Informatik Forsch. Entw. (2007) 22: 45–57
DOI 10.1007/s00450-007-0028-y

REGULÄRE BEITRÄGE

Das AutoMoDe-Projekt

Modellbasierte Entwicklung softwareintensiver Systeme im Automobil

Andreas Bauer · Manfred Broy · Jan Romberg · Bernhard Schätz · Peter Braun · Ulrich Freund · Nuria Mata · Robert Sandner · Pierre Mai · Dirk Ziegenbein

Eingegangen: 20 Oktober 2006 / Angenommen: 27 April 2007 / Online veröffentlicht: 6 Juni 2007

Zusammenfassung Die Entwicklung eingebetteter Software für Automobile ist inhärent komplex und vereint verschiedene Entwicklungsphasen, mehrere fachliche Disziplinen, sowie verschiedene Akteure in beteiligten Unternehmen. Der AutoMoDe-Ansatz zur Entwicklung automotiver Software beschreibt Systeme auf verschiedenen Abstraktionsebenen und definiert schrittweise Übergänge zwischen diesen Ebenen. Neben der Definition geeigneter Ebenen werden zur Modellierung von Echtzeitsystemen ein einheitliches Berechnungsmodell sowie domänenspezifische Beschreibungstechniken verwendet. Automatisierte Anbindungen für Analyse und Synthese komplexer Softwaresysteme mit dem Ziel eines konsistenzbetonten Entwicklungsprozesses wurden realisiert. Die beschriebenen Techniken wurden in den Werkzeugprototypen AutoFocus integriert und im Zusammenspiel mit einer Werkzeugkette demonstriert.

Schlüsselwörter Automotive Software Engineering · Eingebette Software · Synchrone Sprachen · AutoMoDe

Dieses Projekt wurde vom Bundesministerium für Bildung und Forschung (BMBF) unter dem Kennzeichen 01ISC08 gefördert

A. Bauer (✉) · M. Broy · J. Romberg · B. Schätz
Software&Systems Engineering, Institut für Informatik,
Technische Universität München,
Boltzmannstr. 3,
85748 Garching b. München, Deutschland
e-mail: baueran@in.tum.de

P. Braun
Validas AG,
Lichtenbergstr. 8,
85748 Garching b. München, Deutschland

U. Freund · N. Mata
ETAS GmbH,
Borsigstr. 14,
70469 Stuttgart, Deutschland

R. Sandner
BMW AG,
80788 München, Deutschland

P. Mai
PMSF IT Consulting,
Ludwig-Thoma-Str. 11,
87724 Ottobeuren, Deutschland

D. Ziegenbein
Robert Bosch GmbH,
Postfach 30 02 40, 70442 Stuttgart, Deutschland

Abstract Development of embedded automotive software is inherently complex and involves different stakeholders, phases, and disciplines. The AutoMoDe approach to automotive software development defines distinct levels of abstraction for integrated development, and defines stepwise transitions between the levels. Along with the definition of suitable abstraction levels, to support modeling of real-time systems, a homogeneous operational model along with domain-specific notations are used. Automated backend functionalities for analysis and synthesis of complex software systems, with the goal of a consistent development process, were devised. The techniques described have been integrated into the tool prototype AutoFocus and have been demonstrated by the construction of a tool chain.

Keywords Automotive software engineering · embedded software · synchronous languages · AutoMoDe

CR subject classification D.2.2

1 Einleitung

Die steigende Anzahl von Funktionalitäten, die von eingebetteten Systemen im Automobilbereich zur Verfügung gestellt werden, sowie die wachsende Tendenz, Funktionalitäten zu interoperierenden Netzwerken zu integrieren, erhöht die Komplexität softwareintensiver Systeme im Automobil. Die vorherrschende Verwendung von etablierten, stark an konkreten Implementierungsmechanismen sowie an Subsystemen orientierten Ansätzen zur Entwicklung dieser Systeme führt zu einem steigenden Problemdruck, insbesondere in Bezug auf Integration und Qualitätssicherung.

Die Komplexität stellt somit hohe Anforderungen an eine Entwicklungsmethodik. Zum einen soll die Interoperabilität von Funktionen bereits in frühen Stadien der Entwicklung evaluiert werden können. Hierzu ist eine explizite Modellierung der Abhängigkeiten zwischen Funktionen, der Schnittstellen zwischen Modulen, sowie eine Festlegung der Verhaltenssemantik notwendig. Des weiteren sollen Funktionsmodule zunächst unabhängig von der physischen Verteilung modelliert werden können, um Freiheitsgrade im Entwurf bezüglich des Mappings von Funktionen auf Steuergeräte zu erhalten. Diese Freiheitsgrade können dann später im Deployment, z.B. spezifisch nach unterschiedlichen Ausstattungsumfängen, kostenoptimal ausgenutzt werden. Schließlich soll durch geeignete methodische Vorgaben und wohldefinierte Schnittstellen- und Moduldefinitionen ein hoher Wieder- und Mehrfachverwendungsgrad erreicht werden. Über die genannten Punkte hinaus stellt sich mit dem Zusammenwachsen der Anwendungsgebiete diskreter Ereignissysteme und synchroner zeitgesteuerter Systeme die Aufgabe, einen einheitlichen Modellierungsansatz für beide Arten eingebetteter Software zur Verfügung zu stellen, um beide Aspekte eines eingebetteten Systems in Kombination beschreiben, analysieren, und bis hin zu einer Implementierung entwickeln zu können.

