbankschema aus Kapitel 5 ab. Im nächsten Schritt hat er die Möglichkeit, eine solche Beschreibung gemäß dem Rahmenwerk aus Kapitel 5 zu verfeinern. Dazu wählt er das zu verfeinernde Szenario aus einer Liste aus und, nach dem Anklicken der Option „Verfeinern“, muss er nun ein entsprechendes Projekt sowie das Testverfahren innerhalb dieses aus einer in der Abbildung 6.3 beispielhaft aufgezeigten Baumstruktur wählen. Im nächsten Schritt werden dem Entwickler neben den bereits ausgefüllten generischen Attributen die leeren projektspezifischen für jedes Modellierungselement angezeigt. Diese muss er nun ausfüllen, damit am Ende eine projekt- und testartspezifische Fahrszenenbeschreibung entsteht. Am Ende des Beschreibungsvorgangs muss die erzeugte Fahrszene als eine projektspezifische in der Datenbank abgespeichert werden. Nun kann der Entwickler im nächsten und letzten Schritt dieses Anwendungsfalls über ein entsprechendes Menü die Transformationsengine starten. Dazu benötgt er zum einen die Angabe des entsprechenden Transformationstemplates und zum anderen die .bat Datei, mit welcher die Transformationsengine gestartet wird. Am Ende wird letztendlich ein PDF-Dokument erzeugt, welches in aktueller Version eine tabellarische Darstellung der einzufahrenden Fahrszene darstellt, welche für den Testfahrer lesbar ist (Abbildung 6.4). Nach der Messfahrt stellt der Testfahrer die Messungen, wie bereits erwähnt, in die Datenbank ein. Dabei soll die Möglichkeit geschaffen werden, die formale projektspezifische Beschreibung der eingestellten Messung zuzuordnen. Was zur Zeit noch nicht realisiert wurde, aber was der Ansatz erlaubt, ist die Möglichkeit, eine sehr feine Suche auf Metadaten nach bestimmten komplexen Abfolgen von Fahrmanövern durchzuführen. Dabei könnte man die Suchanfrage graphisch in Form einer Statechart-basierten Fahrszenenbeschreibung formulieren. Im nächsten Schritt könnte davon eine SQL-Abfrage abgeleitet werden. Zurzeit geschieht die Anfrageformulierung mit gewöhnlichen Formularen, was nicht sonderlich

flexibel ist und vor allem die Berücksichtigung zeitlicher Aspekte sowie Interaktion mehrerer Verkehrsteilnehmer nicht im vollen Maße erlaubt.

6.1.2 Anwendungsfall: Erstellung einer Testspezifikation nach der Fahrt

Während der realen Messfahrten werden sehr große Mengen an Video-Daten aufgenommen. Diese werden sowohl im Binärformat auf dem Server abgelegt als auch im .avi Format mit geringer Frame-Rate extrahiert. Die extrahierte Version dient primär zu Dokumentationszwecken. Sie wird deshalb in der Suchergebnis-Liste für jede gefundene Messung angezeigt (siehe Abbildung 6.5). Der hier dargestellte Anwendungsfall betrifft die Situation, in der man von Vornherein nicht weiß, wie die einzufahrende Fahrszene aussehen wird. Wie in früheren Kapiteln erwähnt, handelt es

Bild 6.5: Vorschau-Video in der Szenariendatenbank

sich in diesem konkreten Fall um eine Stausituation. In diesem Fall setzt sich ein Entwickler vor das Datenbank-Interface und sucht zunächst nach einer Szene im Video, welche er beschreiben möchte. Anschließend erstellt er mittels Interface für die Statechart-basierte Modellierung von Fahrszenen eine generische Fahrszenenbeschreibung und speichert sie ab. Sofort stellt sich die Frage,wie präzise denn solche Beschreibungen sind. Man muss hier zugeben, dass allein aus den Video-Daten solche Werte wie Fahrzeug-Geschwindigkeiten oder Beschleunigungen nur schätzungsweise angegeben werden können. Der Ablauf einzelner Aktivitäten, also Fahrmanöver, kann aber ohne Schwierigkeiten modelliert werden. Eine ebenfalls berechtigte Frage an dieser Stelle ist „Warum probiert man nicht den umgekehrten Weg und versucht die formalen Fahrszenenbeschreibungen automatisch aus den Messungen zu gewinnen?“ Nun dazu müsste man auf die Algorithmen zur Szenenanalyse zugreifen, welche von den Ingenieuren selbst entwickelt worden sind. Das würde funktionieren, allerdings hat es, aus Sicht des Testens, wenig Sinn Fahrszenen automatisch mit Hilfe von denselben Algorithmen zu beschreiben, die man auch testen möchte. Es muss vielmehr eine gewisse Diskrepanz vorhanden sein: Der Mensch beschreibt in diesem konkreten Anwendungsfall die Testdaten sowie das Soll-Verhalten. Die Testdaten werden vom Algorithmus abgearbeitet und das Ergebnis wird mit dem Soll-Verhalten verglichen. Eine berechtigte Frage ist auch „Warum soll ein Entwickler nach der Fahrt noch eine Fahrszene beschreiben?“. Nun wurde in den früheren Kapiteln erwähnt, dass neben realen Mess-und Testfahrten ebenfalls SiL-Testläufe statfinden. Oftmals möchte man bestimmte Fahrszenen, welche in der Realität eingefahren wurden, in der Simulation überprüfen. Dazu werden aber bestimmte Parameter von den Verkehrsteilnehmern leicht verändert. Dazu gehören beispielsweise Abstände von EGO zu anderen Verkehrsteilnehmern. So möchte man zum Beispiel das Verhalten von Adaptive Cruise Control beim dichten Auffahren des EGO-Fahrzeugs von hinten auf ein Testobjekt in der Simulation überprüfen. Bislang musste man eine Fahrszene für SiL-Simulation komplett neu mit dem entsprechenden simulationsabhängigen Interface beschreiben. Mit dem in dieser Arbeit vorgestellten Ansatz kann ein Entwickler aus einer einmal erzeugten generischen Fahrszenenbeschreibung (egal ob vor oder nach der Fahrt) mehrere SiL-spezifische Fahrszenenbeschreibungen anlegen. Dazu geht er genauso wie im vorigen Anwendungsfall dargestellt vor: Nach dem Abspeichern einer gerade erstellten oder dem Auswählen einer generischen Beschreibung wählt der Entwickler die Option „Verfeinern“. Anschließend muss aus dem Projektbaum das gewünschte Projekt sowie innerhalb davon SiL als Testverfahren gewählt werden. Im nächsten Schritt muss der Entwickler die SiL-spezifischen Attribute mit Werten belegen. Die generischen bleiben bestehen. Dies steigert nicht nur die Wiederverwendbarkeit, sondern verringert auch den zeitlichen Aufwand zur Erstellung einer Testspezifikation. Denn wie bereits erwähnt, gewinnt man mittels der Trans-

formation aus einer solchen plattformunabhängigen Beschreibung nicht nur plattformabhängige Testdatenbeschreibung in Form einer XML-Datei, sondern auch sämtliche für den automatisierten SiL-Testlauf notwendigen Daten in den jeweiligen Formaten und sogar inklusive der Paramterisierungsdateien für die Testablauf-Steuerungsumgebung. Ein automatisierter SiL-Testvorgang kann folgendermaßen aussehen (siehe Abb. 6.6). Das bereits im Kapitel 1 beschriebene Simu-

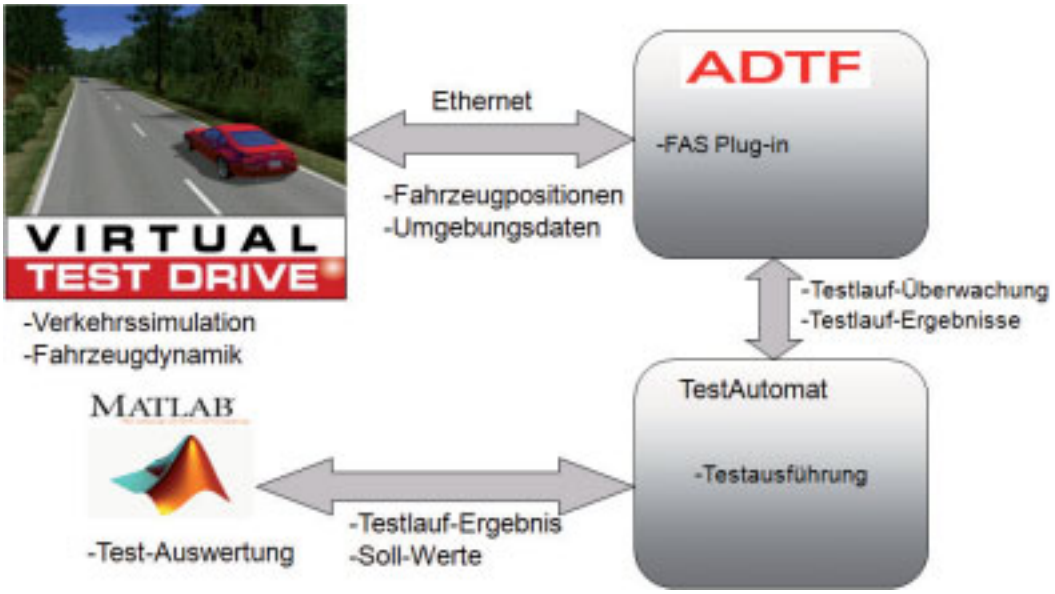


Bild 6.6: Automatiserter SiL Testlauf

lationswerkzeug „Virtual Test Drive“ ist über eine Ethernet-Schnittstelle an den PC mit ADTF angebunden. Zur Ablaufsteuerung wird ein in C-Sharp entwickeltes Framework (Testautomat) [EMW08] verwendet. Dieses ist nach dem „Pipes and Filters“ Architekturmuster aufgebaut (siehe Abbildung 6.7). Vom Framework selbst werden solche allgemeinen Funktionalitäten zur Verfügung gestellt wie Laden, Initialisierung sowie Laufzeitüberwachung einzelner Plug-ins. Die eigentlichen Testablauf-relevanten Funktionalitäten werden in den einzelnen Plug-ins realisiert. So ist zum Beispiel ein Plug-in dafür zuständig, die Testdaten aus der Datenbank herunterzuladen, es überprüft beispielsweise auch, ob vor dem Herunterladen genügend Speicherplatz auf der Festplatte vorhanden ist. Ein anderes Plug-in ist für Initialisieren, Starten sowie Laufzeitüberwachung von ADTF zuständig. Dieses Plug-in initialisiert alle notwendigen Umgebungsvariablen und den eigenen Überwachungsmechanismus und stellt die Aufrufzeile von ADTF zusammen. Während des Ablaufs wird sowohl der Prozess selbst überwacht, in dem ADTF läuft, als auch die Meldungen, die ADTF an eine Mailslot-Adresse [Mic10c] schickt. Sollte dabei eine Meldung mit dem Status „Error“ gelesen werden, wird der Ablauf abgebrochen. Sollte der ADTF ins Stocken

