

4.1.3 Modell-basierter Test von Simulink/Stateflow-Modellen

Mirko Conrad, Ines Fey

DaimlerChrysler AG, Berlin, Germany, {mirko.conrad | ines.fey}@daimlerchrysler.com

ZUSAMMENFASSUNG

Die Modell-basierte Entwicklung hat sich in wichtigen Einsatzbereichen als Standardparadigma für die Erstellung von Steuergerätesoftware etabliert. In der Automobilindustrie hat sich hierbei Simulink/Stateflow als de-facto Standard-Werkzeug für die Modellierung durchgesetzt.

Um die Wirksamkeit und Effizienz des Tests im Rahmen der Modell-basierten Entwicklung (Modell-basierter Test) zu gewährleisten, bedarf es einer methodischen Vorgehensweise, die den Spezifika dieses Entwicklungsparadigmas Rechnung trägt und insbesondere die im Modell enthaltenen testrelevanten Informationsinhalte für den Testprozess nutzbar macht.

Da die Anwendung eines einzelnen Testverfahren für einen gründlichen Test i.d.R. nicht ausreichend ist, müssen verschiedene komplementäre Testverfahren (test design techniques) für Simulink/Stateflow-Modelle geeignet miteinander kombiniert werden. Für eine solche effektive Teststrategie für den Modell-basierten Test hat sich eine Kombination aus funktionalen und strukturellen Testverfahren auf Modellebene (MIL) mit anschließender Wiederverwendung der so ermittelten Testszenarien für den SIL, PIL und HIL Test bewährt.

Ausgehend von der effektiven Teststrategie für den Modell-basierten Test gibt der Beitrag einen Überblick über aktuell vorfindliche Testverfahren und zugehörige Werkzeuge für Simulink/Stateflow-Modelle und ordnet diese den einzelnen Bereichen der effektiven Teststrategie zu. Dieser Überblick kann als Entscheidungshilfe für die Auswahl einer geeigneten Methoden- und Werkzeugkombination im jeweiligen Projektzusammenhang dienen.

BLOCK-ORIENTIERTE MODELLIERUNG UND MODELL-BASIERTE ENTWICKLUNG

Die mathematische Modellbildung, in vielen Ingenieurwissenschaften bereits seit langem etabliert, gewinnt auch bei der Entwicklung eingebetteter Software stark an Bedeutung. Bei der Entwicklung automobiler Steuergeräte wird die mathematische Modellbildung zum einen zur konzeptionellen Vorwegnahme der zu realisierenden Funktionalität der eingebetteten Software (Regler, Steuerung) sowie zum anderen zur Simulation des Verhaltens realer physikalischer Systeme (Strecke) eingesetzt.

Ein Modell besteht aus Funktionsblöcken mit fest definierten Ein- und Ausgängen. Komponenten werden im Blockdiagramm durch gerichtete Kanten zwischen ihren Schnittstellen miteinander verbunden (Abb. 1). Im mathematischen Modell repräsentieren sie damit Gleichungen, welche die Schnittstellenvariablen verschiedener Komponenten in Beziehung setzen. Die Verbindungslinien repräsentieren kausal motivierte Wirkungsrichtungen, welche die Ausgänge eines Blocks als Eingänge eines anderen definieren). Komponenten innerhalb der Hierarchie können andere Komponenten aggregieren oder elementar sein (vgl. /ML00/).

Die Modellierung erfolgt häufig mit blockorientierten, kommerziellen Modellierungs- und Simulationswerkzeugen wie der Matlab- (Matlab, Simulink, Stateflow, siehe /TMW/), MATRIXx - (Xmath, SystemBuild, siehe /NI/) oder ASCET-SD-Produktfamilie (ASCET-SD, siehe /ETAS/). Diese stellen Beschreibungssprachen, Berechnungstechniken und Interpreter / Compiler bereit. Sie unterstützen die Entwicklung und Definition von System- / Softwarekomponenten, ihrer Verbindungen und Schnittstellen durch (semi-)formale graphische Spezifikationen des Modells mittels editierbarer, hierarchischer Blockschaltbilder und erweiterten Zustandsübergangsdiagrammen (Statecharts) /ML00/, SZ03/.

Ein sowohl im akademischen als auch im industriellen Umfeld weit verbreitetes Modellierungs- und Simulationswerkzeug ist Matlab / Simulink / Stateflow (kurz: ML/SL/S, siehe /ML,SL,SF/). Die Produktfamilie, die sich in der Automobilindustrie als de-facto Standard für die Modellierung durchgesetzt hat, unterstützt die Entwicklung und Analyse linearer wie nichtlinearer dynamischer Systeme. Während Matlab eine prozedurale Pro-

grammiersprache bereitstellt, stellen die Erweiterungen Simulink und Stateflow. Simulatoren für Blockschaltbilder bzw. Statecharts zur Verfügung. Graphische Editoren erlauben dabei eine intuitive Darstellung und Entwicklung komplexer Modelle. Hierarchisch strukturierte Modularität dient der Beherrschung der Komplexität /JCZ+00, ML00/.

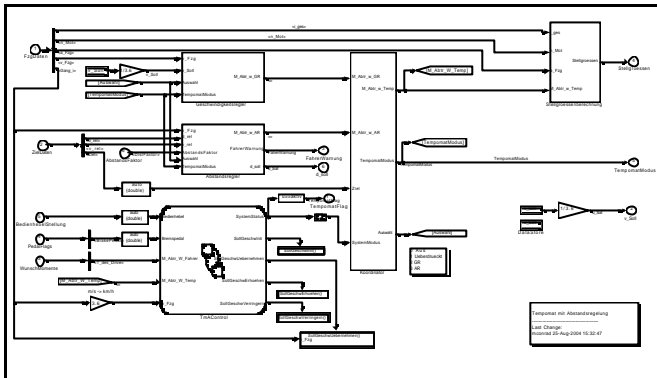


Abb. 1: Simulink/Stateflow -Modell (Beispiel)

Ein Simulink/Stateflow-Modell kann simuliert, d.h. ausgeführt werden. Bei der Simulation folgt die Berechnungskausalität den definierten Wirkungsrichtungen, bis das komplette Modell abgearbeitet ist /ML00/.

Für die Lösung der durch das Modell beschriebenen Gleichungen zu bestimmten Zeitpunkten stehen verschiedene vordefinierte Lösungsverfahren (Solver) zur Auswahl. Solver mit variabler Schrittweite (engl. variable-step solver) sind insbesondere für die genaue Modellierung physikalische Systeme von Bedeutung und werden hauptsächlich bei der Modellierung von Strecke und Umgebung eingesetzt. Für die Entwicklung der eingebetteten Software selbst kommen Solver mit konstanter Schrittweite (engl. fixed-step solver) zum Einsatz, die eine notwendige Voraussetzung für eine effiziente Codegenerierung darstellen.

Extensive Anwendung erfährt die Block-orientierte Modellbildung im Rahmen der Modell-basierten Entwicklung eingebetteter, automobiler Software (siehe /Bec00, Rau03, CFG+05/). Hier wird typischerweise frühzeitig im Entwicklungsprozess in der Beschreibungssprache des Modellierungs- und Simulationswerkzeuges sowohl ein ausführbares Modell der Steuerungs- und Regelungssoftware (Funktionsmodell) als auch ein Modell des umgebenden Systems (Streckenmodell) und dessen Umgebung (Umgebungsmodell erstellt und dieser Verbund simuliert. Während das Strecken- / Umgebungsmodell im weiteren Entwicklungsverlauf schrittweise durch das reale System und dessen reale Umgebung ersetzt wird, dient das Funktionsmodell als Basis für die Implementierung der eingebetteten Software auf dem Steuergerät durch Codegenerierung¹ (vgl. /ML00, JCZ+00, SZ03/).