In diesem Artikel werden die Ergebnisse des AutoMoDe-Projektes vorgestellt, die von den Projektpartnern BMW AG, Robert Bosch GmbH, ETAS GmbH, Technische Universität München, sowie Validas AG im Zeitraum Oktober 2003 bis März 2006 erzielt wurden. Der vorgestellte Ansatz wird durch das AutoFocus-Werkzeug [9] unterstützt. AutoFocus bietet grafische und textuelle Beschreibungstechniken für verschiedene Abstraktionsebenen. Damit ist die Modellierung von Struktur und Verhalten von Software durchgängig möglich. Zusätzlich existieren verschiedene Anbindungen zur Konsistenzanalyse, zur formalen Verifikation, zur Testfallgenerierung [8], sowie zur Code-Generierung. Um die praktische Umsetzung der Konzepte zu überprüfen wurde eine Werkzeugkette bestehend aus etablierten Werkzeugen der ETAS aufgebaut. Anhand dieser konnte gezeigt werden, dass Modelle, modelliert mit dem AutoMoDe-Ansatz, in effiziente Implementierungen für eingebettete Hardware umgesetzt werden können.

Abschnitt 2 diskutiert verwandte Arbeiten zu den in AutoMoDe geleisteten Beiträgen. In Abschn. 3 wird ein kurzer Überblick des AutoMoDe-Ansatzes und der betrachteten Abstraktionsebenen gegeben. Abschnitt 4 erläutert kurz das AutoFocus zugrunde liegende Berechnungsmodell, bevor im darauffolgenden Abschn. 5 die eingesetzten AutoFocus-Notationen beschrieben werden. Das Beispiel der Modellierung einer Antischlupfregelung in Abschn. 6 dient dazu die Ergebnisse zu veranschaulichen und deren praktische Umsetzung zu überprüfen. Der Übergang zur Implementierung ist Gegenstand des Abschn. 7, bevor dieser Artikel mit einer Zusammenfassung endet.

2 Verwandte Ansätze und Beiträge von AutoMoDe

Der Beitrag von AutoMoDe kann in zwei Teilaspekte aufgespalten werden: Zum einen wurde als Ergebnis des Projekts ein methodisches Rahmenkonzept für die Entwicklung automotiver Softwaresysteme definiert. Zum anderen wurden technische Beiträge in den Bereichen Modellierungssprachen und deren semantischer Fundierung, Transformationssprachen für Software-Modellierungswerkzeuge, sowie verteilte Implementierung synchroner Datenflusssprachen durch das Projekt eingebracht.

Es existieren eine Reihe von verwandten Methoden für modellbasierten Entwurf automotiver Softwaresysteme (siehe z.B. [1, 11, 20, 21, 28]). Neben Unterschieden im Detail verwenden die meisten der zitierten Ansätze eine Anzahl von definierten Abstraktionsebenen sowie verschiedene unterstützende Werkzeuge und Notationen mit dem Ziel einer inkrementellen Entwurfsmethodik für automobile Software. Durch die Verwendung heterogener Notationen und Werkzeuge im Stadium des Entwurfs gelingt jedoch typischerweise keine enge Kopplung von Syntax- und Semantikkonzepten über Werkzeug-, Notations-, sowie Phasengrenzen hinweg: verschiedene Werkzeuge realisieren unterschiedliche und zum Teil nicht kompatible Semantiken, die oftmals keine formale Fundierung aufweisen, wie dies z.B. bei diversen UML-basierten Werkzeugen der Fall ist (siehe z.B. [2, 14]). AutoMoDe verwendet dagegen ein einheitliches und semantisch fundiertes Domänenmodell, das durch das AutoFocus-Werkzeug unterstützt wird. Durch die Homogenität des Domänenmodells können neuartige technische Beiträge wie die semantikerhaltende Transformation von Modellen durch Transformationssprachen eingebracht werden. Die semantische Fundierung von mechatronischen Systemen allgemein steht z.B. auch in der Arbeit [15] im Vordergrund. Im Gegensatz zum hier vorgestellten Ansatz werden dort jedoch kontinuierliche Modelle eingesetzt, die weniger abstrakt sind als Modelle in AutoMoDe, welches

z.B. Zeit und konkrete Signallaufzeiten abstrahiert (siehe Abschn. 4).