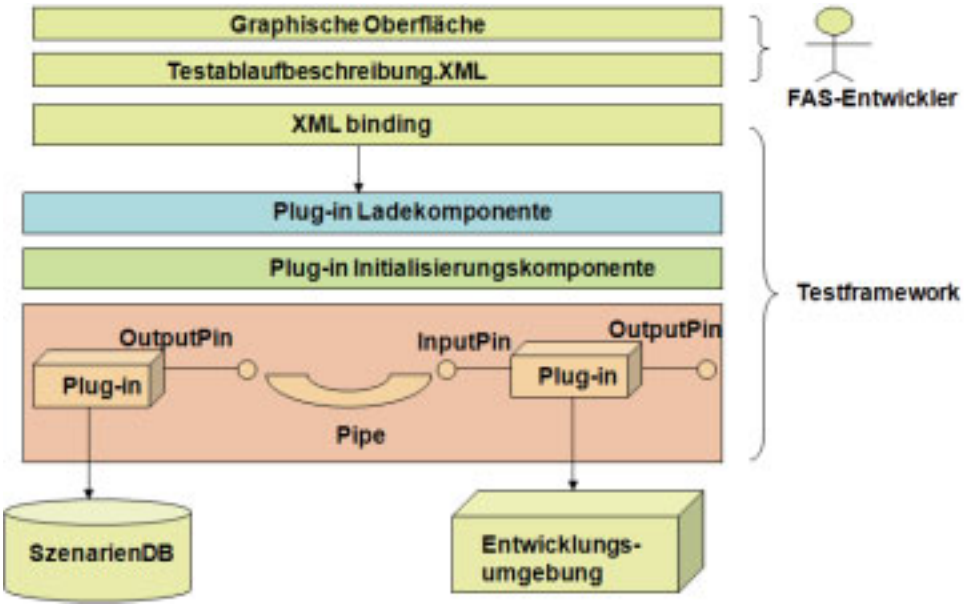


Bild 6.7: Testautomat: Architektur

kommen, weil beispielsweise im Funktionscode ein Pufferüberlauf eingetreten ist, dann sorgt der Testautomat dafür, dass der Ablauf gestoppt wird und die Abarbeitung des nächsten Testfalls beginnt. Der Testautomat sorgt ebenfalls dafür, dass, falls vom Entwickler gewünscht, am Ende der Abarbeitung eines jeden Testfalls dessen Auswertung gestartet wird, indem beispielsweise ein Matlab-Skript angestoßen wird. Schließlich ist der Testautomat dafür zuständig, dass nach der Abarbeitung des einen Testfalls die des nächsten beginnt. Das hier beschriebene Framework ist bereits in den Entwicklungsprozess der Abteilung integriert: Es wurden für drei verschiedene Projekte damit Dauerläufe für die Testvorgänge mit realen Messdaten durchgeführt. Die Dauer des längsten von ihnen betrug sechs Stunden.

Abschließend lässt es sich feststellen, dass das SiL-Format zur Beschreibung von Fahrszenen recht umfangreich ist, es beinhaltet beispielsweise solche Größen wie Motorleistung und Fahrzeuggewicht. Solche Größen müssen nicht jedesmal vom Entwickler bei der Testspezifikationserstellung eingegeben werden. Sie können vielmehr festkodiert und während der Transformation automatisch zum Zielformat hinzugefügt werden.

6.1.3 Ansatzanwendung in einem konkreten Projekt

In diesem Abschnitt wird ein konkretes Projekt aus dem Bereich Adaptive Cruise Control Entwicklung vorgestellt. Anschließend wird auf eine mögliche Anwendung des Ansatzes in diesem Projekt eingegangen. Die nachfolgende Projektbeschreibung wurde aus [SBS+09] übernommen.

„Bis heute werden die meisten Fahrerassistenzsysteme nur subjektiv bewertet (z.B. ACC Annäherungsverhalten). Ziel eines innovativen und durchgängigen Entwicklungsprozesses muss jedoch die einheitliche und objektive Bewertung der Systemeigenschaften und so der Fahrzeugeigenschaften sein. Dadurch ergeben sich zwei wesentliche Vorteile. Zum einen können unterschiedliche Systemausprägungen objektiv verglichen werden, zum anderen kann die Fahrerassistenzfunktion mit den im Kapitel 1 vorgestellten Entwicklungsmethoden (SiL, HiL, ViL) in der Simulation bewertet oder das zukünftige Systemverhalten prognostiziert werden. Abbildung 6.8 zeigt das im Weiteren betrachtete Vorgehen zur Erstellung der Fahrerassistenz-Eigenschaftsspinne auf, welches sich am Bewertungsprozess zur klassischen Fahrdynamik orientiert. Bei diesem

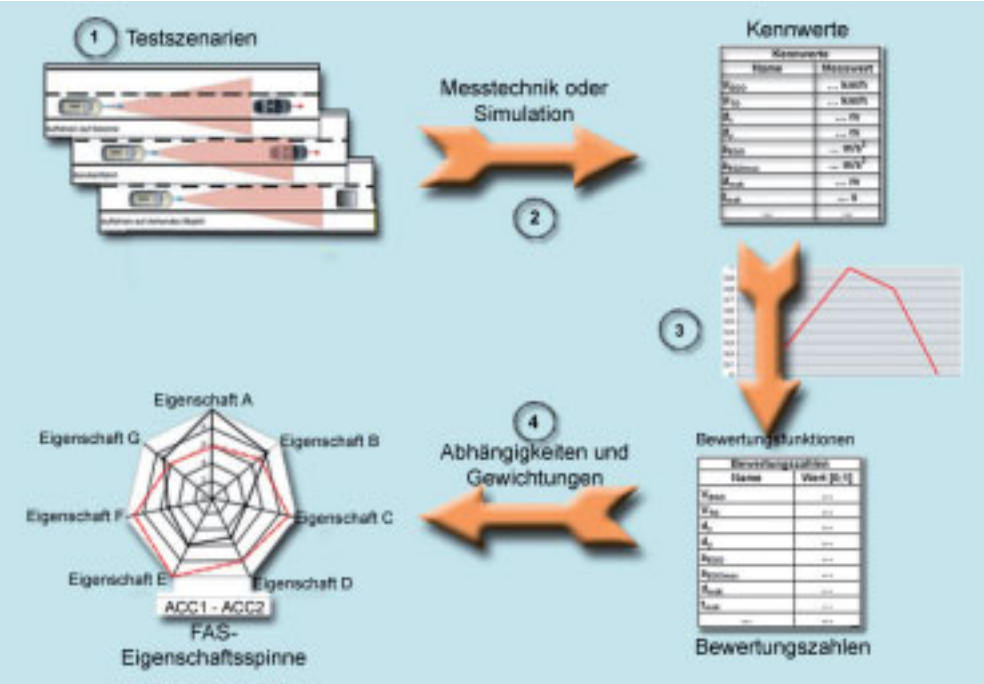


Bild 6.8: Vorgehensweise zur Erstellung einer Eigenschaftsspinne [SBS+09]

Bewertungsprozess werden über Testszenarien Kennwerte ermittelt, diese dann normiert und zu Fahrzeugeigenschaften zusammengefasst.

- Schritt 1: Die Auswahl geeigneter Testszenarien (Fahrmanöver) ist für die Güte der Bewertung von entscheidender Bedeutung.
- Schritt 2: Eine weitere Herausforderung beim Realtest ist die messtechnische Erfassung der fahrphysikalischen Kennwerte. Hierfür musste ein Referenzsystem aufgebaut werden,

das aus einer hochgenauen DPGS-Inertialsensorplattform besteht. Durch die Ausstattung jedes am Testszenario beteiligten Fahrzeuges mit einem dieser Systeme können so Größen der Eigendynamik und der Umwelt wie der exakte Abstand, Relativgeschwindigkeiten oder -beschleunigungen erfasst werden. Damit die durchgeführten Fahrmanöver zusätzlich einheitlich und reproduzierbar sind, muss das Zielfahrzeug mit einem Lenk- und Bremsrobotersystem ausgestattet werden.

- Schritt 3: Um aus den so ermittelten Ergebnissen die Fahrzeugeigenschaften berechnen zu können, wird jeder Kennwert über eine Bewertungsfunktion in eine dimensionslose Bewertungszahl überführt. Die Bewertungsfunktion stellt hierbei das jeweilige Sollverhalten dar, welches durch Studien im Vorfeld festgelegt werden muss.

- Schritt 4: Über Abhängigkeiten und Gewichtungen der Bewertungszahlen werden abschließend die Eigenschaften und so die Fahrerassistenz-Eigenschaftsspinne ermittelt. Dies ermöglicht die geforderte objektive Bewertung der untersuchten Systeme beziehungsweise Systemausprägungen“.

Aus dem vorgestellten Projekt wird ersichtlich, dass zur Bewertung von Kriterien mehrere Testverfahren eingesetzt werden. Der Ausgangspunkt beziehungsweise die Testdaten für das jeweilige Testverfahren ist ein Fahrszenario. Aktuell muss jedes Szenario für jede Testmethode in einem eigenen Format zusammengestellt werden (Siehe Kapitel 1). Das kostet Zeit. Den Zeitaufwand kann man reduzieren, indem ein einziges Mal eine generische Fahrszenenbeschreibung zusammengestellt und in der Datenbank abgespeichert wird. Anschließend kann sie für das jeweilige Testverfahren verfeinert werden (siehe Abschnitt 6.1.1). Außerdem können die Soll-Werte mit in der Testspezifikation plattformunabhängig abgespeichert werden. Schließlich können im Transformationstemplate Parameter festkodiert werden, welche zum Starten eines automatisierten Testlaufs z.B. mit SiL notwendig sind (Zugangsdaten zur Datenbank, Parametrisierungen zum Ansteueren von ADTF, Aufrufparameter für einen Matlab-Auswertungsskript, in welchem die Bewertungsfunktionen implementiert sind etc.). Somit werden neben der Testspezifikation im plattformspezifischen Format ebenfalls Zusatzinformationen generiert, die zum Starten z.B. eines SiL-Testlaufs mit dem oben dargestellten Testautomaten notwendig sind. Aktuell müssen die Dateien per Hand editiert und zusammengetragen werden. Dies ist ebenfalls wesentlich zeitaufwändiger als eine automatische Generierung.