Spezifisch für diese Art der Entwicklung ist, dass das Funktionsmodell sowohl die gewünschte Funktion spezifiziert als auch ein Design für die Umsetzung der Funktion vorgibt und schließlich auch Grundlage der Implementierung mittels Codegenerierung ist. Anders ausgedrückt besitzt ein solches Funktionsmodell sowohl Spezifikations-, als auch Design- und Implementierungsaspekte. In der Praxis wird diesen unterschiedlichen Anforderungen durch eine evolutionäre Weiterentwicklung der Funktionsmodelle von einem frühen logischen Modell hin zu einem Implementierungsmodell Rechnung getragen (Modellevolution. Im Vergleich zur traditionellen Softwareentwicklung mit einer klaren Phasentrennung ist bei der Modell-basierten Entwicklung ein stärkeres Zusammenwachsen der Phasen Spezifikation, Design und Implementierung zu verzeichnen (Abb. 2).

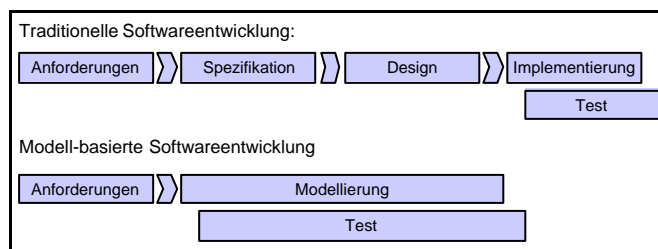


Abb. 2: Gegenüberstellung von traditioneller und Modell-basierter Softwareentwicklung

¹ Alternativ kann die Codierung natürlich auch weiterhin von Hand erfolgen.

Durch das frühzeitige Vorliegen ausführbarer Funktionsmodelle kann bereits in dieser Phase durch den Funktionsentwickler mit unterschiedlichen Techniken erprobt und geprüft werden. Auf diese Weise können auftretende Fehler früh erkannt und damit kostengünstig beseitigt werden. Das Spektrum der dabei anwendbaren Verfahren reicht von der interaktiven ad-hoc Simulation bis zum systematischen Test des Modells.

Die Verwendung des Modells als Testobjekt ermöglicht einen im Vergleich zur herkömmlichen Entwicklung deutlich frühzeitigeren Beginn der Testdurchführung (Abb. 2).

Wesentliche Vorteile der Modell-basierten Entwicklung liegen damit in der Durchgängigkeit der verwendeten Modellierungsmittel und Werkzeuge, im Effizienzgewinn durch die Codegenerierung sowie in der Tatsache, dass Test und Erprobung bereits auf Modellebene beginnen können. Auf die Spezifika des entwicklungsbegleitenden Testprozesses in der Modell-basierten Entwicklung wird im Folgeabschnitt detaillierter eingegangen.

Generell ist das Paradigma der Modell-basierten Entwicklung nicht an das Vorhandensein eines bestimmten Modelltyps gebunden. Neben blockorientierten Modellen können z.B. auch UML-Modelle Verwendung finden. Dieser Beitrag fokussiert jedoch auf die Ausprägung der Modell-basierten Entwicklung mit ML/SL/SF.

MODELL-BASIERTER TEST VON STEUERGERÄTESOFTWARE

Da die meisten Fahrzeugfunktionen vom reibungslosen Funktionieren der eingebetteten Software abhängig sind, muss diese zuverlässig und weitestgehend fehlerfrei sein. Dies kann nur erreicht werden, wenn die konstruktive, fehlervermeidende Qualitätssicherung während der Modell-basierten Entwicklung durch eine auf den Entwicklungsprozess abgestimmte Kombination analytischer, fehlererkennender Qualitätssicherungsmaßnahmen ergänzt wird.

Ein wesentliches Element der analytischen Qualitätssicherung bei eingebetteten Systemen im Kraftfahrzeug ist der Fahrversuch. Der Fahrversuch, der eigentlich zur Applikation der Steuer- und Regelparameter und der Validierung des Gesamtfahrzeugs dient, wird jedoch häufig auch missbräuchlich zum Auffinden von softwarebezogenen Spezifikations-, Design- und Implementierungsfehlern verwendet.

Fahrversuche sind mit beträchtlichen Aufwänden verbunden. Für den Test eines einzelnen Steuergerätes sind nicht selten Fahrleistungen von mehreren Millionen Kilometern erforderlich. Zudem liegen Fahrversuche häufig auf dem (zeitlich) kritischen Pfad der Steuergeräteentwicklung, da sie i.d.R. gegen Ende der Systementwicklung stattfinden. Zu diesem Zeitpunkt sind die durchschnittlichen Kosten für die Fehlerbeseitigung bereits sehr hoch. Aufgrund des hohen Zeit- und Kostendrucks muss daher häufig ein Kompromiss zwischen frühzeitiger Markteinführung und der erreichten Prüftiefe akzeptiert werden. Im Ergebnis werden Entwicklungs- und Implementierungsfehler eingebetteter Software zu spät im Entwicklungsprozess oder gar erst beim Kunden entdeckt /Sim97/.

Der Einsatz von Methoden und Verfahren zum systematischen Testen von Software während ihrer Entwicklung ermöglicht für bestimmte Fehlerklassen im Vergleich zum Fahrversuch eine frühzeitigere und effizientere Fehleraufdeckung. Voraussetzung hierfür ist jedoch eine an den praktischen Erfordernissen orientierte Adaption dieser Methoden und Verfahren an die Spezifika der Anwendungsdomäne. Um eine hohe Wirksamkeit sicherzustellen, dürfen die Tests nicht nur am Ende der Entwicklung durchgeführt werden, sondern müssen entwicklungsbegleitend stattfinden.

Die Modell-basierte Entwicklung eröffnet hier die Möglichkeit, insbesondere das ausführbare Modell als zusätzliche Informationsquelle für den Testprozess zu nutzen. Dies kann auf vielfältige Weise geschehen, z.B.:

- Modell als Testobjekt:** das frühzeitige Vorliegen des ausführbaren Modells erlaubt bereits dieses im Rahmen sogenannter Modelltests einer analytischen Qualitätssicherung zu unterziehen. Während textuelle Spezifikationen i.d.R. nur eine informelle Prüfung erlauben, ermöglicht der Test des Modells weitergehende Aussagen über die Qualität der Lösung bereits vor Beginn der eigentlichen Implementierung. Dadurch können bestimmte Prüfkategorien in frühere Entwicklungsphasen verlegt und damit Fehler frühzeitig gefunden und kostengünstig korrigiert werden.
- Modell als Testbasis:** die im Modell vorhandenen Informationen können als Basis für die Ermittlung der Testszenarien dienen.
- Modell als Testorakel:** das ausführbare Modell kann in bestimmten Fällen zur Ermittlung von Sollwerten, d.h. als Testorakel, verwendet werden. Dies erleichtert die Durchführung von Back-to-back-Tests zwischen den unterschiedlichen Repräsentationsformen des Testobjektes.

- ❑ Modell als Basis für Metriken: zur Steuerung der Testtiefe bzw. des Testendes können neben den strukturellen Überdeckungsmaßen auf Softwareebene (Codeüberdeckung, engl. code coverage) zusätzlich Überdeckungsmetriken auf Modellebene (Modellüberdeckung, engl. model coverage) zum Einsatz kommen.
- ❑ Wiederverwendung von Testszenarien: Die den Modelltests zugrundeliegenden Testszenarien können in späteren Entwicklungsphasen bspw. für den Softwaretest oder den Test des eingebetteten Systems wiederverwendet werden.