Von allen genannten Ansätzen stützt sich AutoMoDe im Speziellen auf Vorgängerprojekte Automotive [28] sowie EAST-EEA [1]. Dabei wurden verschiedene Defizite der Vorgänger adressiert: im Vergleich zu Automotive, dass auf der UML 1.x-Notation basierte, wird in AutoMoDe mit der AutoFocus-Notation ein explizites Modell für Komponenten und ihre (nachrichtenbasierten) Schnittstellen eingeführt, sowie Unterstützung zur Modellierung regelungstechnischer Algorithmen auf Basis diskreter, feingranularer multiraten Clocks (vgl. Abschn. 4). AutoFocus ist in wesentlichen Konzepten eng verwandt zu der Komponentendarstellung in UML 2.0, so dass die weiterführende Möglichkeit einer UML-standardkonformen Darstellung nicht als kritisch angesehen wird. Als Vorteil gegenüber einer rein standardbezogenen Darstellung können die Arbeiten mit AutoFocus jedoch auf ein unzweideutiges und für die Domäne geeignetes semantisches Modell abgestützt werden. Im Vergleich sowohl mit dem Automotive-Projekt als auch mit EAST-EEA wurde in AutoMoDe ein stärkerer Schwerpunkt auf die Modellierung und Erhaltung von *Verhaltenseigenschaften* eingebetteter Systeme gelegt. Mit dem gleichzeitigen Start der AUTOSAR-Entwicklungspartnerschaft [24] konnte AutoMoDe zudem von den technischen Ergebnissen her auf die "Virtual Functional Bus"-Architektur von AUTOSAR abgestimmt werden.

Gegenüber regelungstechnisch motivierten Werkzeugen wie Matlab/Simulink [19] oder ASCET [12] sowie aus anderen Anwendungsgebieten (z.B. Telekommunikation) stammenden Ansätzen wie UML 2.0 Komponentendiagrammen bietet der AutoFocus-gestützte Ansatz den Vorteil von domänenspezifische Konzepten wie z.B. Betriebsmodus, Funktion, oder Periode.

In diesem Artikel wird unter anderem die explizite Modellierung von *Betriebsmodi* als ein Mittel der grobgranularen Dekomposition eingebetteter Systeme als Ergebnis des AutoMoDe-Projekts beschrieben. Diese Idee wurde auch von anderen Autoren beschrieben, so z.B. [18]. Zusätzlich zur reinen Verwendung einer neuen Notation setzt unser Ansatz Modi über mehrere Abstraktionsebenen hinweg ein, und untersucht speziell Transformationen zwischen verschiedenen strukturierten, jeweils auf einen Einsatzzweck hin optimierte Modus-Hierarchien.

Die Idee, zeit- und ereignisgesteuerte Takte als Boolesche Ausdrücke (Clocks) zu definieren, sowie ein flankierendes Verfahren zur Inferenz und Überprüfung von Clocks, stammt aus dem Gebiet der synchronen Datenflusssprachen [6]. Im Vergleich zu den in [6] beschriebenen Ansätzen, unterscheidet sich das im Projekt AutoMoDe entworfene Inferenzverfahren, durch erhöhte Skalierbarkeit bei einer für die Praxis ausreichenden Ausdrucksmächtigkeit der Modellierungssprache.

3 Überblick

Der AutoMoDe-Ansatz vereint verschiedene Aspekte einer modellbasierten, durchgängigen Entwicklungsmethodik:

Domänenmodell: In AutoMoDe wurde ein integriertes Domänenmodell zur Entwicklung softwareintensiver eingebetteter Systeme im Automobilbereich definiert.

Notationen: Das Domänenmodell integriert Modellierkonzepte, die in der Entwicklersicht durch eine Anzahl von problemorientierten grafischen und textuellen Notationen repräsentiert werden. Dabei bieten unterschiedliche Diagramme, zum Beispiel für Verhalten und Struktur, jeweils alternative Sichten auf ein integriertes Modell.

Abstraktionsebenen und Transformationsschritte: Um den Ansatz insgesamt zu strukturieren, werden verschiedene Abstraktionsebenen eingeführt. Formalisierte Transformationsschritte dienen dem Übergang zwischen Abstraktionsebenen sowie der Umformung von Modellen auf gleicher Ebene mit dem Ziel der Erhaltung wesentlicher Eigenschaften.

Werkzeugkette: Der AutoMoDe-Ansatz wird durch eine Werkzeugkette unterstützt (siehe Abb. 1), die Editieren, Analyse, sowie Transformation von Modellen auf verschiedenen Abstraktionsebenen zulässt.

Die AutoMoDe-Entwurfsmethodik basiert stark auf einer Gliederung der Modellierungsartefakte in domänenspezifische Abstraktionsebenen ähnlich zu der in [1] vorgeschlagenen (siehe Abb. 2).

Für die Beschreibung der Modelle auf den abstrakteren Modellierungsebenen wird das AutoFocus-Werkzeug verwendet. Neben dem Editieren und Validieren von Modellen unterstützt AutoFocus verschiedene Arten von Modelltransformationen, die im Detail im folgenden Abschnitt und in [5] beschrieben werden. Für die konkreteren und stärker implementierungsbezogenen Abstraktionsebenen schließt