6.2 Interviews mit den Entwicklern

6.2.1 Aufbau des Interviews

Es wurden insgesamt 6 Einzel-Interviews a eine Stunde und 15 Minuten durchgeführt. Während dieser mussten die Entwickler zunächst eine generische Fahrszene modellieren, die auf einem Blatt mit Hilfe eines Piktogramms und einer kurzen Beschreibung dargestellt wurde. Anschließend wurde den Entwicklern der Verfeinerungs- und der Transformationsschritt gezeigt. Dazu wurde als Zielformat eine PDF-Tabelle gewählt für den Bereich realer Messfahrten. Schließlich wurde die Möglichkeit aufgezeigt, aus einer generischen Fahrszenenbeschreibung nach einem Verfeinerungsschritt eine XML-Datei zu erzeugen, welche die Fahrszenenbeschreibung für Virtual Test Drive beinhaltet. Die zu modellierende Fahrszene ist in der Abbildung 6.9 dargestellt. Das EGO-Fahrzeug sowie ein weiteres Testobjekt fahren konstant auf der zweiten Spur mit

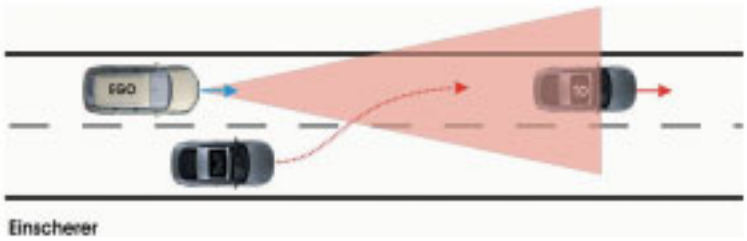


Bild 6.9: Die zu modellierende Fahrszene

einer Geschwindigkeit von 50 km/h. Das dritte Testobjekt fährt konstant auf der ersten Spur mit einer Geschwindigkeit von 50 km/h. Nun schert das Testobjekt auf der ersten Spur ein. Anschließend bremst das EGO-Fahrzeug auf 40 km/h ab. Ebenfalls wurde dem Entwickler als Hilfestellung eine Vorgehensweise vorgelegt, nach deren Durchlaufen er zu einer vollständigen Szenarienbeschreibung gelangen würde. Diese sieht folgendermaßen aus:

- Schritt 1: Überlege, wieviele Verkehrsteilnehmer es in der Beschreibung geben soll?
- Schritt 2: Lege die Szene im Editor entsprechend an!
- Schritt 3: Modelliere Start-Zustand (Drag and Drop)!
- Schritt 4: Modelliere End-Zustand (Drag and Drop)!
- Schritt 5: Modellliere alle Setup-Zustände (Drag and Drop in den jeweiligen Subzustand, der durch die Überschrift „Traffic Participant“ gekennzeichnet ist)!

- Schritt 6: Modelliere (Drag and Drop) und ordne alle Aktionszustände in der Zeit-Achse an!
- Schritt 7: Verbinde die Zustände mit den Transitionen. Ignoriere dabei zunächst durch das Anklicken der Taste „Finish“ die Transitionseinstellungen!
- Bestimme für jede Transition die entsprechenden Einstellungen (Event, Condition, Action): a. Zeitliche Abhängigkeiten zwischen den Zuständen. b. Eventuell die Dauer einzelner Zustände (Falls notwendig!)!
- Schritt 8: Belege die Zustandsattribute mit konkreten Werten!
- Schritt 9: Speichere die erstellte Beschreibung in der Datenbank ab!

6.2.2 Feedback von den Entwicklern

Generelles Feedback von den Entwicklern war, dass der Ansatz sinnvoll ist und dass er u.a. eine Vereinheitlichung der Fahrszenenbeschreibung für die FAS-Domäne darstellt. Das zu modellierende Beispiel wurde verstanden. Die Entwickler haben ebenfalls den Unterschied zwischen generischer und projekt- bzw. testartspezifischer Beschreibung verstanden und somit auch den Verfeinerungsschritt (siehe Abschnitt 4.5) und die damit verbundene Wiederverwendbarkeit von Testspezifikationsbeschreibungen. Die Zeitersparnis durch die Festkodierung bestimmter sich nicht ändernder Parameter in der Testspezifikation wurde ebenfalls erkannt. Das vorgestellte Prototyp soll nun so weiterentwickelt werden, dass man es produktiv in der Abteilung einsetzen kann, um mehr Erfahrung damit zu sammeln. Dies gilt sowohl für den Bereich realer Messfahrten als auch für den Bereich SiL-Testläufe. Im weiteren werden nun die einzelnen ermittelten Problematiken katgorisiert dargestellt.

Übergang von räumlicher Darstellung zur Zeit-bezogener

Eine wichtige Erkenntnis ist, dass die Modellierung von Fahrszenen mit den Statecharts einen Übergang von schwerpunktmäßig räumlicher Darstellung einer Fahrszene zur vordergründig Zeit-bezogenen darstellt. Ein Entwickler hat im Bewusstsein meistens eine räumliche Darstellung („Wo fährt welches Fahrzeug und was tut es“). Die Modellierung mit den Statecharts betrachtet eine Fahrszene vordergründig aus zeitlicher Perspektive. Die räumliche Darstellung wird ebenfalls berücksichtigt in Form von Attributen jeweiliger Zustände. Die Hauptschwierigkeit bestand darin, die relative Position eines Testobjekts zum Eigenfahrzeug in der formalen Notation anzugeben,

da dies einen gewissen Abstraktionsschritt darstellt. Die von den Entwicklern vorgeschlagene Lösung besteht darin, die Notation von relativen Fahrzeugpositionen interaktiv zu gestalten (Abbildung 6.10). Jedes Piktogramm im Bild soll anklickbar sein. Im Hintergrund wird die jeweilige

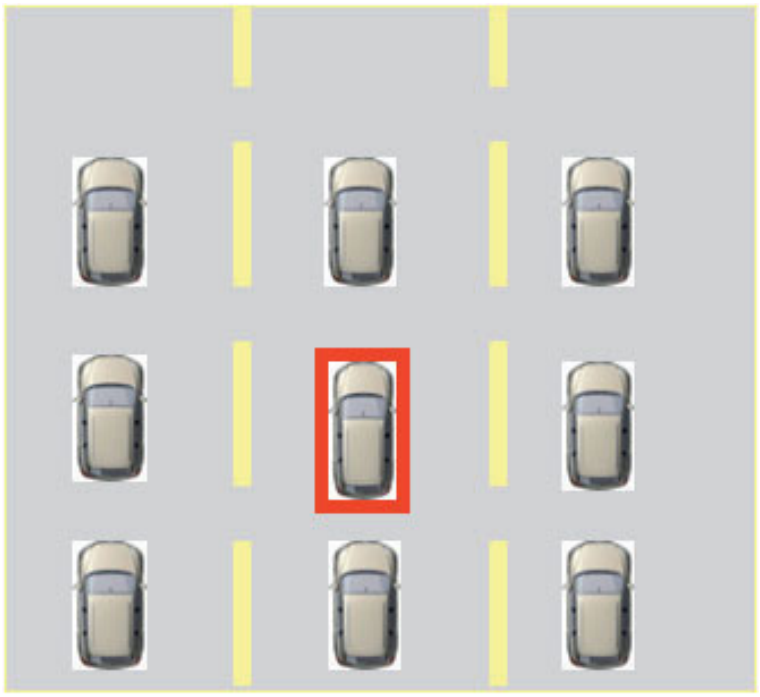


Bild 6.10: interaktive Notation von Fahrzeugpositionen

ausgewählte Position in einen Integer umgewandelt und als entsprechendes Attributwert („CurrentRelativePosition“) im Zustand eingetragen. Dasselbe betrifft die Bestimmung von Fahrspuren, in welchen die einzelnen Verkehrsteilnehmer fahren.

Ein anderer Aspekt, welcher ebenfalls einen Abstraktionsschritt darstellt, ist die Tatsache, dass alle Zustände, die das Verhalten eines Fahrzeugs darstellen, in einem kompositen Subzustand modelliert werden müssen. Bei der Modellierung führte dies bei einigen Entwicklern zum Problem, dass sie die atomaren Zustände einfach neben dem globalen Zustand modelliert haben, ohne diese in die jeweiligen kompositen Subzustände per „Drag and Drop“ „hineinzuziehen“. Die Lösung dieses Problems wäre, dass man bereits beim Anlegen eines globalen Zustands den Entwickler in einem Dialog fragt, was die Setup-Zustände für die einzelnen Teilnehmer wären. Anschließend werden sie vom System in den jeweiligen kompositen Subzuständen korrekt platziert. Somit weiß der Entwickler bereits, wo er die Aktionszustände modellieren soll.

Weiterhin hat ein Entwickler versucht, die einzelnen atomaren Zustände aus unterschiedlichen kompositen Subzuständen mit einer Transition zu verbinden. Nun, dies wurde durch eine Assistenzfunktion unterbunden, da die Notation so etwas nicht erlaubt.

Eine weitere Frage, die die Entwickler gestellt haben, betraf die Modellierung von zeitlichen Abhängigkeiten zwischen den Zuständen: „Was passiert, wenn der Zustand B nach Zustand A vorkommt, er wird vom Entwickler aber fälschlicherweise an der Zeitachse vor Zustand A modelliert?“ Nun, dazu wurde schon von Professor Harel ein Ereignis-Synchronisationsmechanismus vorgeschlagen (Siehe Abschnitt 5.1.2) und in dieser Arbeit implementiert.