Unterschiedliche Testverfahren können dabei auf verschiedene Art und Weise vom Vorhandensein eines ausführbaren Modells profitieren. Daher sollte die Modell-basierte Entwicklung durch eine geeignete Kombination verschiedener Testmethoden begleitet werden, wofür sich der Begriff des Modell-basierten Tests eingebürgert hat.

Unter Modell-basiertem Testen (engl. model-based testing) soll im weiteren der entwicklungsbegleitende Testprozess im Rahmen der Modell-basierten Entwicklung, der eine Kombination unterschiedlicher, sich gut ergänzender Testverfahren umfasst und dabei das ausführbare Modell als reichhaltige Informationsquelle für den Test benutzt, verstanden werden.

Auch im Rahmen des Modell-basierten Tests kommt der Auswahl der Testszenarien unter den Testaktivitäten eine zentrale Rolle zu, da sie entscheidend für Umfang und Qualität des Tests ist. Deshalb müssen adäquate Testverfahren, anhand derer die einzelnen Entwicklungsstadien (Modell, Software, eingebettetes System) geprüft werden sollen, Kernpunkte eines entwicklungsbegleitenden, Modell-basierten Testprozesses sein /Con04/.

EFFEKTIVE TESTSTRATEGIE FÜR DEN MODELL-BASIERTEN TEST

Da ein singuläres Testverfahren i.d.R. nicht zu einer ausreichenden Testabdeckung führt, werden in der Praxis möglichst komplementäre Testverfahren im Rahmen einer Teststrategie geeignet miteinander kombiniert. Leistungsfähige Teststrategien umfassen dabei Kombinationen von Funktions- und Strukturtestverfahren In /Gri95/ wird hierfür der Begriff der effektiven Teststrategie geprägt.

Ziel einer effektiven Teststrategie ist es, eine geeignete Kombination verschiedener Testverfahren bereitzustellen, die eine hohe Fehleraufdeckungswahrscheinlichkeit gewährleistet. Eine effektive Teststrategie für den Modell-basierten Test muss dabei die Spezifika der Modell-basierten Entwicklung und in Sonderheit das Vorhandensein eines ausführbaren Modells angemessen berücksichtigen und damit in zwei Punkten über eine allgemeine hinausgehen. Sie sollte:

- ❑ Testverfahren, bei denen das ausführbare Modell der Software als Testbasis fungiert und
- ❑ die verschiedenen Arten (Ausprägungen) des Testobjektes, die typischerweise im Rahmen der Modell-basierten Entwicklung auftreten, berücksichtigen.

Den Schwerpunkt einer solchen Teststrategie bildet die systematische Ableitung von Testszenarien aus der funktionalen Spezifikation, den Schnittstellen und dem ausführbaren Modell der eingebetteten Software. Ergänzend wird ein geeignetes Strukturtestkriterium auf Modellebene definiert, anhand dessen die Güte der so ermittelten Tests bewertet werden kann. Falls erforderlich, sind solange weitere Testszenarien zu definieren, bis das gewählte Strukturtestkriterium erfüllt ist. Ist so eine ausreichende Testabdeckung auf Modellebene gewährleistet, können die Funktions- und Strukturtestszenarien im Rahmen von Back-to-back-Tests für den Test der Software und des eingebetteten Systems wiederverwendet werden, um die funktionale Äquivalenz zwischen dem ausführbaren Modell und den daraus abgeleiteten Repräsentationsformen nachzuweisen.

Im Einzelnen wird dabei wie folgt vorgegangen:

① Systematischer Funktionstest auf Modellebene: Zunächst werden frühzeitig im Entwicklungsprozess funktionsorientierte Testszenarien systematisch aus der funktionalen Spezifikation, den Schnittstellen und dem ausführbaren Modell abgeleitet.

Die Ermittlung der Testszenarien ist solange fortzusetzen werden, bis eine ausreichende Anforderungs- und Wertebereichsüberdeckung erreicht ist. Die ermittelten Testszenarien sind dann im Rahmen eines Modelltests auszuführen. Die Modellreaktion auf die Testszenarien ist aufzuzeichnen.

② Monitoring der Modellüberdeckung: Um möglichst frühzeitig zu Aussagen über die Abdeckung der Struktur des Testobjekts zu gelangen, bietet sich die Verwendung von strukturorientierten Überdeckungskriterien auf

Modellebene (model coverage) an, da deren Erfüllung schon gemessen werden kann, bevor die eigentliche Software vorliegt.

Die mit den in Schritt ① (und ggf. in Schritt ③) ermittelten Testszenarien erreichte Modellüberdeckung ist zu messen. Wird eine ausreichende Modellstrukturüberdeckung erreicht, kann zu Schritt ④ übergegangen werden. Andernfalls ist mit Schritt ③ fortzufahren.

③ Strukturtest auf Modellebene: Falls bisher keine ausreichende Modellüberdeckung erreicht wurde, sind die nicht überdeckten Modellteile zu identifizieren und gezielt Testszenarien zur Überdeckung dieser Modellteile zu erstellen und zu der bestehenden Testsuite hinzuzufügen. Danach ist Schritt ② erneut durchzuführen. Dieser Vorgang ist fortzusetzen, bis eine ausreichende Modellüberdeckung erzielt ist.

Die ermittelten Testszenarien sind dann im Rahmen eines Modelltests auszuführen. Die Modellreaktion auf die Testszenarien ist aufzuzeichnen.

④ Durchführung von Back-to-back-Tests mit der Software und dem eingebetteten System: Stehen im weiteren Entwicklungsverlauf die aus dem Modell abgeleitete Software bzw. das eingebettete System zur Verfügung, sind die (in den Schritten ① und ③ entstandenen) Testszenarien an der Software bzw. dem eingebetteten System zu wiederholen. Die Systemreaktion ist wiederum aufzuzeichnen und mit der Modellreaktion zu vergleichen.

Bei hinreichender Übereinstimmung (funktionaler Äquivalenz) kann davon ausgegangen werden, dass die Transformation des Modells in C-Code und dessen Einbettung ins Steuergerät fehlerfrei erfolgt ist.

Die sich aus den Schritten ① bis ④ ergebende effektive Teststrategie für den Modell-basierten Test ist in Abb. 3 schematisch dargestellt. Die Auswahl der Testverfahren in den Schritten ① und ③, des Strukturtestkriteriums in Schritt ② sowie der Vergleichsverfahren in Schritt ④ sind bei Bedarf projektspezifisch zu adaptieren. Eine solche Auswahl kann auf Basis der Übersicht im Folgekapitel erfolgen.

Die durch die effektive Teststrategie für den Modell-basierten Test definierte Kombination aus funktionalen und strukturellen Testverfahren auf Modellebene mit anschließender Wiederverwendung der so ermittelten Testszenarien für den Test der Software und des eingebetteten Systems bringt einen der Hauptvorteile des Modell-basierten Ansatzes - bereits das ausführbare Modell der späteren Software systematisch testen zu können - deutlich zum Tragen. Testaktivitäten können so in frühere Entwicklungsphasen verlagert und damit effizient durchgeführt werden.

Die systematische Ermittlung von Testszenarien unter funktionalen wie unter strukturellen Gesichtspunkten ermöglicht eine effektive Aufdeckung softwarebezogener Fehlertypen. Eine ausreichende Testabdeckung wird im Rahmen der effektiven Teststrategie durch eine Kombination verschiedener (z.B. anforderungs-, datenbereichs- und (modell-)strukturorientierter) Überdeckungskriterien gewährleistet.

Anhang A gibt einen Überblick über verschiedene funktionale und strukturelle Testverfahren für den Modell-basierten Test sowie unterstützende Werkzeuge.