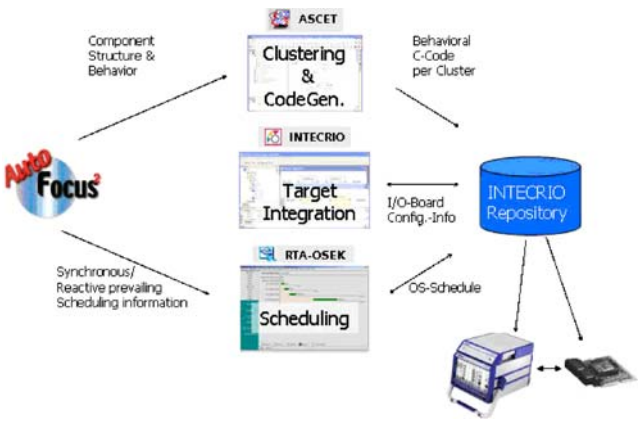

Abb. 1 AutoMoDe-Werkzeugkette

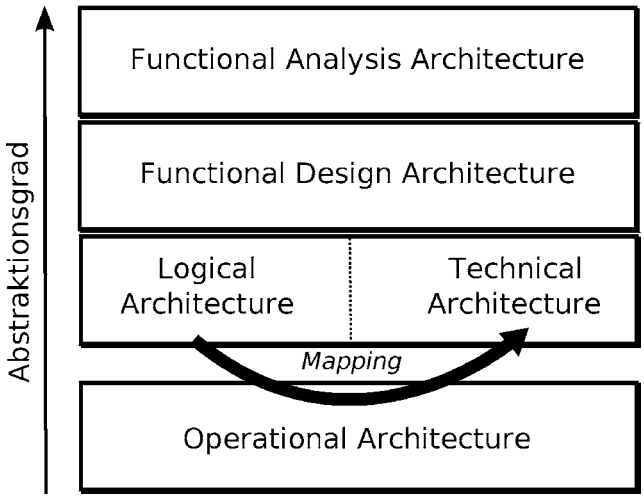

Abb. 2 AutoMoDe-Abstraktionsebenen

die AutoMoDe-Werkzeugkette die kommerziellen Werkzeuge ASCET [12], RTA OSEK Planner [17], sowie INTECRIO [13] mit ein. Die letztgenannten Werkzeuge zur implementierungsbezogenen Entwicklung und Synthese automotiver Softwaresysteme sind in dieser Domäne praktisch etabliert.

Functional analysis architecture (FAA). Die abstrakteste der betrachteten Ebenen ist die Functional Analysis Architecture. Sie ist eine fahrzeugweite und bzgl. der Struktur und der funktionalen Abhängigkeiten vollständige Beschreibung. Übergreifendes Ziel der Beschreibung auf FAA-Ebene ist vor allem die Identifikation wesentlicher Abhängigkeiten und eventueller Konflikte zwischen Funktionalitäten, sowie die Konzeptvalidierung anhand von unvollständigen oder prototypischen Verhaltensbeschreibungen. Der Begriff der *Funktionalität* bezeichnet eine benutzersichtbare Elementarfunktion auf FAA-Ebene. Die Beschreibung auf FAA-Ebene konzentriert sich dabei auf Kernelemente der eigentlichen Systemanwendung bzw. des Regelalgorithmus: Im Sinne der Anwendung untergeordnete Schichten wie z.B. Sensor- und Aktuatorvorverabeitung, Diagnosefunktionalitäten, sowie Systemüberwachung und Plausibilisierung werden nicht betrachtet. Auf FAA-Ebene werden die Funktionalitäten unabhängig von Realisierungsentscheidungen in Hardware oder Software betrachtet. Systeme sind auf FAA-Ebene typischerweise nach funktionalen Gesichtspunkten und somit vor allem nach Sichtbarkeit durch den Benutzer gruppiert. Als Beispiel für eine solche Gruppierung kann man unter dem Begriff „Längsdynamiksteuerung“ alle diejenigen Funktionalitäten versammeln, die einen Einfluss auf die (durch den Benutzer wahrgenommene) Längsdynamik eines Fahrzeuges haben, also z.B. die Vortriebs-/Motorsteuerung, die Steuerung der Bremse, sowie möglicherweise eine Antischlupfregelung. Ein und dieselbe Komponente kann dabei mehrfach in verschiedenen Kontexten ohne weitere Einschränkung vorkommen (als Typ bzw. Referenz).

FAA-Darstellungen sind typischerweise *strukturorientiert*: während die Strukturierung in Funktionalitäten und Abhängigkeiten bzw. Datenflüsse essentiell für die FAA-Modellierung ist, spielt das Verhalten der Einzelfunktionalitäten eine untergeordnete Rolle. Somit wird die FAA in AutoFocus primär durch die in Abschn. 5 beschriebenen Systemstrukturdiagramme (SSDs) repräsentiert, und in untergeordneter Weise durch die AutoFocus-Notationen für detailliertes Verhalten. Mit AutoFocus können vielfältige, FAA-bezogene Konsistenzbedingungen überprüft und durchgesetzt werden.

Functional design architecture (FDA). Die *functional design architecture* stellt bereits eine bzgl. der Struktur und des Verhaltens vollständige Beschreibung von Software-Systemen im Automobil dar. Der methodische Schwerpunkt liegt dabei auf einer Verifikation des Softwareverhaltens sowie auf der Identifikation gemeinsamer und wiederverwendbarer Softwarekomponenten. Gegenüber der FAA steht eine stärker realisierungsorientierte und weniger nutzerorientierte Strukturierung des Systems im Vordergrund. So kann z.B. die obengenannte nutzerbezogene FAA-Strukturierung von Fahrdynamikfunktionen in „Längsdynamiksteuerung“ und „Querdynamiksteuerung“ auf FDA-Ebene durch eine stärker technische, organisatorische, sowie wiederverwendungsbezogene Strukturierung ersetzt werden. Bezüglich der Vielfachheit von Komponenten gilt, dass auf der FDA-Ebene eine Minimierung des wiederholten Auftretens einer gegebenen Komponente angestrebt wird. Der methodische Nutzen ist dabei, dass durch die Zusammenfassung identischer Funktionalitäten in einmalig auftretenden Komponenten Potenziale für Mehrfachverwendung aufgedeckt werden können, die sich beim Übergang von einer funktionsorientierten Strukturierung (FAA) hin zu einer an der Realisierung orientierten Struktur ergeben. Ein Beispiel für ein derartiges Potenzial ist z.B. die zunächst häufig mehrfach eingeplante Vorverarbeitung und Aufbereitung von Sensorsignalen, die in der späteren Realisierung an einer Stelle aufbereitet werden sollten, um anschließend verteilt kommuniziert und verwendet zu werden.