Interface-bezogene Anmerkungen

Die Entwickler wünschten sich eine bessere Schrift, bessere Benennung von Modellierungselementen sowie die Darstellung der Zeitachse unterhalb vom globalen Zustand. All dies lässt sich problemlos anpassen. Ebenfalls haben sich die Entwickler gewünscht, dass die Start- sowie Endzustände automatisch beim Abspeichern der Beschreibung hinzugefügt werden, damit sie diese nicht manuell modellieren müssen. Dies lässt sich ebenfalls mit geringem Aufwand implementieren.

Zielformate

Bei dem PDF-Dokument haben sich die Entwickler neben einer Tabelle auch ein Piktogramm gewünscht, das aus der formalen Beschreibung generiert wird. Dies ist ebenfalls machbar.

Generelle Modellierungsfragen

In dieser Kategorie haben sich die Entwickler dafür interessiert, wo man Wetterverhältnisse modellieren soll. Dafür ist der globale Zustand vorgesehen, wo dann angegeben werden kann, ob es geregnet hat oder ob die Fahrbahn trocken war. Eine andere Frage betraf die automatische Generierung von Straßenverläufen für VTD. Dies ist eine separate Arbeit, in welcher die während einer Messfahrt aufgenommenen GPS-Daten in das spezielle XML-Streckenformat [Vir10a] transformiert werden. Eine interessante Frage betraf den Unterschied zwischen einer Statechart-basierten Fahrszenen-Beschreibung und einer Formular bzw. Attribut-basierten. Nun, eine Attribut-basierte Beschreibung wird bereits von MDM und der Szenariendatenbank verwendet. Diese erlaubt es, eine Messung mit einer minimal notwendigen Menge an Attributen zu beschreiben. Was sich auf diese Art und Weise nicht oder nur sehr umständlich beschreiben lässt, ist die komplexe zeitliche Abfolge der Aktionen mehrerer Verkehrsteilnehmer. Insbesondere würde sich die Beschreibung von kausalen Abhängigkeiten mühsam gestalten. Eine Formular-basierte Fahrszenenbeschreibung würde somit beispielsweise nicht den Präzisionsgrad bei der Suche in

der Datenbank liefern wie die Statechart-basierte. Außerdem könnte man aus einem Formular nur schwer beispielsweise ein Steuerungsprogramm für den Roboter ableiten, welcher eine bestimmte Abfolge von Fahrmanövern beim Testen einer Pre-Crash Funktion abfährt. Der hier vorgestellte Ansatz besitzt einen wesentlich höheren Grad an Formalität, der auch die Transformation in einen programmiersprachlichen Code wesentlich erleichtert.

Zum Schluß brachten die Entwickler ein gewichtiges Argument, der für die Verwendung eines domänenspezifischen Metamodells statt UML spricht. Die Entwickler fanden die Verwendung von Bedingungen in der Statechart-Notation überflüssig und wünschten sich, dass alles durch Ereignisse modelliert wird. Würde man allerdings die Bedingungen aus einem UML-Metamodell ausschließen, würde es dem OMG-Standard für diese Sprache nicht mehr entsprechen. UML wäre somit nicht einsetzbar. Ein domänenspezifisches Metamodell kann hingegen so gestaltet werden, dass dort keine Bedingungen vorkommen.

Abschließend lässt sich feststellen, dass der vorgestellte Ansatz sicherlich eine gewisse Lernzeit erfordert. Nach den Eindrücken aus dieser Evaluierung sollte diese jedoch wesentlich geringer sein als bei vergleichbaren Modell-basierten Ansätzen wie Matlab/Simulink/Stateflow oder einem MDA-basierten Ansatz wie EXAM, zu welchem umfangreiche Übungsunterlagen existieren und ebenfalls mehrstündige Übungseinheiten stattfinden. Beim hier vorgestellten Ansatz hat es circa 30-40 Minuten gedauert, bis die Entwickler das Prinzip begriffen haben.

6.3 Analyse der gesetzten Ziele

Im Folgenden werden die am Anfang dieser Arbeit definierten Ziele im Hinblick auf ihre Erfüllung betrachtet. Dabei wird jedes im Kapitel 1 definierte Ziel einzeln analysiert.

- Vereinheitlichung der Testspezifikationsbeschreibung: Das entwickelte Vorgehensmodell schreibt die Wahl einer Szenarienmodellierungsmethode für die gesamte Anwendungsdomäne vor. In dieser Domäne soll die gewählte Methode durchgängig von allen beteiligten Projekten und Abteilungen angewandt werden. Die kleinste gemeinsame Basis für alle am Software-Entwicklungsprozess Beteiligten ist die generische Szenarienbeschreibung, welche über Projektgrenzen hinweg ausgetauscht werden kann. Somit ist das Ziel der Vereinheitlichung erreicht.

- Formatunabhängigkeit der Testspezifikationsbeschreibung: Ein wichtiger Beitrag dieser Arbeit ist die Definition eines domänenunabhängigen Metamodells (Basis-Metamodell), welches als Ausgangsbasis für die Definition eines beliebigen domänenspezifischen Meta-

modells zur formalen Definition von Testspezifikationen dient. Die Formatunabhängigkeit ist durch das Vorhandensein eines solchen Metamodells garantiert, denn die erzeugten Instanzen, also konkrete Testspezifikationen, sind formal und können somit in jede beliebige Zielnotation transformiert werden. Die Transformationsfähigkeit wurde zudem durch eine konkrete Konzeptanwendung in der Domäne der Fahrer-Assistenzsysteme unter Beweis gestellt. Es wurde also gezeigt, dass zum einen eine Testspezifikation sich auf die im Vorgehensmodell dargelegte Art und Weise formalisieren lässt. Zum anderen wurde auch konkret demonstriert, dass aus praktischer Sicht eine Menge von Technologien und Werkzeugen existiert, mit denen sich die so definierten Testspezifikationen transformieren lassen. Wichtig zu erwähnen ist, dass durch die Formatunabhängigkeit die Testspezifikationen sich auf einer höheren semantischen Ebene beschreiben lassen. Somit sind sie auch für Dritte wesentlich verständlicher als beispielsweise proprietäre Skripte zur Automatisierung von Testvorgängen.

- Mehrplattform-Fähigkeit der Testspezifikationsbeschreibung: Dieses Ziel wird im Wesentlichen durch die Formatunabhängigkeit erreicht. Denn deren Vorhandensein setzt im Prinzip die Transformierbarkeit einer Testspezifikation in mehrere Zielformate voraus und somit auch die Mehrplattform-Fähigkeit.

- Schaffung der notwendigen Voraussetzung zum Testwissenstransfer zwischen den frühen Entwicklungsphasen eingebetteter reaktiver Systeme und den nachgelagerten: Das entwickelte Vorgehensmodell spezifiziert, wie die Auswahl der Technologien und die Realisierung einer konkreten technischen Infrastruktur zur Verwaltung, Transformation und Wiederverwendbarkeit der Testspezifikationen geschehen soll. Zudem wurde mittels prototypischer Implementierung in der FAS-Domäne eine technische Infrastruktur geschaffen, durch deren Nutzung auch die nachgelagerten Entwicklungsstufen auf das bereits erzeugte und formalisierte Testwissen zugreifen können. Selbstverständlich müssen für die nachgelagerten Stufen Projekt- beziehungsweise Testverfahren-spezifische Attribute definiert werden sowie die entsprechenden Transformationstemplates. Der Aufwand dafür sollte sich in Grenzen halten, denn die Definition der spezifischen Attribute ist, technisch gesehen, lediglich mit dem Eintragen in die entsprechenden Datenbanktabellen verbunden. Ebenfalls lassen sich die Transformationstemplates durch die Einfachheit der Transformationssprache relativ schnell programmieren. Eventuell können sogar die Templates der vorgelagerten Entwicklungsstufen zumindestens teilweise wiederverwendet werden. Wichtig zu erwähnen ist, dass diese Infrastruktur die bereits in der Abteilung vorhandene IT-Landschaft berücksichtigt.

- Nahtlose Integration der vorhandenen Methodik in den täglichen Entwicklungsprozess: Das vorgestellte Vorgehensmodell wurde mit Hilfe von Aktivitätsdiagrammen spezifiziert, wodurch sich bei sehr komplexen Software-Entwicklungsprozessen und großer Anzahl von beteiligten Projekten Prozessunterstützungslösungen implementieren lassen, welche die Einführung sowie die Einhaltung des Vorgehensmodells erleichtern.

Ein wichtiges erreichtes Ziel ist selbstverständlich die Tatsache, dass die für die Anwendungsdomäne geschaffene technische Infrastruktur, die Wiederverwenbarkeit von Testspezifikationen sowohl bei realen Messfahrten als auch bei SiL ermöglicht. Dadurch wird eindeutig Zeit gespart.

Kapitel 7

Zusammenfassung und Ausblick

In diesem Kapitel wird zum einen das Erreichte zusammengefasst (Abschnitt 7.1). Im darauffolgenden Abschnitt 7.2 werden einige mögliche Weiterentwicklungsmöglichkeiten des Ansatzes aufgezeigt.

7.1 Zusammenfassung des Erreichten

In dieser Arbeit wurde ein umfassendes Vorgehensmodell zur formalen Definition von Testspezifikationen für reaktive eingebettete Systeme entwickelt. Das Vorgehensmodell basiert auf dem Grundgedanken der Modell-getriebenen Software-Entwicklung und ist im Hinblick auf den Einsatz in der industriellen Umgebung erschaffen worden. Das Vorgehensmodell erlaubt die nahtlose Einführung der szenarienbasierten Definition von formalen und domänenspezifischen Testspezifikationen im täglichen Entwicklungsprozess einer oder mehrerer kooperierenden Abteilungen.

Neben dem benannten Vorgehensmodell wurde, wie bereits angedeutet, ein Ansatz zur Definition plattformunabhängiger, szenarienbasierter Testspezifikationen entwickelt. Dieser hat die Wiederverwendbarkeit von solchen formalen Testspezifikationen als höchstes Ziel sowohl innerhalb der Projekte, welche in den jeweiligen Abteilungen laufen, als auch über die Grenzen einzelner Abteilungen hinweg.