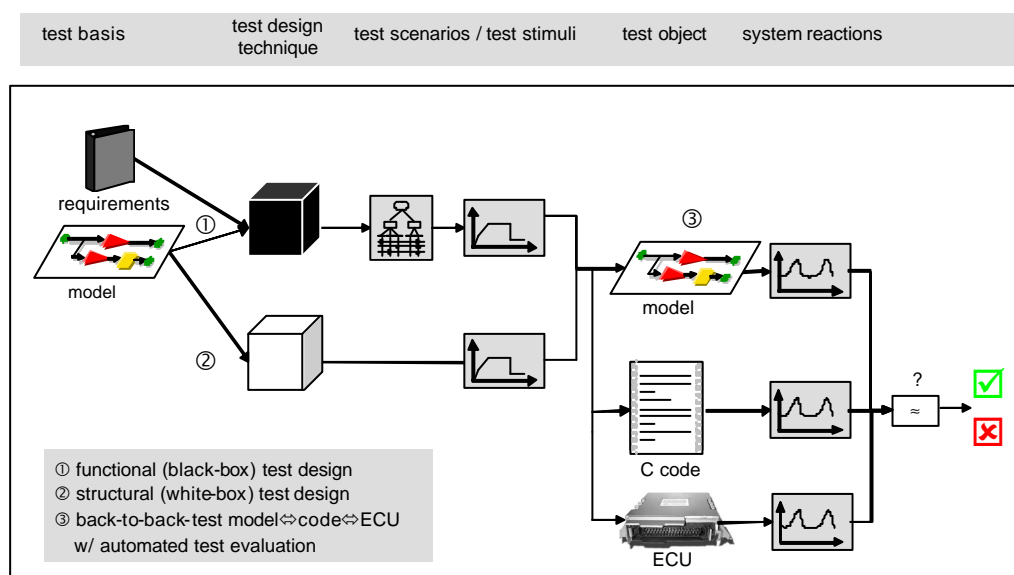


Abb. 3: Effektive Teststrategie für den Modell-basierten Test (Überblick)

ZUSAMMENFASSUNG UND AUSBLICK

Zur Unterstützung der Modell-basierten Entwicklung im Testbereich wurden unterschiedliche Modell-basierte Testverfahren und –ansätze entwickelt.

Im betrachteten Anwendungsgebiet, der Entwicklung eingebetteter Fahrzeugsoftware, kommen dabei auch spezifische Verfahren und Ansätze zum Einsatz, die von den allgemeinen, in anderen Einsatzgebieten verwendeten Vorgehensweisen, abweichen. Typisch für die verwendeten Ansätze ist die starke Nutzung der ohnehin vorhanden Simulink / Stateflow-Funktionsmodelle für Testzwecke.

Um einen umfassenden Test zu gewährleisten, sind verschiedene Modell-basierte Testverfahren miteinander zu kombinieren. Die Gesamtheit der derart ermittelten Tests dient der Validierung einer oder mehrerer Evolutionsstufen des Funktionsmodells. Später können die nachfolgenden Modellstufen sowie der generierte Code mit Hilfe von Back-to-back Tests unter Wiederverwendung der vorhandenen Tests abgesichert werden. Dient dabei das bereits validierte Originalmodell als ‚goldene Referenz‘, kann die Testauswertung beim Test der nachfolgenden Modellstufen im Wesentlichen auf Signalvergleiche zwischen aktuellen und Referenzoutputs reduziert werden.

Modell-basiert entwickelte, eingebettete Fahrzeugsoftware, kann mit Hilfe der beschriebenen Teststrategie und den unterstützenden Werkzeugen systematisch und effizient getestet werden.

ANHANG: VERFAHREN UND WERKZEUGE FÜR DEN TEST VON SIMULINK/STATEFLOW-MODELLEN

Der Anhang listet verschiedene Verfahren und Werkzeuge für den Test von Simulink/Stateflow-Modellen auf. Aufgrund der dynamischen Entwicklung in diesem Bereich und der Vielzahl von proprietären, teilweise nicht publizierten Lösungen kann kein Anspruch auf Vollständigkeit erhoben werden.

Testverfahren für den Funktionstest von Modellen

Evolutionärer Sicherheitstest (EST)

Der evolutionäre Sicherheitstest (Evolutionary Safety Testing, EST) /PCG05/ ist ein automatischer Testansatz für den Sicherheitstest von Funktionsmodellen. Hierbei werden erweiterte evolutionäre Algorithmen dazu verwendet, im Rahmen eines Optimierungsprozesses nach Testdatensequenzen zu suchen, bei deren Ausführung gegebene Sicherheitsanforderungen verletzt werden.

Evolutionäre Algorithmen können für eine Vielzahl verschiedener Suchprobleme eingesetzt werden. Voraussetzungen für ihre Anwendung sind die Definition des Suchraumes in dem die Lösungssuche erfolgt und einer Bewertungsfunktion, mit der die Eignung (fitness) der ermittelten Lösungsvorschläge bewertet werden kann. Die Interpretation der Suche nach Testszenarien, die eine bestimmte Sicherheitsanforderung verletzen, als Optimierungsproblem und die Verwendung evolutionärer Algorithmen zu dessen Lösung bildet die Grundidee für den evolutionären Sicherheitstest. Ähnlich wie in natürlichen Evolutionsprozessen werden Lösungen dabei iterativ verbessert und Optima durch Variation und Selektion alternativer Lösungsmöglichkeiten gefunden bzw. approximiert.

Die im jeweiligen Iterationsschritt durch die evolutionären Algorithmen vorgeschlagenen Testszenarien werden zur Stimulation des funktionalen Modells verwendet. Die Eignung eines Testszenarios, die gegebene Sicherheitsanforderung zu verletzen, wird dabei mit Hilfe der Bewertungsfunktion bestimmt. Die Fitnesswerte einzelner Testszenarios des aktuellen Schrittes werden dann zur Ermittlung besserer Testszenarien im Folgeschritt herangezogen.

I/O-Orientiertes Test Design

Das im Rahmen von STEP-X vorgeschlagene I/O-Orientierte Testdesign /HSM+04/ ist ein funktionales Testverfahren, das auf einer vereinfachten Variante der weiter oben beschriebenen Klassifikationsbaum-Methode für eingebettete Systeme CTM_{EMB} basiert.

Klassifikationsbaum-Methode für Eingebettete Systeme CTM_{EMB}

Die Klassifikationsbaum-Methode für eingebettete Systeme (Classification-tree Method for Embded Systems, CTM_{EMB})² /CDF+99, Con04, Con04a, KM05/, eine Erweiterung der Klassifikationsbaum-Methode /GG93/, ist ein funktionales Testverfahren, das die systematische Ermittlung zeitveränderlicher Testszenarien (Testsequenzen) für ein ausführbares Modell erlaubt.

Der Eingabedatenraum des Testobjektes (Funktionsmodell oder Subsystem), der als durch die n Eingangssignale des Testobjektes aufgespannter (n-dimensionaler) Hyperwürfel aufgefaßt werden kann, wird dabei disjunkt und vollständig in Äquivalenzklassen unterteilt. Diese Äquivalenzklassen bzgl. des Eingabedatenraumes abstrahieren in für den Test geeigneter Weise von einzelnen, konkreten Eingabewerten. Bei der Partitionierung des Eingabedatenraumes, die graphisch in Form eines (Klassifikations-)Baumes veranschaulicht wird, wird eine Unterteilung dergestalt angestrebt, dass sich die einzelnen Äquivalenzklassen uniform bezüglich der Aufdeckung von potentiellen Fehlern verhalten (Uniformitätshypothese).