FDA-Komponenten werden in AutoFocus durch Komponenten in Systemstrukturdiagrammen beschrieben: sie können somit wiederum hierarchisch durch andere Komponenten aufgebaut sein, und sind über typisierte, gerichtete Kanäle und Konnektoren miteinander verbunden. Die Verhaltensdarstellung ist in AutoFocus mit den in Abschn. 5 beschriebenen Notationen zur Verhaltensdarstellung möglich. Die Überprüfung FDA-spezifischer Konsistenzbedingungen wie z.B. Struktur- oder Verhaltensvollständigkeit kann ebenfalls durch AutoFocus vorgenommen werden. Hierbei wird auf syntaktischer Ebene die Vollständigkeit der FDA-Modelle überprüft.

Logical/technical architecture (LA/TA). Die detaillierteste AutoMoDe-Abstraktionsebene gliedert sich in Logical Architecture und Technical Architecture. In der LA werden die Software-Komponenten in so genannte Cluster gruppiert: dabei repräsentiert ein Cluster eine kleinste verteilbare Einheit, z.B. im Sinne eines Tasks oder einer Routine. Als Strukturierungskriterium treten somit noch stärker als in der FDA technische Eigenschaften wie gemeinsame Aktivierungsfrequenz, Priorität und Kritikalität in den Vordergrund. Technische Informationen wie z.B. Takte/Perioden oder Implementierungstypen müssen auf der Logical Architecture vollständig spezifiziert sein. Die technical architecture (TA) repräsentiert alle für die Zuordnung von logischen Clustern zu technischen Ressourcen relevanten Konzepte, wie z.B. Steuergeräte, Busse und Slots bzw. Nachrichten der Kommunikationsmedien. Die eigentliche Zuordnung von Clustern zu Steuergeräten bzw. Datenflüssen zu Kommunikationsmedien erfolgt in der hier nicht weiter behandelten *operational architecture (OA)*, die dem aus den implementierungsorientierten Werkzeugen synthetisierten HW/SW-System entspricht.

Die LA kann auszugsweise in AutoFocus modelliert werden: zu diesem Zweck sind vor allem die Modellierungskonzepte Clocks (für heterogene Aktivierungsfrequenzen) sowie Implementierungstypen hilfreich. In vollständiger Ausprägung können LA und TA in ASCET, INTECRIO, sowie RTA OSEK Planner modelliert werden.

4 Berechnungsmodell

Das dem AutoMoDe-Projekt zugrundeliegende Berechungsmodell von AutoFocus verwendet eine nachrichtenbasierte, taktsynchrone Semantik mit uniformem, systemweitem Takt [10]. Dabei werden Rechenschritte innerhalb eines Taktes nicht weiter zeitlich aufgelöst. Modelle in AutoFocus werden als Netzwerke von *Komponenten* bzw. *Blöcken* definiert, die *Nachrichten* mit der Umgebung und untereinander über explizite Schnittstellen (Nachrichtenports) sowie Konnektoren zwischen Schnittstellen (Nachrichtenkanäle) austauschen. Nachrichten besitzen im abstrakten Modell einen Zeitstempel in Bezug auf den globalen Takt. Dieses datenflussorientierte Berechnungsmodell unterstützt einen hohen Grad an Modularität dadurch, dass Komponentenschnittstellen vollständig und explizit durch syntaktische Konstrukte abgebildet werden. Das Berechnungsmodell verringert den Grad an Komplexität in Echtzeitsystemen: Das Nachrichtenparadigma mit diskreter Zeitbasis abstrahiert von Implementierungsdetails wie detailliertem Timing, kausaler Abfolge interner Berechnungsschritte, und verwendeten Kommunikationsmedien.

Beispiele für solche Implementierungsdetails beinhalten die Reihenfolge der Ankunft von Implementierungsnachrichten mit demselben logischen Zeitstempel, oder die genaue Dauer des Datentranfers auf einem Kommunikationsmedium. Somit werden Echtzeitintervalle der Implementierung durch logische Zeitintervalle abstrahiert. Das nachrichtenorientierte, zeitdiskrete Paradigma kann sowohl periodische als auch sporadische Berechnungen abbilden, wie es z.B. für die Modellierung von zeitgesteuertem und ereignisgesteuertem Verhalten notwendig ist.