Eines der Konzepte bei dem Ansatz ist die Idee, die zu testende Domäne durch ein Metamodellbasiertes Wörterbuch einzuschränken, aus welchem der Entwickler eine solche Testspezifikation erzeugt. Es besteht die Möglichkeit, jedes Element, auf mehreren Abstraktionsstufen durch entsprechende Attribute zu beschreiben, wodurch die Wiedereverwendbarkeit gesteigert wird.

Ein anderer Grundpfeiler des vorgestellten Ansatzes ist die Möglichkeit, die szenarienbasierten

Testspezifikationen in einem Zielplattform-unabhängigen Format zu definieren, um daraus in einem Transformationsschritt diverse plattformabhängige Zielformate zu erzeugen.

Außerdem wurde ein Metamodell definiert, in welchem domänenunabhängig die Elemente definiert sind, die, sollte man sich für die Verwendung des vorgestellten Ansatzes entscheiden, als Basis genommen werden müssen. Dies bedeutet insbesondere, dass von ihnen domänenspezifische Meta-Modellierungselemente abgeleitet werden müssen. Dadurch wird die Universalität des Ansatzes verdeutlicht.

Im praktischen Bereich dieser Arbeit ging es darum, zu zeigen, dass alles theoretisch Angedachte sich auch umsetzen lässt. Dazu wurde zum einen das konzipierte Vorgehensmodell auf die Domäne der Fahrer-Assistenzsysteme angewandt. Als Ergebnis ist ein Domänen-Metamodell zur formalen Definition der Testspezifikationen für Fahrer-Assistenzsysteme mit Hilfe von Statecharts entstanden. Zum anderen erfolgte prototypische Implementierung des Ansatzes für eine Vorentwicklungsabteilung im Bereich Fahrwerk und Elektronik. Dabei musste sicherlich im Vorfeld untersucht werden, welche Technologien und Werkzeuge bereits auf dem Markt existieren und welche noch eigenständig zu entwickeln sind.

Schließlich ging es am Ende darum, den Ansatz zu evaluieren, indem Interviews mit den Entwicklern durchgeführt und ausgewertet wurden. Es wurde festgestellt, dass der Ansatz sinnvoll ist. Ebenfalls wurden Vorschläge zur Weiterentwicklung dokumentiert.

Neben der wissenschaftlichen Arbeit wurde in der Abteilung ebenfalls eine Infrastruktur zur automatisierten Durchführung von Blackbox-Tests erzeugt, deren Kern ein Framework bildet [EMW08]. Dieses ist imstande, unterschiedlichste am Testlauf beteiligte Anwendungen zu einem automatisierten Ablauf zu integrieren.

7.2 Ideen zur Weiterentwicklung

Zugegebenermaßen ließ sich nicht alles Angedachte in den drei Jahren dieses Projekts realisieren. Deshalb wird nun darauf eingegangen.

7.2.1 Praktische Weiterentwicklung

Was die praktische Weiterentwicklung speziell in der Domäne vorausschauender FAS angeht, so muss noch folgendes geschehen:

- Es müssen die Ergebnisse der zweiten Iteration im Vorgehensmodell implementiert werden. Diese betreffen primär die Verbesserungen an der graphischen Benutzeroberfläche, welche sich während der Evaluierung herauskristallisiert haben (siehe Abschnitt 6.2).

- Es muss eine Suchkomponente implementiert werden, welche die Suche nach bestimmten Abfolgen von Fahrszenen ermöglicht. Die Suche geschieht selbstverständlich auf deren Statechart-basierten Beschreibungen.

- Es müssen weitere Attribute für die Beschreibungselemente gesammelt, strukturiert und im System eingetragen werden. Dementsprechend müssen auch die Transformationstemplates weiterentwickelt werden.

- Es muss eine komfortable Administrator-Oberfläche geschaffen werden, mit welcher man schnell und unkompliziert neue Kategorien, Beschreibungselemente sowie deren Attribute eintragen, umstrukturieren und löschen kann.

7.2.2 Wissenschaftliche Weiterentwicklung

Was den wissenschaftlichen Aspekt dieser Arbeit betrifft, so gibt es hier noch zwei Richtungen, in welchen in der Zukunft gearbeitet werden kann. Zum einen hat es sich bei Gesprächen mit den Entwicklern herauskristallisiert, dass es nicht nur syntaktischer Überprüfung der Testspezifikationen bedarf, sondern auch semantischer. In der aktuellen Situation könnte man auch den Sachverhalt modellieren, dass das Fahrzeug in einem Tunnel fährt und es in diesem regnet. Um solche Absurditäten zu vermeiden, bedarf es wissenschaftlicher Untersuchungen, mit welchen Methoden man auch die semantische Korrektheit der Testspezifikationen unmittelbar vor dem Einfügen in eine Datenbank überprüfen kann. Dies ist mit Sicherheit eine eigene Promotion wert. Denkbar wäre hier der Ansatz, bei dem eine formale Testspezifikation in eine logische Sprache transformiert wird (z.B. Ontology Web Language [W3C10b] oder Prolog [Hod01]), auf welcher semantische Überprüfungen automatisiert stattfinden können. Wird hier ein Fehler entdeckt, wird die Testspezifikation nicht in die Datenbank eingetragen.

Ein zweiter Aspekt betrifft die Möglichkeit, aus solchen formalen Testspezifikationen so genannte Benutzungsmodelle [SEGK09] automatisch zu generieren. Ein Benutzungsmodell stellt alle möglichen Interaktionen eines Benutzers mit der FUT dar. In [SEGK09] werden solche Benutzungsmodelle im Rahmen des statistischen Testens mit Hilfe von Markov-Ketten spezifiziert. Die Hauptidee besteht darin, aus einzelnen bereits existierenden Interaktionbeschreibungen, wie sie in der Anwendungsdomäne durch Fahrszenen beschrieben werden, ein solches Benutzungsmodell automatisiert zu erstellen. Im nächsten Schritt sollen daraus ebenfalls automatisiert neue, noch nicht getestete Fahrszenen generiert und in der Szenendatenbank abgespeichert werden. Dazu

bedarf es der Definition eines Algorithmus, welcher aus einzelnen Fahrszenen ein komplettes Benutzungsmodell erstellt. Zum anderen müssen auch formale Kriterien definiert werden, nach denen neue Fahrszenen generiert werden. Der komplette Vorgang ist in der Abb. 7.1 aufgezeigt.

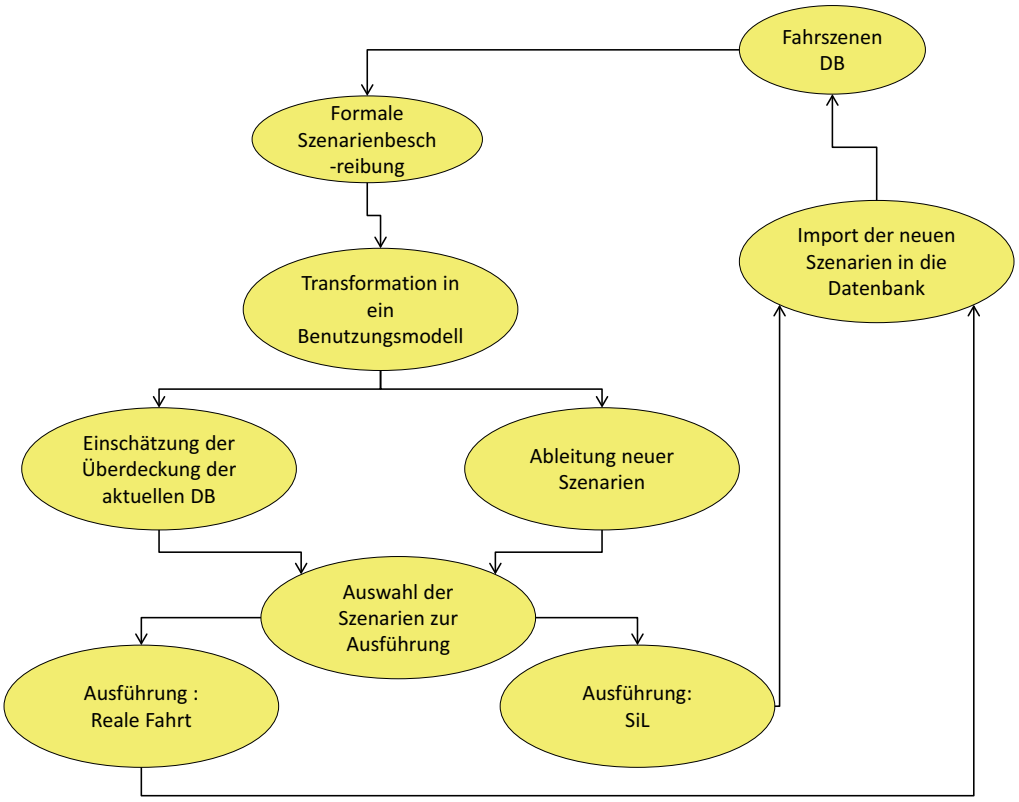


Bild 7.1: Gewinnung von neuen Szenarien aus den Benutzungsmodellen

Verzeichnis der Bilder

Literaturverzeichnis

[AD94] ALUR, Rajeev ; DILL, David L.: A theory of timed automata. In: *Theoretical Computer Science* 126 (1994), Nr. 2, 183 - 235. http://dx.doi.org/DOI:10.1016/0304-3975(94)90010-8. – ISSN 0304–3975

[Ado10] ADOBE: *Portable Document Format*. http://www.adobe.com/products/acrobat/adobepdf.html. Version: 2010. – Letzter Besuch: 18.01.2010

[AHR00] AALST, W. van d. ; HEE, K. van ; REIJERS, H.: Analysis of Discrete-time Stochastic Petri Nets. In: *Statistica Neerlandica*, Blackwell Publishing, 2000, S. 237–255. http://dx.doi.org/10.1111/1467-9574.00139

[Alb10] ALBERT, A.: *Plug-on device*. http://www.plugondevice.com/. Version: 2010. – Letzter Besuch: 20.01.2010

[Alt] ALT, O.: *Integration textueller Anforderungen und Modell-basiertem Testen mit SysML*. http://pi.informatik.uni-siegen.de/. – Letzter Besuch: 21.01.2010

[ATS10] ATS SOFTWARE RESEARCH AND CONSULTING: *Classification Tree Editor CTE 2.0 für MS Windows*. http://user.cs.tu-berlin.de/~sepradm/ws9900/cte/cte20.doc.html. Version: 2010. – Letzter Besuch: 18.01.2010