Die Definition der Testsequenzen auf Basis der Partitionierung des Eingabedatenraumes ist eine kombinatorische Aufgabe. Dazu werden einzelne Äquivalenzklassen ausgewählt und zu einer zeitlichen Folge aneinandergereiht. Die Auswahl der Klassen und deren zeitliche Abfolge wird durch eine kompakte graphische Notation unterstützt, die die schrittweise Beschreibung abstrakter Signalverläufe für jedes Eingangssignal erlaubt.

Modell-basierter Black-box Test (MB³T)

Der Modell-basierte Black-box Test (Model-based Black-box Test, MB³T) /CFS04, CFS05/ ist ein systematisches funktionales Testverfahren für die Ableitung von Testszenarien aus zwei komplementären Blickwinkeln. Dabei werden zunächst aus der Anforderungsspezifikation logische, nicht ausführbare Tests abgeleitet,

² früher auch als CTM/ES bekannt

die später durch eher technische, ausführbare Tests, die aus dem Modell abgeleitet werden, ergänzt werden. Zusätzlich kann im Rahmen der MB³T die Konsistenz zwischen beiden Mengen von Tests überprüft werden. Diese Checks gewährleisten die notwendige Testgüte und –vollständigkeit.

Die spezifikationsbasierten, logischen Tests können bereits frühzeitig im Entwicklungsprozess mit Hilfe der Klassifikationsbaum-Methode (Classification-tree Method, CTM) /GG93, Gri95/ ermittelt werden. Bei der CTM wird zunächst die Spezifikation des Testobjektes auf verschiedene testrelevante Aspekte hin untersucht. Für jeden Testaspekt wird, ggf. iterativ, eine disjunkte und vollständige Partitionierung erstellt auf deren Basis die Festlegung der Tests erfolgt. Die so erstellten spezifikationsbasierten Tests sind abstrakt, nicht ausführbar und enthalten keinerlei Design- oder Implementierungsdetails.

Wenn im weiteren Entwicklungsverlauf ein ausführbares Funktionsmodell zur Verfügung steht, werden mit Hilfe der CTM_{EMB} aus diesem ausführbare, zur Schnittstelle des Testobjekts passende Testszenarien erzeugt. Beide Gruppen von Tests werden durch Klassifikationsbäume und zugehörige Kombinationstabellen visualisiert.

Eine auf den graphischen Notationen der Klassifikationsbaum-Methode aufbauende Vorgehensweise zur Analyse der Beziehungen zwischen nicht ausführbaren und ausführbaren Tests gewährleistet eine klare Abbildung zwischen den zu unterschiedlichen Zeitpunkten aus verschiedenen Blickwinkeln erstellten Tests und stellt damit deren Konsistenz sicher.

Prototyp-basierter Test für hybride reaktive Systeme

Prototyp-basierter Test für hybride reaktive Systeme /HPP+03/ ist ein strukturierter Ansatz zur Erzeugung von Testszenarien für gemischt diskret-kontinuierliche Systeme. Ein rein diskretes, speziell für Testzwecke konstruiertes Automatenmodell beschreibt die Menge der möglichen Systemläufe in abstrakter Weise und dient als Informationsquelle für die Ableitung der Testsszenarien. Diese werden, nach einer entsprechenden Konkretisierung zur Stimulation des eigentlichen gemischt-kontinuierlichen Funktionsmodells benutzt.

Zunächst wird ein rein diskretes Automatenmodell derjenigen Teile des Systemverhaltens, die für den Test relevant sind, erstellt. Dieses Systemverhaltensmodell stellt eine diskrete Abstraktion des zu testenden gemischt-kontinuierlichen Funktionsmodells in Bezug auf typische Benutzungsszenarien oder Betriebsmodi dar. Die Traversierung dieses Modells auf Basis bestimmter Überdeckungskriterien führt zu (zunächst abstrakten) Testszenarien. Die resultierende Testsuite kann z.B. alle (Paare von) Transitionen des Testmodells überdecken oder andere Überdeckungskriterien erfüllen.

Mittels einer zur ursprünglichen Abstraktion dualen Konkretisierungsrelation können die abstrakten Testszenarien an das zu testende Funktionsmodell adaptiert werden. Die Konkretisierung kann bspw. die Dauer einzelner Zustände bzw. Übergänge oder die Veränderungen der Eingangssignale in den einzelnen Zeitintervallen betreffen. Ergebnis der Adaption sind Testdatenfolgen mit denen das Funktionsmodell zu Testzwecken stimuliert werden kann.

Testgenerierung durch Modelchecking

Modelchecking-Verfahren, die in jüngster Zeit erfolgreich auf Simulink/Stateflow-Modelle angewendet wurden /ESJ+05/, können auch zur Erzeugung von Testfällen für den Modell-basierten Test benutzt werden.

Dies kann auf verschiedene Arten geschehen, bspw. in Form von Anforderungstests (Gegenbeispiele für Anforderungen, die nicht bewiesen werden können, dienen als Testfälle), Strukturtests (der Modelchecker wird zur Erzeugung von Tests, die alle Zustände, Transitionen, Lookup-Table-Einträge etc. überdecken) oder Unit Checking /GP03/. Dieser Bereich stellt ein aktuelles Forschungsgebiet dar.

Die verfügbaren Modelchecking-Techniken sind derzeit auf die diskreten Stateflow-Anteile realer Anwendungen beschränkt.

Time Partition Testing (TPT)

Time Partition Testing (TPT) /Leh00, Leh04/ ist ein auf der Klassifikationsbaum-Methode /GG93/ basierender Ansatz für den funktionalen Test des kontinuierlichen Verhaltens eingebetteter Systeme.

Dabei wird ein explizites Testmodell erstellt, das die graphische Definition von Testfällen für kontinuierliches, reaktives Verhalten ermöglicht. Tests werden in abstrakter Form durch hierarchische, parallele Automaten modelliert, die den generellen Ablauf eines Testfalls widerspiegeln. Diese Beschreibung kann mit TPT bis auf die Ebene konkreter Daten verfeinert werden. Testdatenfolgen werden dabei in aufeinanderfolgende Phasen unterteilt, deren Abfolge und Übergänge durch die Zustandsautomaten beschrieben werden (Time Partitioning).

Eine Menge von Testfällen kann dabei gemeinsam modelliert und zu einem generelleren Ablaufschema kombiniert werden. Mögliche alternative Signalverläufe eines Zustandes (Varianten) werden dabei zusammengefasst. Die Definition von Testfällen wird durch die Einführung von Varianten zu einer kombinatorischen Aufgabe, die mit Hilfe der Klassifikationsbaum-Methode gelöst werden kann.

TPT kann als Plug-in zur Testbeschreibung innerhalb des Testwerkzeugs MTest verwendet werden. Auf diese Weise können die Tests auf ein in Simulink/Stateflow erstelltes Funktionsmodell angewendet werden.

Zufallstests

Auch im Bereich des Modell-basierten Tests können Zufallstechniken zur Erzeugung von Testdaten eingesetzt werden.

Generatoren für Zufallstests sind in einigen Testwerkzeugen enthalten, können aber auch durch einfachste Mittel wie Excel Spreadsheets ersetzt werden.

Die strukturelle Überdeckungsrate, die mit Zufallstests auf Modellebene erreicht werden kann, ist insbesondere dann nicht ausreichend, wenn Stateflow Charts oder andere Systemteile mit inneren Zuständen zu überdecken sind.

Testverfahren für den Strukturtest von Modellen

Constraint-basierte Testdatenanalyse

Die Constraint-basierte Testdatenanalyse für eingebettete Steuerungssoftware /LG03, Lin04, Lin05/ ermöglicht die systematische Testdatenermittlung ausgehend von diskret-kontinuierlichen Funktionsmodellen.