Um die heterogenen (a) periodischen Takte bzw. Frequenzen typischer Automotive-Anwendungen komfortabel modellieren zu können, werden in AutoFocus so genannten *Clocks* [7] verwendet. Jeder Nachrichtenstrom in AutoFocus ist mit einer Clock assoziiert: Die Clock ist dabei ein Boolescher Ausdruck, welcher die Anwesenheit bzw. Abwesenheit einer Nachricht eines Nachrichtenstroms beschreibt. Zur Laufzeit evaluiert der Ausdruck zu logisch wahr, wenn der Nachrichtenstrom eine Nachricht enthält.

Mit expliziten *Sampling-Operatoren* zwischen Komponenten bzw. Clustern kann ein Datenfluss von einer Clock auf eine andere überführt werden. Mit Hilfe von strukturell über den Elementaroperatoren der Sprache definierten Regeln ist es dann möglich, für jedes AutoFocus-Konstrukt *Vorbedingungen* sowie *Inferenzregeln* bezüglich der Clocks anzugeben. Das AutoFocus-Werkzeug kann sowohl die Korrektheit des Designs in Bezug auf die Clocks (Erfüllung aller Vorbedingungen) überprüfen, als auch auf sämtliche interne und ausgabeseitige Clocks eines Systems schließen (Anwendung der Inferenzregeln).

5 AutoFocus-Notationen

AutoFocus bietet eine Anzahl von verschiedenen Notationen, die sich bei der Modellierung eingebetteter Systeme bewährt haben. Diese werden im Folgenden beschrieben.
Systemstrukturdiagramme (SSD). Systemstrukturdiagramme bieten eine architekturbezogene Gesamtsicht auf ein Software-System und werden für die Abstraktionsebenen FAA und FDA verwendet. Dabei werden Schnittstellen, Kommunikationsabhängigkeiten und hierarchische Dekomposition hinsichtlich der Funktionsarchitektur (FAA) bzw. der Softwarearchitektur (FDA) beschrieben. Dabei wird das System als Netzwerk von reaktiven *Komponenten* mit taktweiser Berechnung – einschließlich einer Eingabe- und einer Ausgabeinteraktion – beschrieben, die Nachrichten und Signale über gerichtete *Kanäle* untereinander austauschen. Komponenten sind entweder atomar, oder setzen sich hierarchisch aus Subkomponenten, die in einem eigenen SSD spezifiziert sind, zusammen. Durch die taktweise Interaktion von Komponenten bietet deren architektonische Modularität bereits auf hoher Abstraktionsebene „Sollbruchstellen“ für die spätere Partitionierung, ohne diese konkret vorwegzunehmen.

Abbildung 3 zeigt ein typisches SSD für die Darstellung der in Abschn. 6 beschriebenen Antischlupfregelung. Das SSD ist Teil eines Gesamtsystems. Die nicht einer Komponente zugeordneten Ports definieren dabei die Schnittstelle des Systemausschnitts zum Restsystem. Ein Ziel der FAA-Modellierung ist die Identifikation eventueller Konflikte zwischen Funktionalitäten. Durch die konsequent an funktionalen Gesichtspunkten orientierte Strukturierung existiert im FAA-Modell in diesem Fall eine eindeutige, hier durch `VehicleMotionCoordinator` repräsentierte Stelle im Modell, an der ein möglicher Koordinationsbedarf zwischen den (funktional eng verwandten) Anfragen `DesiredThrottlePositionDriver` und `EngineIntervention` offensichtlich wird.

Im Gegensatz zu den mit SSDs modellierten Funktionalitäten auf FAA-Ebene müssen SSD-modellierte Softwarekomponenten auf FDA-Ebene ein definiertes Verhalten aufweisen. Verhaltensspezifikationen atomarer Softwarekomponenten sind in Form von *Datenflussdiagrammen (DFDs)* mit algorithmischen Datenfluss, *mode transition diagrams (MTDs)* mit am Betriebsmodus orientierter dekomponierter Darstellung, sowie mit *state transition diagrams (STDs)* für eine automatenorientierte, ereignisgesteuerte Darstellung möglich.

Datenflussdiagramme (DFD). Obwohl ähnlich in der Struktur zu SSDs, beschreiben Datenflussdiagramme den algorithmischen Datenfluss im Zuge des Berechnungsschrittes von Komponenten. DFDs werden typischerweise auf allen Abstraktionsebenen eingesetzt. Auf FAA-Ebene überwiegen dabei prototypische Verhaltensbeschreibungen. DFDs sind aus atomaren oder hierarchischen *Blöcken* aufgebaut, die ähnlich zu SSDs über gerichtete und typisierte Kanäle verbunden sind. Das Verhalten von atomaren Blöcken kann entweder durch ein Mode Transition Diagram (MTD, siehe folgende Abschnitte), ein State Transition Diagram (STD, siehe [9]), oder durch einen textuellen Ausdruck in der Basissprache von AutoFocus [16] gegeben sein. Hierarchische Blöcke werden wiederum durch DFDs definiert.

In Abb. 4 ist als Beispiel für ein DFD ein Algorithmus zur Erkennung von Radschlupf bei Fahrzeugen abgebildet. So ist zum Beispiel der Block `Difference` durch den Ausdruck `ReferenceSpeed - WheelSpeed` definiert, wobei `ReferenceSpeed` und `WheelSpeed` die Eingangsports des Blocks bezeichnen (in der grafischen Darstellung nicht gezeigt). Durch diesen Mechanismus ist es möglich, Blockbibliotheken für diskrete Steuerungs- und Regelalgorithmen aufzubauen, ähnlich zu kommerziellen Werkzeugen.