[Aud07] AUDI: Extended Automation Method / Audi. 2007. – Konzeptpapier

[Aud10] AUDI: *Mess Data Management*. https://www.mdm-community.org/index.php?option=com_frontpage&Itemid=1. Version: 2010. – Letzter Besuch: 20.01.2010

[Aut07] AUTOCLUB EUROPA: *Hitliste Autopannen 2007*. http://www.ace-online.de/download/. Version: 2007. – Letzter Besuch: 17.01.2010

[Aut10] AUTOMOBILITY: *Antiblockier-System*. http://www.autostandards.de/abs-A6.html. Version: 2010. – Letzter Besuch: 17.01.2010

[Bak05] BAKER, P.: *Model-Driven Testing: Using the UML Testing Profile*. Berlin,Heidelberg : Springer, 2005. – ISBN 978–3–540–72562–6

[Bar10] BARTZ, Rainer: ASAM ODS / Association for Standardisation of Automation and Measuring Systems. Version: 2010. http://www.asam.net/index.php?option=com_content&task=view&id=117&Itemid=207. – Specification. – Online–Ressource. Letzter Besuch: 20.01.2010

[Bea09] BEAULIEU, A.: *Einführung in SQL*. Beijing : O'Reilly, 2009. – ISBN 978–3–89721–937–3

[Ben08] BENZ, Sebastian: AspectT: aspect-oriented test case instantiation. In: *AOSD '08: Proceedings of the 7th international conference on Aspect-oriented software development*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–044–9, S. 1–12. http://dx.doi.org/http://doi.acm.org/10.1145/1353482.1353484

[BH97] BHARADWAJ, Ramesh ; HEITMEYER, Constance: Verifying SCR Requirements Specifications Using State Exploration. In: *In Proceedings of First ACM SIGPLAN Workshop on Automatic Analysis of Software*, 1997, S. 9–24

[BHO89] BALCER, M. ; HASLING, W. ; OSTRAND, T.: Automatic generation of test scripts from formal test specifications. In: *SIGSOFT Softw. Eng. Notes* 14 (1989), Nr. 8, S. 210–218. http://dx.doi.org/http://doi.acm.org/10.1145/75309.75332. – ISSN 0163–5948

[Boc08] BOCK, T.: *Vehicle in the Loop - Test- und Simulationsumgebung für Fahrerassistenzsysteme*. München, Deutschland, Technische Universität München, Dissertation, 2008

[Bos10] BOSCH: *Controller Area Network*. http://www.semiconductors.bosch.de/en/20/can/index.asp. Version: 2010. – Letzter Besuch: 18.01.2010

[BPSM+10] BRAY, T. ; PAOLI, J. ; SPERBERG-MCQUEEN, C.M. ; MALER, E. ; YERGEAU, F.: *Extensible Markup Language*. http://www.w3.org/TR/2006/REC-xml-20060816/#sec-origin-goals. Version: 2010. – Letzter Besuch: 17.01.2010

[Bun10] BUNDESMINISTERIUM FÜR BILDUNG UND FORSCHUNG: *Invent: Das Ergebnisbericht*. `http://www.invent-online.de`. Version: 2010. – Letzter Besuch: 18.01.2010

[CGP00] CLARKE, E. ; GRUMBERG, O. ; PELED, D.: *Model Checking*. USA : MIT Press, 2000. – ISBN 0–262–03270–8

[Cha10] CHANG, K.: *Conception and Implementation of a Transformation Tool for the Generation of Platform-Dependent Test-Specification descriptions in Driver-Assistance Systems*. Erlangen, Deutschland, Friedrich-Alexander Universität Erlangen-Nürnberg, Studienarbeit, Februar 2010

[Cza04] CZARNECKI, Krzysztof: Overview of generative software development. In: *In Proceedings of Unconventional Programming Paradigms (UPP) 2004, 15-17 September, Mont Saint-Michel, France, Revised Papers*, Springer-Verlag, 2004, S. 313–328

[Dai04] DAI, ZR.: Model-Driven Testing with UML 2.0 / Fraunhofer FOCUS, Berlin, Deutschland. 2004. – Technical Report

[DDS03] DEUSSEN, Peter H. ; DIN, George ; SCHIEFERDECKER, Ina: A TTCN-3 Based Online Test and Validation Platform for Internet Services. In: *ISADS '03: Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)*. Washington, DC, USA : IEEE Computer Society, 2003. – ISBN 0–7695–1876–1, S. 177

[DH98] DAMM, W. ; HAREL, D.: LSCs: Breathing Life into Message Sequence Charts. In: *Formal Methods in System Design*, Kluwer Academic Publishers, 1998, S. 293–312. `http://dx.doi.org/10.1023/A:1011227529550`

[DJG03] DIETSCHE, K.H. ; JÄGER, T. ; GMBH, Bosch: *Kraftfahrtechnisches Taschenbuch*. Vieweg `http://ebooks.ub.uni-muenchen.de/17691/`

[DKV00] DEURSEN, Arie van ; KLINT, P. ; VISSER, J.M.W.: Domain-Specific Languages / Annotated Bibliography. ACM SIGPLAN Notices. DRAFT. 2000. – Forschungsbericht

[dSp10a] DSPACE: *Automotive Simulation Models*. `http://www.dspace.de/ww/de/gmb/home/products/sw/automotive_simulation_models.cfm`. Version: 2010. – Letzter Besuch: 20.01.2010

[dSp10b] DSPACE: *Control Desk*. http://www.dspace.de. Version: 2010. – Letzter Besuch: 20.01.2010

[Ecl] ECLIPSE FOUNDATION: *Eclipse Modelling Framework*. http://www.eclipse.org/modeling/emf/. – Letzter Besuch: 21.01.2010

[Ecl04] ECLIPSE FOUNDATION: *Java Emitter Templates*. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html. Version: 2004. – Letzter Besuch: 22.01.2010

[Ecl10a] ECLIPSE FOUNDATION: *MOF Script*. http://www.eclipse.org/gmt/mofscript/. Version: 2010. – Letzter Besuch: 20.01.2010

[Ecl10b] ECLIPSE FOUNDATION: *Open Architecture Ware Framework*. http://www.eclipse.org/workinggroups/oaw/. Version: 2010. – Letzter Besuch: 20.01.2010

[EGZ09] ENTIN, Vladimir ; GANSLMEIER, Thomas ; ZAWICKI, Krystian: Formale und formatunabhängige Fahrszenarienbeschreibung für automatisierte Testvorgänge im Bereich der Entwicklung von Fahrer-Assistenzsystemen. In: FISCHER, Stefan (Hrsg.) ; MAEHLE, Erik (Hrsg.) ; REISCHUK, Rüdiger (Hrsg.): *Informatik 2009 Im Focus das Leben. Beiträge der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*. Luebeck, 2009. – ISBN 978–3–88579–248–2, S. 329

[EMW08] ENTIN, Vladimir ; MEYER-WEGENER, Klaus: Ein Framework für die Testautomatisierung bei Fahrer-Assistenz-Systemen. In: MAALEJ, Walid (Hrsg.) ; BRÜGGE, Bernd (Hrsg.): *Software Engineering (Workshops)* Bd. 122, GI, 2008. – ISBN 978–3–88579–216–1, S. 395–398

[EMWL08] ENTIN, V. ; MEYER-WEGENER, K. ; LOEBEL, C.: Model-Based Automation of the Test Processes in the Pre-Development of Driver Assistance Systems. In: GÜHMANN, C. (Hrsg.): *Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik II*. Renningen : Expert, 2008. – ISBN 978–3–816–92818–8, S. 112–119

[EN02] ELMASRI, R. ; NAVATHE, S.: *Grundlagen von Datenbanksystemen*. USA : Addison-Wesley, 2002. – ISBN 386894012X

[Ent06] ENTIN, V.: *Eine Datenbank von Messdaten aus Fahrszenarien für Tests von Fahrerassistenzsystemen*. Erlangen, Deutschland, Department Informatik, Lehrstuhl 6 für Datenmanagement der Friedrich-Alexander Universität Erlangen-Nürnberg, Diplomarbeit, Oktober 2006

[Ent09a] ENTIN, V.: Specification of the modelling elements of the statechart notation for the description of the real driver scenes / Friedrich-Alexander-Universität Erlangen-Nürnberg. Erlangen, 2009. – Internal Paper

[Ent09b] ENTIN, V.: Wahl der graphischen Modellierungsmethode zur Darstellung von Fahrszenarien / Friedrich-Alexander-Universität Erlangen-Nürnberg. Erlangen, 2009. – Internes Dokument

[ESK+09] ENTIN, V. ; SIEGL, S. ; KERN, A. ; REICHEL, M. ; MEYER-WEGENER, K.: A Scenario-Centric Approach for the Definition of the Formal Test Specifications of Reactive Systems. In: *TAIC PART '09: Proceedings of the 2009 Testing: Academic and Industrial Conference. Practice and Reserach Techniques*. Los Alamitos, CA, USA : IEEE, 2009. http://dx.doi.org/10.1109/TAICPART.2009.1

[Ext08] EXTESSY: EXACT Guide / Extessy AG. 2008. – Dokumentation

[Ext10] EXTESSY AG: *EXACT*. http://www.extessy.com/de/?id=3b7efa09444a31c5d58596e5bbf87d47. Version: 2010. – Letzter Besuch: 18.01.2010

[FAM06] FIGUEIREDO, André L. L. ; ANDRADE, Wilkerson L. ; MACHADO, Patrícia D. L.: Generating interaction test cases for mobile phone systems from use case specifications. In: *SIGSOFT Softw. Eng. Notes* 31 (2006), Nr. 6, S. 1–10. http://dx.doi.org/http://doi.acm.org/10.1145/1218776.1218788.– ISSN 0163–5948

[Far05] FARRENKOPF, Armin: Einführung in die Fahrzeugelektronik / Ohm-Hochschule, Nürnberg, Deutschland. 2005. – Vorlesungsunterlagen

[Fed10] FEDERAL MOTOR CARRIER SAFETY ADMINISTRATION: *Lane Departure Warning Systems*. http://www.fmcsa.dot.gov/facts-research/research-technology/report/lane-departure-warning-systems.htm. Version: 2010. – Letzter Besuch: 17.01.2010