Zur Steuerung der Testdatenermittlung wird ein über Mutationen festgelegtes und mittels Constraints beschriebenes Testkriterium verwendet. Die grundlegende Idee des Verfahrens besteht darin, mittels verschiedener Klassen von modellbasierten Mutationsoperatoren zunächst Mutanten des Originalmodells (virtuell) zu erzeugen und dann Testdaten zu generieren, die die Mutanten vom Originalmodell unterscheiden können.

Das Potential des Verfahrens liegt in der Tatsache, dass die Aufdeckung eines Fehlers im Originalprogramm vergleichbare Testdaten erfordert, wie das Erkennen bestimmter Mutanten. Testdaten, welche sämtliche Mutanten erkennen lassen, dabei jedoch mögliche Fehler im Originalprogramm nicht aufdecken, sind unwahrscheinlich.

Das Verfahren befindet sich im Entwicklungsstadium, Werkzeugunterstützung für die Datenerzeugung ist derzeit noch nicht vorhanden.

Modell-basierte Testfallextraktion

Die Modell-basierte Testfallextraktion /HSP05/ ist ein Strukturtestverfahren mit dessen Hilfe Modellstrukturtests auf nachvollziehbare Art und Weise erzeugt werden.

Auf Basis der Struktur des Funktionsmodells wird eine sogenannte 'Werteausleuchtung' vorgenommen. Der Ansatz unterscheidet sich dahingehend von dem nachfolgenden, dass keine heuristischen Suchtechniken, sondern deterministische Verfahren eingesetzt werden, d.h. eine Wiederholung der Testdatenerzeugung führt stets zu den gleichen Tests. Einzelne Testdaten können darüber hinaus einem bestimmten Modellstrukturelement zugeordnet werden.

Werkzeugunterstützung für den Ansatz ist in Form des Model-based Test Case Extractors MOTCase-X vorhanden. Das Werkzeug unterstützt derzeit aber noch keine Stateflow-Anteile und nur eine Teilmenge der vorhandenen Simulink-Blöcke.

Modellstrukturtests

Aus Simulink/Stateflow-Modellen können analog zu prozeduralen Programmiersprachen Kontrollflußgraphen abgeleitet werden. Auf Basis solcher aus dem Modell abgeleiteten Kontrollflußgraphen lassen sich strukturelle Überdeckungskriterien auf Modellebene definieren /Ald02, BCS+03/. Diese Modellüberdeckungskriterien lassen sich als Testselektionskriterien für die automatische Ableitung von Strukturtests für Modelle einsetzen. Um eine hohe Strukturüberdeckung der Funktionsmodelle zu erreichen, sind in der Regel Folgen von Testdaten zu erzeugen.

Verschiedene Tools realisieren die Idee solcher Testsequenzgeneratoren. Hierzu gehören beispielsweise Reactis Tester /Reactis/, Beacon Tester /Beacon/ und das Evolutionäre Testtool ET /WSB01/.

Werkzeuge für den Test von Simulink/Stateflow-Modellen

CTE/ES, CTE/XL

Die Klassifikationsbaum-Editoren CTE/ES (Classification-tree Editor for Embedded Systems) /CTE/ und CTE/XL (Classification-tree Editor with Extended Logics) /LW00/ bieten integrierte Werkzeugunterstützung für die Klassifikationsbaummethode. CTE/ES und CTE/XL (mit kleineren Abstrichen) können zur Unterstützung der Testfallermittlung im Rahmen CTM_{EMB} eingesetzt werden, der auch CTE/XL zur Darstellung der Kombinatorik im Rahmen von TPT. CTE/ES ist als Plug-In zur Testbeschreibung in das Werkzeug MTest integriert, der CTE/XL in TPT.

ET

Das Evolutionäre Test Tool ET /WSB01/ ist ein proprietärer Testdatengenerator für verschiedene Plattformen, die auch für den Test von Simulink/Stateflow-Modellen eingesetzt werden kann.

Abhängig von der Parametrierung der evolutionären Algorithmen können mit Hilfe heuristischer Suchstrategien Testdaten, die unterschiedlichen Kriterien genügen, erzeugt werden. Anwendungen des Werkzeugs umfassen den u.a. den evolutionären Strukturtest von Modellen sowie den evolutionären Sicherheitstest.

Embedded Validator

Der Embedded Validator EV /EV, ESJ+05/ ist eine Werkzeugumgebung für die formale Verifikation von reaktiven Modellteilen mittels Modelchecking.

Die Funktionsmodelle müssen dabei zunächst nach TargetLink übertragen werden. Der Modelchecker kann dann als Testdatengenerator für Modelle verwendet werden. Derzeit werden nur Stateflow-Anteile in ausreichendem Maße unterstützt.

MATT

Das Matlab Automated Testing Tool MATT /MATT, Tur01/ ist eine einfache Testumgebung für Simulink und den daraus mittels Real-Time Workshop generierten C-Code.

Testdatenfolgen für MIL und SIL Tests können dabei semi-automatisch vorgegeben werden. Stateflow wird derzeit nicht unterstützt. Als Add-ons für MATT können das Real-time Analysis Testing Tool (RATT) und das Graphical Input Specification Tool (GIST) verwendet werden.

MEval

Das Werkzeug MEval/MEval, CSW05/ automatisiert die Testauswertung bei Regressions- und Back-to-back-Tests im Rahmen der modellbasierten Entwicklung durch einen automatisierten robusten Vergleich der Testoutputs, die als zeitbehaftete Signalverläufe vorliegen, mit den zugehörigen Referenzsignalverläufen.

Die zu vergleichenden Signale können aus anderen Testwerkzeugen, wie bspw. MTest, erzeugt oder in Form von .mat Files importiert werden. Ein robuster Signalvergleich, der Abweichungen sowohl im Zeit- als auch im Wertebereich toleriert, wird über verschiedene, im Rahmen eines Baukastenkonzeptes miteinander kombinierbare Algorithmen erreicht. Eine zentrale Rolle spielt dabei das Differenzmatrix-Verfahren, mit dessen Hilfe lokale und globale zeitliche Abweichungen detektiert und analysiert werden können. Die Vergleichsergebnisse werden dabei sowohl in graphischer als auch in textueller Form aufbereitet.

MOTCase-X

Der Model-based Test Case Extractor MOTCase-X /HSP05/ ist ein proprietärer, automatischer Testdatengenerator für Simulink-Modelle. Basis für die Testdatenerzeugung sind verschiedene durch das Tool definierte Überdeckungskriterien auf Modellebene.

Die Testdatenerzeugung ist deterministisch, die generierten Tests können in einem .xml-Format exportiert werden. Stateflow-Modelle und bestimmte Simulink-Blöcke werden

MTest

Die Modell-basierte Testumgebung MTest /MTest, LBE+04, LB05/ stellt eine Plattform für das systematische und automatisierte Testen bereit. Die Testentwicklung kann dabei auf Basis der integrierten Klassifikationsbaummethode, durch direkte Testdatenvorgabe oder durch Integration zusätzlicher Plug-Ins zur Testbeschreibung wie TPT erfolgen. Darüber hinaus ist es möglich, Testvektoren aus anderen Quellen (z.B. Meßdaten) einfach zu importieren und auf das zu testende Modell anzuwenden. Dabei unterstützt MTest das durchgängige Testen auf Model-in-the-Loop-, Software-in-the-Loop- und Processor-in-the-Loop-Ebene

Diese eher Black-Box-orientierten Funktionalitäten werden durch die Anbindung der Simulink Model Coverage- und der TargetLink Code Coverage-Funktionalität zur Messung von Strukturüberdeckungen ergänzt. Eine weitergehende Unterstützung von White-Box-Tests ist durch Anbindung an automatische Testvektorgeneratoren ist auf dieser Basis möglich.