Da ein Berechnungsschritt innerhalb eines Taktes innerhalb von Komponenten gekapselt ist, abstrahiert ein DFD von den detaillierten zeitlichen Aspekten der Teilberechnungen, ähnlich zu synchronen Sprachen [6]. Im AutoFocus-Werkzeug wird die Verwendung von synchroner Kommunikation von einer Kausalitätsprüfung begleitet, die Modelle mit unverzögerten Schleifen ablehnt, da diese nicht implementierbar sind. Die Modellierung „gleichzeitiger“ Aktionen im Modell auf FAA-, FDA- und LA-Ebene stellen eine Abstraktion von möglichen sequenziell geordneten, zeitbehafteten Berechnungen und Nachrichtentransfers auf der Ebene der OA dar. Die Berechnungen einer Komponente werden taktweise beobachtet: dieser Takt definiert somit implizit die maximale Zeit für die Summe sequenzieller Berechnungen auf der OA-Ebene.

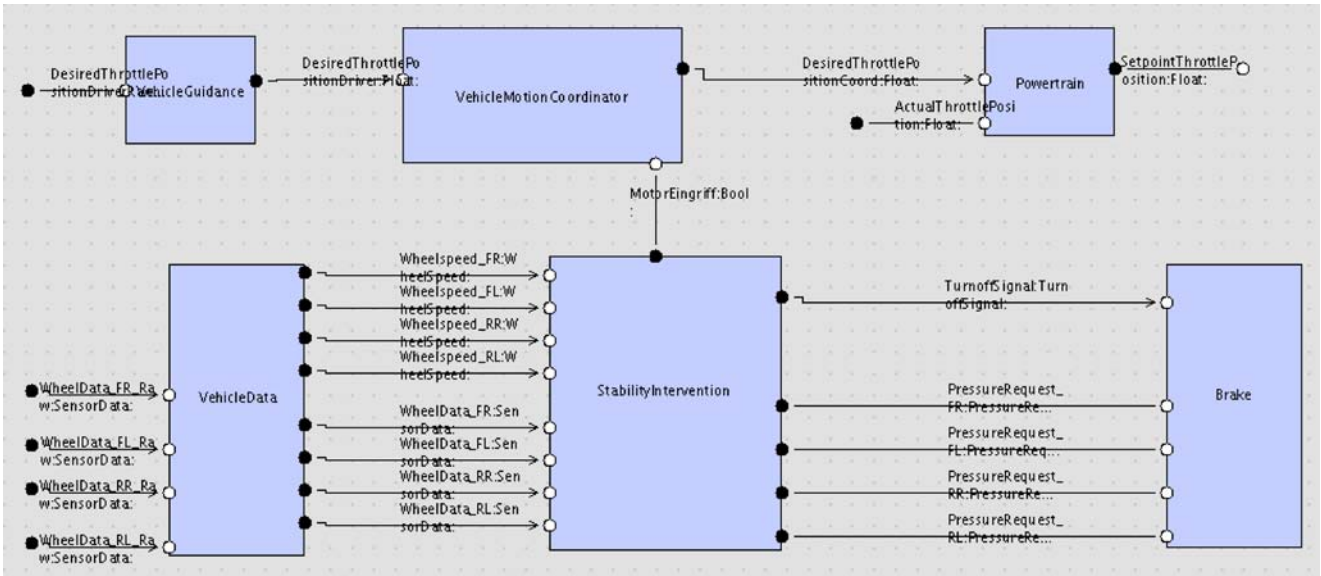

Abb. 3 Beispiel für ein SSD auf FAA-Ebene

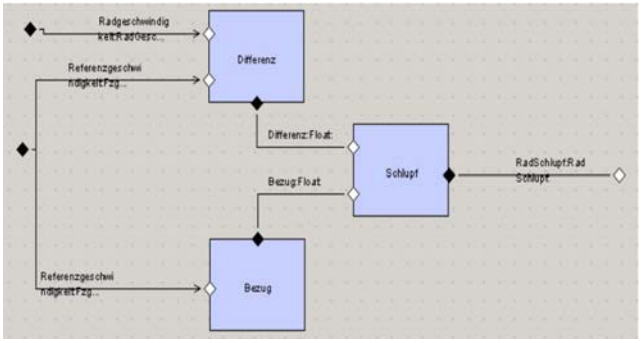

Abb. 4 Beispiel für ein DFD zur Radschlupferkennung

Mode-Transition-Diagramme (MTD). Die gegenüber [9] in AutoFocus neu eingeführten Mode Transition Diagrams werden auf allen Abstraktionsebenen dazu verwendet, zwischen alternativen Betriebsmodi oder Konfigurationen einer Komponente zur Laufzeit hin- und herzuschalten. MTDs bestehen aus Modi und Transitionen zwischen Modi. Dabei ist die grafische Darstellung ähnlich der eines endlichen Automaten: Knoten des Graphen entsprechen Modi, Kanten entsprechen Transitionen. Jede Transition ist mit einem Booleschen Ausdruck beschriftet, der sich auf die Eingangsports der durch das MSD definierten Komponente bezieht. Bei der Ausführung wird in einem gegebenen Systemschritt der Modus über eine Transition gewechselt, falls die Auswertung der Booleschen Bedingung logisch wahr ergibt. Jedem Modus ist dabei ein untergeordnetes Verhalten in Form eines SSD oder DFD zugeordnet.