[FG09] FRASER, G. ; GARGANTINI, A.: An Evaluation of Specification Based Test Generation Techniques Using Model Checkers. In: GUERRERO, J. (Hrsg.): *Testing: Academic and Industrial Conference - Practice and Research Techniques, 2009. TAIC PART 09*. Los Alamitos USA : IEEE, 2009, S. 72–81. http://dx.doi.org/10.1109/TAICPART.2009.1

[FL03] FRANK, U. ; LAAK, B. van: Anforderungen an Sprachen an Modellierung von Geschäftsprozessen / Institut für Wirtschaftsinformatik, Universität Koblenz. Version: 2003. http://www.uni-koblenz.de/%7Eiwi/publicfiles/Arbeitsberichte/Nr34.pdf. Koblenz, 2003. – Arbeistbericht. – Online–Ressource

[For10] FORMAL SYSTEMS: *FDR2: a refinement checker*. http://www.fsel.com/software.html. Version: 2010. – Letzter Besuch: 20.01.2010

[GDD09] GASEVIC, D. ; DJURIC, D. ; DEVEDZIC, V.: *Model Driven Engineering and Ontology Development*. Berlin, Heidelberg : Springer, 2009. – ISBN 3642002811

[Geb00] GEBHARDT, A.: *Rapid Prototyping - Werkzeuge für die schnelle Produktentstehung*. München : Hanser, 2000. – ISBN 3–446–21242–6

[GG93] GROCHTMANN, M. ; GRIMM, K.: Classification Trees for Partition Testing, Software Testing. In: *Verification and Reliability*. Hoboken : Wiley, 1993, S. 63–82

[GH99] GARGANTINI, Angelo ; HEITMEYER, Constance: Using model checking to generate tests from requirements specifications. In: *SIGSOFT Softw. Eng. Notes* 24 (1999), Nr. 6, S. 146–162. http://dx.doi.org/http://doi.acm.org/10.1145/318774.318939. – ISSN 0163–5948

[GH08] GONZALEZ-PEREZ, C. ; HENDERSON-SELLERS, B.: *Metamodelling for Software Engineering*. Chichester, UK : Wiley, 2008. – ISBN 0470030364

[GHR+03] GRABOWSKI, Jens ; HOGREFE, Dieter ; RÉTHY, György ; SCHIEFERDECKER, Ina ; WILES, Anthony ; WILLCOCK, Colin: An introduction to the testing and test control notation (TTCN-3). In: *Comput. Netw.* 42 (2003), Nr. 3, S. 375–403. http://dx.doi.org/http://dx.doi.org/10.1016/S1389-1286(03)00249-4. – ISSN 1389–1286

[Gom01] GOMEZ, M.: *Hardware-in-the-Loop Simulation*. http://www.embedded.com/story/OEG20011129S0054. Version: 2001. – Letzter Besuch: 20.01.2010

[Ham05] HAMILL, P.: *Unit Test Frameworks*. Sebastopol : O'Reilly Media, 2005. – ISBN 0–596–0689–6

[Har87] HAREL, David: Statecharts: A visual formalism for complex systems. In: *Sci. Comput. Program.* 8 (1987), Nr. 3, S. 231–274. http://dx.doi.org/http://dx.doi.org/10.1016/0167-6423(87)90035-9. – ISSN 0167–6423

[Har09] HARSÁNYI, G.: *SensEdu - an Internet-Based Short Course in Sensorics*. http://www.sensedu.com/menu.html. Version: 2009. – Letzter Besuch: 18.01.2010

[HBC+96] HEWETT ; BAECKER ; CARD ; CAREY ; GASEN ; MANTEI ; PERLMAN ; STRONG ; VERPLANK: *ACM SIGCHI Curricula for Human-Computer Interaction*. http://old.sigchi.org/cdg/cdg2.html#2_1. Version: 1996. – Letzter Besuch: 18.01.2010

[Hei02] HEITMEYER, C.: Software Cost Reduction / Naval Research Laboratory. Version: 2002. http://chacs.nrl.navy.mil/5540/publications/CHACS/2002/2002heitmeyer-encse.pdf. Washington, 2002. – Encyclopedia of Software Engineering. – Online–Ressource

[HKKR05] HITZ, M. ; KAPPEL, G. ; KAPSAMMER, E. ; RETSCHITZEGGER, W.: *UML At Work. Objektorientierte Modellierung mit UML 2*. Heidelberg, Deutschland : dpunkt, 2005. – ISBN 3–89864–261–5

[Hoa78] HOARE, C. A. R.: Communicating sequential processes. In: *Commun. ACM* 21 (1978), Nr. 8, S. 666–677. http://dx.doi.org/http://doi.acm.org/10.1145/359576.359585. – ISSN 0001–0782

[Hod01] HODGSON, JPE.: *Prolog: ISO Norm*. http://pauillac.inria.fr/~deransar/prolog/docs.html. Version: 2001. – Letzter Besuch: 22.01.2010

[Hol04] HOLZMANN, G.J.: *The Spin Model Checker: Primer and Reference Manual*. Boston, USA : Pearson, 2004. – ISBN 0–321–22862–6

[HPSR87] HAREL, D. ; PNUELI, A. ; SCHMIDT, JP. ; R, Sherman: On the Formal Semantics of Statecharts. In: *Symposium on Logic in Computer Science*. USA : IEEE, 1987

[IAV05] IAV: Fahrszenarienkatalog für die Automatische Wegverkürzung / IAV. 2005. – Katalog

[Imm07] IMMEN, Stephan: *Jahresbericht des Kraftfahrt-Bundesamtes*. http://www.kba.de/. Version: 2007. – Letzter Besuch: 17.01.2010

[Int10] INTERNATIONAL BUSINESS MACHINES: *Rational Doors*. http://www-142.ibm.com/software/products/de/de/ratidoor. Version: 2010. – Letzter Besuch: 20.01.2010

[Ise08] ISERNHAGEN, H.: Softwaretest in Verbindung mit einer automatisierten Testdatenauswertung. In: GÜHMANN, C. (Hrsg.): *Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik II*. Renningen : Expert, 2008. – ISBN 3–816–92818–8, S. 378–387

[ITU04] ITU: Message Sequence Charts / International Telecommunication Union. 2004. – Recommendation

[Jan04] JANOCHA, H.: *Actuators - basics and applications*. Berlin : Springer, 2004. – ISBN 3–540–61564–4

[JSW07] JAVED, A. Z. ; STROOPER, P. A. ; WATSON, G. N.: Automated Generation of Test Cases Using Model-Driven Architecture. In: *AST '07: Proceedings of the Second International Workshop on Automation of Software Test*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2971–2, S. 3. http://dx.doi.org/http://dx.doi.org/10.1109/AST.2007.2

[Kar07] KARLSCH, M.: *A model-driven framework for domain specific languages*. Potsdam, Hasso-Plattner-Institute of Software Systems Engineering, Master's Thesis, January 2007

[Lam03] LAMBERG, K.: Durchgängiges, automatisiertes Testen bei der Entwicklung von Automobilelektronik. In: *1. Tagung Simulation und Test in der Funktions- und Softwareentwicklung* (2003)

[Lam06] LAMBERG, Klaus: Model-based testing of automotive electronics. In: GIELEN, Georges G. E. (Hrsg.): *DATE*, European Design and Automation Association, Leuven, Belgium, 2006. – ISBN 3–9810801–0–6, S. 91

[Las10] LASSALLE TECHNOLOGIES: *AddFlow*. http://www.lassalle.com/. Version: 2010. – Letzter Besuch: 22.01.2010

[Löb08] LÖBEL, C.: *ADTF: Framework für Fahrerassistenz- und Sicherheitssysteme*. FISITAWorldAutomotiveCongress. Version: 2008

[LBEO04] LAMBERG, K. ; BEINE, M. ; ESCHMANN, M. ; OTTERBACH, R.: Model-Based Testing Of Embedded Automotive Software Using Mtest. In: *In-Vehicle Networks and Software, Electrical Wiring Harnesses, and Electronics and Systems Reliability* (2004). ISBN 0–7680–1319–4

[Lig09] LIGGESMEYER, Peter: *Software-Qualität*. Spektrum Akademischer Verlag http://ebooks.ub.uni-muenchen.de/17691/

[Lu94] LU, Peng: Test case generation for specification-based software testing. In: *CASCON '94: Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, 1994, S. 41

[Man05] MANTELL, K.: *From UML to BPEL*. http://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/. Version: 2005. – Letzter Besuch: 21.01.2010

[Mat10a] MATHWORKS: *Matlab*. http://www.mathworks.de/. Version: 2010. – Letzter Besuch: 18.01.2010

[Mat10b] MATHWORKS: *Simulink*. http://www.mathworks.de/products/simulink/. Version: 2010. – Letzter Besuch: 18.01.2010

[MBB02] MELLOR, S. ; BALCER, JM. ; BALCER, M.: *Executable UML: A Foundation for Model-Driven Architecture*. Boston-USA : Addison-Wesley, 2002. – ISBN 0–201–74804–5

[MDBS08] MEEL, Franciscus van ; DUBA, Georg-Peter ; BOCK, Thomas ; STRASSER, Benedikt: Developing Properties for Driver Assistance Systems by Means of an Inovative and Constant Development Process. In: *FISITA 2008 World Automotive Congress - Springer Automotive Media* II (2008)

[Mer07] MERHING, C.: *Open Architecture Ware*. Münster, Deutschland, Westfälische Wilhelms-Universität, Seminararbeit, Mai 2007

[Met10] METHOD PARK: *Stages - Die Prozessmanagement-Suite*. http://www.methodpark.de/produkte/stages-process-management-suite-ueberblick/. Version: 2010. – Letzter Besuch: 20.01.2010

[Mic] MICROSOFT CORPORATION: *Erste Schritte mit Windows Forms*. http://msdn.microsoft.com/de-de/library/ms229601(VS.80).aspx. – Letzter Besuch: 21.01.2010

[Mic10a] MICRONOVA: *Extended Automation Method*. http://www.exam-ta.de/. Version: 2010. – Letzter Besuch: 20.01.2010

[Mic10b] MICROSOFT: *EXCEL*. http://office.microsoft.com/de-de/excel/FX100487621031.aspx?ofcresset=1. Version: 2010. – Letzter Besuch: 18.01.2010