Reactis Tester

Reactis Tester /Reactis/ ist ein automatischer Testdatengenerator für Simulink / Stateflow Modelle. Basis für die Testdatenerzeugung sind verschiedene durch das Tool definierte Überdeckungskriterien auf Modellebene.

Die Testdatenerzeugung erfolgt nach heuristischen Suchstrategien, die generierten Tests können in einem Simulink / Stateflow kompatiblen Format exportiert werden.

Safety Test Builder

Der Safety Test Builder (STB) /STB/ ist ein automatischer Testdatengenerator für Simulink / Stateflow Modelle. Basis für die Testdatenerzeugung sind verschiedene durch das Tool definierte Überdeckungskriterien auf Modellebene.

Die Tests können sowohl auf MIL als auch auf SIL Ebene ausgeführt werden. STB unterstützt darüberhinaus Back-to-back Tests zwischen Modell und Code.

Simulink Verification and Validation Toolbox

Die Simulink Verification and Validation Toolbox /Simulink/ umfasst verschiedene Komponenten, die unterschiedliche Phasen des Modell-basierten Testprozesses abdecken.

Der Signal Builder Block erlaubt die direkte Testdatendefinition abschnittsweise definierter Signale auf graphischem Wege. Das Simulink Model Coverage Tool erlaubt die Messung der mit einer vorhandenen Testsuite erreichten Modelüberdeckung hinsichtlich struktureller und anderer Kriterien und stellt einfache Metriken zur Beurteilung der Modellkomplexität bereit. Das Requirement Management Interface ermöglicht die Verknüpfung von Modellelementen und Testfällen mit textuellen Anforderungen in DOORS. Eine Bibliothek vordefinierter Verifikationsblöcke ermöglicht die Überprüfung der Einhaltung von Eigenschaften während aller durchgeführten Testläufe.

Die einzelnen Komponenten können in eine vorhandene in-house Modeltestumgebung integriert oder mit anderen Werkzeugen für den Modell-basierten Test kombiniert werden.

TPT

Time Partition Testing TPT /Leh04/ ist eine proprietäre Test Design Umgebung zur Unterstützung der gleichnamigen Testbeschreibungstechnik. Neben der Testbeschreibung mit Hilfe des Time Partitioning ist eine direct test data definition möglich. Die Testausführung erfolgt über angekoppelte back end tools. So ist z.B. eine Testausführung für MIL, SIL und PIL Tests über die Integration von TPT in MTest möglich. TPT enthält eine Komponente zur (Python) skript-basierten Testauswertung.

REFERENCES

- /Ald02/ Aldrich, W.J.: Using Model Coverage Analysis to Improve the Controls Development Process. Proc. AIAA Modeling and Simulation Technologies Conference and Exhibition, Monterey (US), 2002.
- /ASCET-SD/ ASCET-SD, ETAS GmbH, de.etasgroup.com/products/ascet
- /Beacon/ Beacon Tester: Applied Dynamics International Inc., www.adi.com/products_be_bss_te.htm, 2003.
- /Bec00/ P. Bechberger: Modellbasierte Softwareentwicklung für Steuergeräte. Automobiltechnische Zeitschrift / Motortechnische Zeitschrift, Sonderausgabe Automotive Electronics, Jan. 2000, S.16-24
- /BCS+03/ Baresel, A., Conrad, M., Sadeghipour, S., Wegener, J.: The Interplay between Model Coverage and Code Coverage. 11. Europ. Int. Conf. on Software Testing, Analysis and Review (EuroSTAR 03), Amsterdam (NL), Dec. 2003.
- /CDF+99/ M. Conrad, H. Dörr, I. Fey, A. Yap: Model-based Generation and Structured Representation of Test Scenarios. Workshop on Software-Embedded Systems Testing (WSEST), Gaithersburg (US), Nov. 1999

- /CFG+05/ M. Conrad, I. Fey, M. Grochtmann, T. Klein: Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler. Informatik Forsch. Entw. Vol. 19 (2005), Sonderheft Modellierung
- /CFS04/ Conrad, M., Fey, I., Sadeghipour, S.: Systematic Model-Based Testing of Embedded Control Software: The MB³T Approach. Proc. ICSE 2004 Workshop W14S on Software Engineering for Automotive Systems (SEAS'04), Edinburgh (UK) May, 2004, pp. 17-25
- /CFS05/ Conrad, M., Fey, I., Sadeghipour, S.: Systematic Model-Based Testing of Embedded Automotive Software. Electronic Notes in Theoretical Computer Science 111 (2005) 13–26
- /Con04/ Conrad, M.: Modell-basierter Test eingebetteter Software im Automobil: Auswahl und Beschreibung von Testszenarien. PhD Thesis, Deutscher Universitätsverlag, Wiesbaden (D), 2004
- /Con04a/ Conrad, M.: A Systematic Approach to Testing Automotive Control Software. Proc. 30. Int. Congress on Transportation Electronics (Convergence '04), Detroit (US), Oct., 2004, pp. 297-308 (SAE Techn. Paper #2004-21-0039).
- /Con04b/ Conrad, M.: Systematic Testing of Embedded Automotive Software - The Classification-Tree Method for Embedded Systems. Proc. Dagstuhl Seminar N^o 04371 'Perspectives of Model-based Testing', Schloß Dagstuhl (D), Sept. 2004
- /CSW05/ Conrad, M., Sadeghipour, S., Wiesbrock, H.-W.: Automatic Evaluation of ECU Software Tests, SAE World Congress 2005, Detroit (US), April 2005 (SAE Techn. Paper #2005-01-1659)
- /CTE/ CTE for Embedded Systems, Razorcat Development GmbH, www.razorcat.com
- /ESJ+05/ Eisler, S., Scheidler, C., Josko, B., Sandmann, G., Stroop, J.: Preliminary Results of a Case Study: Model Checking for Advanced Automotive Applications. Fomal Methods Europe (FME'05), 2005
- /ETAS/ ETAS GmbH: www.etas.de
- /EV/ Embedded Validator. product information, www.osc-es.de/products/en/embeddedvalidator.php
- /EW01/ El-Far, I.K., Whittaker, J.A.: Model-based Software Testing. In J.J. Marciniak (Ed.): Encyclopedia of Software Engineering, 2nd edition, Wiley 2001.
- /GG93/ M. Grochtmann, K. Grimm: Classification Trees for Partition Testing. Software Testing, Verification and Reliability, 3 (1993), S. 63-82
- /GP03/ Gunter, E. L., Peled, D.: Unit Checking: Symbolic Model Checking for a Unit of Code. Verification: Theory and Practice 2003: 548-567
- /Gri95/ K. Grimm: Systematisches Testen von Software - Eine neue Methode und eine effektive Teststrategie. Dissertation, GMD-Bericht Nr. 251, Oldenbourg, 1995
- /HPP+03/ Hahn, G., Philipps, J., Pretschner, A., Stauner, T.: Prototype-based Tests for Hybrid Reactive Systems. Proc. 14th IEEE Intl. Workshop on Rapid System Prototyping, pp. 78-85, San Diego, June 2003
- /HSP05/ Hermes T., Schultze, A., Predelli O.: Software Quality is not a Coincidence: A Model-Based Test Case Generator. SAE World Congress 2005, Detroit, Michigan, US, Apr. 2005 (SAE technical paper #2005-01-1664)
- /HSM+04/ Horstmann, M., Schnieder, E., Mäder, P., Nienaber, S., Schulz, H.-M.: A framework for interlacing Test and/with Design. ICSE 2004 Workshop on Software Engineering for Automotive Systems, Edinburgh, UK, 2004
- /JCZ+00/ M. Jersak, Y. Cai, D. Ziegenbein, R. Ernst: A Transformational Approach to Constraint Relaxation of a Time-driven Simulation Model. Proc. of 13. Int. Symposium on System Sythesis, Madrid (ES), Sept. 2000
- /KCF+04/ Klein, T., Conrad, M., Fey, I., Grochtmann, M.: Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler. Proc. Modellierung 2004, Lecture Notes in Informatics (LNI), Vol. P-45, pp. 31-41
- /KM05/ Krupp, A., Mueller, W.: Die Klassifikationsbaummethode für eingebettete Systeme mit Testmustern für nichtkontinuierliche Reglerelemente. Proc. GI Jahrestagung 2005, Workshop Automotive Software Engineering (GI ASE'05), Bonn, Germany, Sept. 2005 (to appear)
- /LBE+04/ Lamberg, K., Beine, M., Eschmann, M., Otterbach, R., Conrad, M., Fey, I.: Model-based Testing of Embedded Automotive Software using MTest. SAE World Congress 2004, March 2004, Detroit, USA (SAE technical paper #2004-01-1593).
- /LB05/ Lamberg, K., Beine, M.: Testmethoden und -tools in der modellbasierten Funktionsentwicklung. Proc. Workshop 'Simulations - und Testmethoden für Software in Fahrzeugsystemen' (Annual meeting ASIM/GI working group 4.5.5 'Simulation of Technical Systems'), Berlin, Germany, March 2005
- /Leh00/ Lehmann, E.: Time Partition Testing: A Method for Testing Dynamic Functional Behaviour. Proceedings of TEST2000, London, Great Britain, May 2000.
- /Leh04/ Lehmann, E.: Time Partition Testing: Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen. PhD Thesis, TU Berlin, Germany, 2004
- /LG03/ Linder, P., Göhner, P.: Modellbasierte Testdatenermittlung mit Constraints. 17. Symposium Simulationstechnik (ASIM), Magdeburg, Germany, Sept, 2003
- /Lin04/ Linder, P.: Modellbasiertes Testen von eingebetteter Software: Ein Ansatz auf der Grundlage von Signalfussplänen. Proc. Automotive - Safety & Security 2004, Stuttgart, Germany, Oct. 2004

- /Lin05/ Linder, P.: Constraintbasierte Testdatenanalyse für eingebettete Steuerungssoftware. Proc. ASIM-STS 05, pp. 40-49. Proc. Workshop 'Simulations- und Testmethoden für Software in Fahrzeugsystemen' (Annual meeting ASIM/GI working group 4.5.5 'Simulation of Technical Systems'), Berlin, Germany, March 2005
- /LW00/ Lehmann, E.; Wegener, J.: Test Case Design by Means of the CTE XL. Proc. 8. Europ. Int. Conf. on Software Testing, Analysis & Review (EuroSTAR 2000), Copenhagen, Denmark, Dec. 2000
- /MATT/ MATT - Matlab Automated Testing Tool, University of Montana, www.cs.umt.edu/RTSL/matt
- /MEval/ MEval, ITPower Consultants, www.itpower.de/meval.html
- /ML00/ R. Merz, L. Litz: Objektorientierte Mathematische Modellierung - generische Methoden bei komplexen dynamischen Systemen. Informatik Spektrum, Band 23, Heft 2, Apr. 2000, S. 90-99
- /ML,SL,SF/ MATLAB, Simulink, Stateflow, The MathWorks, www.mathworks.com/products/matlab, www.mathworks.com/products/simulink, www.mathworks.com/products/stateflow
- /MTest/ MTest, dSPACE, www.dspace.de/ww/en/pub/products/sw/expsoft/mtest.htm
- /NI/ National Instruments Corp.: www.ni.com
- /PCG05/ H.Pohlheim, M. Conrad, A. Griep: Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences
SAE World Congress 2005, Detroit (US), April 2005, SAE Paper SAE2005-01-0750
- /PLP01/ Pretschner, A., Lötzbeyer, H., Philipps, J.: Model Based Testing in Evolutionary Software Development. Proc. 11. IEEE Intl. Workshop on Rapid System Prototyping, pp. 155-160, 2001
- /PP04/ Prenninger, W., Pretschner, A.: Abstractions for Model-Based Testing. Proc. TACoS'04, Barcelona, March 2004
- /Ran02/ Ranville, S.: Practical Application of Model-Based Software Design for Automotive, SAE Conference, Detroit, MI, May 2002,
- /Ran03/ Ranville, S.: Matlab "Add-On" Tools for State-Of-The-Art Embedded Software Development, 2003, Proc. 22nd AIAA/IEEE/SAE Digital Avionics System Conference, Indianapolis, US, Oct. 2003
- /Ran04/ Ranville, S.: Case Study of Commercially Available Tools that Apply Formal Methods to a Matlab/Simulink/Stateflow Model. SAE World Congress 2004. Detroit, MI, US, March 2004 (SAE technical paper #2004-01-1765) 2004
- /Ran06/ Ranville, S.: Practical Advice for Using Test Generation Tools. To appear, 2006
- /Rau03/ A. Rau: Model-Based Development of Embedded Automotive Control Systems. Dissertation, Dept. of Computer Science, Universität Tübingen (D), 2002.
- /Reactis/ Reactis Tester, Reactive Systems Inc., www.reactive-systems.com/tester.msp
- /Sim97/ D. Simmes: Entwicklungsbegleitender Systemtest für elektronische Fahrzeugsteuergeräte. Dissertation, Herbert Utz Verlag Wissenschaft, München (D), 1997
- /Simulink/ Model-Based and System-Based Design - Using Simulink (Version 5). The MathWorks, Natick (US), Juli 2002
- /STB/ Safety test Builder, TNI Valiosys, www.tni-software.com
- /SZ03/ J. Schäuffele, T. Zurawka: Automotive Software Engineering. Vieweg, Juli 2003
- /TMW/ The MathWorks Inc.: www.mathworks.com
- /Tur01/ Turlapati, R.: Leveraging Test Measurements Into Proposing Additional Domain Tests. Master's Thesis. Department of Computer and Information Sciences, East Tennessee State University, May 2001
- /WSB01/ Wegener, J., Stahmer H., Baresel, A.: Evolutionary Test Environment for Automatic Structural Testing. Special Issue of Information and Software Technology, Vol. 43, pp. 851-854, 2001

KONTAKT:

Dr.-Ing. **Mirko Conrad** studierte an den Technischen Universitäten Dresden und Berlin Informatik und promovierte 2004 an der TU Berlin. 1996 kam er zur DaimlerChrysler Forschung wo er im Labor Software-Technologien als Projektleiter im Bereich Modell-basierte Entwicklung/Modell-basierter Test tätig ist. Er ist Autor bzw. Co-Autor von mehr als 40 Veröffentlichungen auf dem Gebiet des Automotive Software Engineering. Er ist Mitglied der GI-Fachgruppen 'Testen, Analysieren und Verifizieren von Software' und 'Automotive Software Engineering', des Software Arbeitskreises der FAKRA Arbeitsgruppe 'Funktionale Sicherheit' sowie des MathWorks Automotive Advisory Boards (MAAB).

Ines Fey, studierte von 1990 bis 1995 Informatik in Berlin. Seit 1996 ist sie als wissenschaftliche Mitarbeiterin im Bereich Forschung Information und Kommunikation der DaimlerChrysler AG tätig. Schwerpunkte der Arbeits- und Forschungstätigkeit liegen im Bereich der Modell-basierten Entwicklung und des Tests von Software im Automobil.