[Mic10c] MICROSOFT CORPORATION: *Mailslots*. http://msdn.microsoft.com/en-us/library/aa365576%28VS.85%29.aspx. Version: 2010. – Letzter Beusch: 22.01.2010

[MSUW04] MELLOR, S. ; SCOTT, K. ; UHL, A. ; WEISE, D.: *Model-Driven Testing: Using the UML Testing Profile*. Boston, USA : Addison-Wesley, 2004. – ISBN 0–201–78891–8

[Mü07] MÜLLER, S.: *Modellbasierte IT-Unterstützung von wissensintensiven Prozessen. Dargestellt am Beispiel medizinischer Forschungsprozesse*. Erlangen, Deutschland, Friedrich-Alexander-Universität Erlangen-Nürnberg, Diss., 2007

[MW05] MITSCHKE, M. ; WALLENTOWITZ, H.: *Dynamik der Kraftfahrzeuge*. Heidelberg+New York : Springer, 2005. – ISBN 3–540–42011–8

[Net10] NETWORK WORKING GROUP: *ASCII format for Network Interchange*. http://tools.ietf.org/html/rfc20. Version: 2010. – Letzter Besuch: 20.01.2010

[Obj] OBJECT MANAGEMENT GROUP: *XML Metadata Interchange*. http://www.omg.org/technology/documents/formal/xmi.htm. – Letzter Besuch: 21.01.2010

[Obj08a] OBJECT MANAGEMENT GROUP: *Business Model Process and Notation*. http://www.omg.org/spec/BPMN/1.1/. Version: 2008. – Letzter Besuch: 21.01.2010

[Obj08b] OBJECT MANAGEMENT GROUP: MOF Model to Text Transformation / OMG. Version: 2008. http://www.omg.org/spec/MOFM2T/1.0/. – Specification. – Online–Ressource. Letzter Besuch: 20.01.2010

[Obj09] OBJECT MANAGEMENT GROUP: *Unified Modelling Language*. http://www.omg.org/technology/documents/formal/uml.htm. Version: 2009. – Letzter Besuch: 20.01.2010

[Obj10a] OBJECT MANAGEMENT GROUP: *Meta Object Facility*. http://www.omg.org/mof/. Version: 2010. – Letzter Besuch: 20.01.2010

[Obj10b] OBJECT MANAGEMENT GROUP: *Query View Transformation*. http://www.omg.org/spec/QVT/1.0/. Version: 2010. – Letzter Besuch: 20.01.2010

[Obj10c] OBJECT MANAGEMENT GROUP: *UML Testing Profile*. http://www.omg.org/technology/documents/formal/test_profile.htm. Version: 2010. – Letzter Besuch: 20.01.2010

[ORLS06] OKIKA, Joseph C. ; RAVN, Anders P. ; LIU, Zhiming ; SIDDALINGAIAH, Lokesh: Developing a TTCN-3 test harness for legacy software. In: *AST '06: Proceedings of the 2006 international workshop on Automation of software test*. New York, NY, USA : ACM, 2006. – ISBN 1–59593–408–1, S. 104–110. http://dx.doi.org/http://doi.acm.org/10.1145/1138929.1138950

[Par96] PARASHKEVOV, J.: ARC - a tool for efficient refinement and equivalence checking for CSP. In: *IEEE Int. Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP)*. Washington, DC, USA : IEEE Computer Society, 1996. – ISBN 0–7695–1876–1, S. 68–75

[Pet06] PETROV, I.: *Meta-data, Meta-Modelling and Query Processing in Meta-data Repository Systems*. Deutschland : Shaker, 2006

[Poh08] POHL, K.: *Requirements Engineering*. Heidelberg, Deutschland : dpunkt, 2008. – ISBN 978–3–89864–550–8

[Rec08] RECH, J.: *Ethernet: Technologien und Protokolle für die Computervernetzung*. Heidelberg : Heise, 2008. – ISBN 978–3–936931–40–2

[Rei85] REISIG, Wolfgang: *Petri nets: an introduction*. New York, NY, USA : Springer-Verlag New York, Inc., 1985. – ISBN 0–387–13723–8

[SA03] SI ALHIR, S.: *Learning UML*. Sebastopol, USA : O'Reilly, 2003. – ISBN 0–596–00344–7

[SBS+09] STRASSER, B. ; BOCK, T. ; SIEDERSBERGER, K.H. ; MAURER, M. ; BUBB, H.: Vernetzung von Test- und Simulationsmethoden für Fah- rerassistenzsysteme. In: *VDI-Berichte* (2009)

[SC08] STEINWART, I. ; CHRISTMANN, A.: *Support Vector Machines*. New York : Springer, 2008. – ISBN 978–0–387–77241–7

[Sch96] SCHEER, AW.: *ARIS-House of Business Engineering: Von der Geschäftsprozeßmodellierung zur Workflow-gesteuerten Anwendung, vom Business Process Reengineering zum Continuous Process Improvement*. `http://www.iwi.uni-sb.de/Download/iwihefte/heft133.pdf`. Version: 1996. – Letzter Besuch: 20.01.2010

[Sch01] SCHEER, AW.: *ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen*. Heidelberg : Springer, 2001. – ISBN 3–540–41601–3

[Sch06] SCHWARZER, R.: *ATLAS Model Management Architecture*. Leipzig, Deutschland, Universität Leipzig, Seminararbeit, Februar 2006

[Sch08] SCHMID, H.: Hardware-in-the-Loop Technologie: Quo Vadis? In: GÜHMANN, C. (Hrsg.): *Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik II*. Renningen : Expert, 2008. – ISBN 3–816–92818–8, S. 195–202

[SEGK09] SIEGL, Sebastian ; ENTIN, Vladimir ; GERMAN, Reinhard ; KIFFE, Gerhard: Model Driven Testing with Time Augmented Markov Chain Usage Models - Computations and Test Case Generation Algorithms for Time Augmented Markov Chain Usage Models. In: SHISHKOV, Boris (Hrsg.) ; CORDEIRO, José (Hrsg.) ; RANCHORDAS, Alpesh (Hrsg.): *ICSOFT (1)*, INSTICC Press, 2009. – ISBN 978–989–674–009–2, S. 202–207

[SL05] SPILLNER, A. ; LINZ, T.: *Basiswissen Software-Test*. Heidelberg : dpunkt, 2005. – ISBN 3–89864–358–1

[Spa] SPARX SYSTEMS: *Enterprise Architect*. `http://www.sparxsystems.com.au/`. – Letzter Besuch: 21.01.2010

[SRH90] STONEBRAKER, M. ; ROWE, L. A. ; HIROHAMA, M.: The Implementation of POSTGRES. In: *IEEE Trans. on Knowl. and Data Eng.* 2 (1990), Nr. 1, S. 125–142. http://dx.doi.org/http://dx.doi.org/10.1109/69.50912. – ISSN 1041–4347

[Sun] SUN MICROSYSTEMS: *The source for Java developers*. http://java.sun.com/. – Letzter Besuch: 21.01.2010

[SVEH07] STAHL, Thomas ; VÖLTER, Markus ; EFFTINGE, Sven ; HAASE, Arno: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. 2. Heidelberg : dpunkt, 2007. – ISBN 978–3–89864–448–8

[SZ06] SCHÄUFFELE, J. ; ZURAWKA, T.: *Automotive Software Engineering*. Wiesbaden : Vieweg, 2006. – ISBN 3–8348–0051–1

[Tan08] TANENBAUM, A.: *Verteilte Systeme*. München, Deutschland : Pearson Studium, 2008. – ISBN 978–3–8273–7293–2,3–8273–7293–3

[Tho05] THOMS, K.: *Codegenerierung mit dem openArchitectureWare Generator 3.0*. http://www.itemis.de/itemis-ag/publikationen/fachartikel/language=de/6417/oaw-teil-1-codegenerierung-mit-oaw-3-0. Version: 2005. – Letzter Besuch: 22.01.2010

[Toa] TOADSOFT: *Toad: Market-leading tool that provides quick and easy database development and administration*. http://www.toadsoft.com/. – Letzter Besuch: 21.01.2010

[van99] VAN DER AALST, W. M. P.: Formalization and verification of event-driven process chains. In: *Information and Software Technology* 41 (1999), Nr. 10, 639 - 650. http://dx.doi.org/DOI:10.1016/S0950-5849(99)00016-6. – ISSN 0950–5849

[VDA10a] VDA: *Automotive SPICE*. http://www.automotivespice.com/. Version: 2010. – Letzter Besuch: 17.01.2010

[VDA10b] VDA QUALITÄTSMANAGEMENT-CENTER: *Umfang von Automotive Spice*. http://www.vda-qmc.de/software-prozesse/automotive-spice/. Version: 2010. – Letzter Besuch: 18.01.2010

[Vir10a] VIRES: *OpenDirve-Format*. http://www.opendrive.org/. Version: 2010. – Letzter Besuch: 18.01.2010

[Vir10b] VIRES: *Virtual Test Drive*. http://www.audi.com/aev/brand/de/Projekte_und_Tools/VirtualTestDrive.html. Version: 2010. – Letzter Besuch: 22.02.2010

[W3C10a] W3C: *HTML 5 Specification*. dev.w3.org/html5/spec/Overview.html. Version: 2010. – Letzter Besuch: 18.01.2010

[W3C10b] W3C: *Ontology Web Language: Overview*. http://www.w3.org/TR/owl-features/. Version: 2010. – Letzter Besuch: 22.01.2010

[WDT+05] WILLCOCK, C. ; DEISS, T. ; TOBIES, S. ; KEIL, S. ; ENGLER, F. ; SCHULZ, S.: *An Introduction to TTCN-3*. Chichester, UK : Wiley, 2005. – ISBN 10–0–470–01224–2

[Whi08] WHITE, S.: *Introduction to BPMN*. http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf. Version: 2008. – Letzter Besuch: 21.01.2010

[WHW09] WINNER, H. ; HAKULI, S. ; WOLF, G.: *Handbuch: Fahrerassistenz-Systeme*. Wiesbaden : Vieweg+Teubner, 2009. – ISBN 978–3–8348–0287–3

[Wol08] WOLTERS, U.: Langfristige Investitionssicherung in der Testautomatisierung. In: GÜHMANN, C. (Hrsg.): *Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik II*. Renningen : Expert, 2008. – ISBN 3–816–92818–8, S. 120–129