Das Verhalten der Komponente zu einem bestimmten Zeitpunkt wird durch das mit dem aktiven Betriebsmodus assoziierten untergeordneten SSD oder DFD definiert. Untergeordnete Diagramme können beliebig tief hierarchisch verfeinert sein. Abhängig vom Modus eines MTDs können so unterschiedliche, den Modus zugeordnete DFDs, als Berechnungsvorschrift einer Komponente aktiviert sein. Sie stellen ein wichtiges und bislang in der Anwendung wenig etabliertes Mittel der Fein- und Grobdekomposition von eingebetteten Systemen dar.

Abbildung 5 zeigt ein Beispiel aus einer AutoMoDe-Fallstudie für die Modellierung von Betriebszuständen

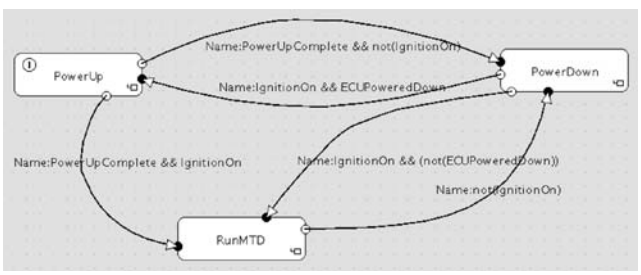

Abb. 5 Beispiel für ein MTD mit Betriebsmodi einer Motorsteuerung

einer Motorsteuerung. Dabei wird zwischen den Modi `PowerDown` (ausgeschalteter Zustand), `PowerUp` (Hochlauf der Steuerung) sowie `RunMTD` (betriebsfähiger Zustand) unterschieden und je nach Wertbelegung der Booleschen Eingänge `PowerUpComplete` (Beendigung der Hochlaufphase) und `IgnitionOn` (Stellung des Zündschlosses) zwischen den Modi umgeschaltet.

Cluster-Communication-Diagramme (CCD). Cluster Communication Diagrams werden auf der obersten Hierarchieebene der LA verwendet und zeigen eine an der Verteilbarkeit orientierte Sicht auf die Software-Komponenten. Cluster sind die kleinsten verteilbaren Einheiten des Software-Systems, d.h. ein Cluster wird nie auf mehrere Tasks einer Applikation verteilt. Die Partitionierung der FDA-Komponenten in Cluster kann sich dabei beispielsweise an den Clocks orientieren. Je nach Scheduling-Strategie sind an bestimmten Clustergrenzen Verzögerungen im Modell erforderlich: durch geeignete Restrukturierung können an diesen Stellen z.B. die durch die SSD-Strukturierung eingebrachten Verzögerungen ausgenutzt werden.

Die grafische Notation für CCDs entspricht der von DFDs. Auf CCDs gelten jedoch weitere Einschränkungen: Cluster können keine Subcluster enthalten. Ausserdem werden die durch die gewählte Scheduling-Strategie erforderlichen Verzögerungen zwischen Clustern auf den CCDs überprüft und durchgesetzt.

Auf der LA-Ebene werden Implementierungstypen eingeführt. Diese bilden plattformbezogene Eigenschaften des Modells in Bezug auf die Implementierung ab. Ein Implementierungstyp ist die konkrete Realisierung eines abstrakten Typs, so wie z.B. `Int8` die 8-bit-Realisierung des abstrakten Typs `Integer` darstellt. Die durch den Übergang auf Implementierungstypen eingebrachten Änderungen des Verhaltens können in AutoFocus simuliert werden.

Implementierung von CCDs als kommunizierende Echtzeittasks. CCDs können durch kommunizierende, von einem Echtzeitbetriebssystem gesteuerte Tasks implementiert werden. Typische Implementierungsplattformen im Automobil verwenden zum Teil präemptives Scheduling. Damit stellen sich einige technische Herausforderungen für die semantikerhaltende Implementierung von deterministischen, logisch gezeiteten Modellen, wie sie in der CCD-Beschreibung inhärent sind. In [23] ist ein Verfahren beschrieben, wie CCDs mit beliebigen Multiraten auf der Basis von präemptivem Scheduling mit fest vergebenen Prioritäten implementiert werden können. Die Methode basiert auf einer ratenmonotonen Abbildung von mit Clocks versehenen Clustern zu priorisierten Echtzeittasks. Dabei wird der Cluster mit der kleinsten Deadline oder Periode auf die Task mit der höchsten Priorität abgebildet. Kommunikation zwischen Clustern muss in bestimmten Fällen durch doppelt gepufferte Interprozesskommunikation (IPC) [22] abgesichert werden, um die Semantik

des Modells in der Implementierung zu erhalten. Die auf der Implementierungsebene auftretenden Verzögerungen werden dabei auf Modellebene durch die logisch gezeiteten Verzögerungen an den Clustergrenzen abgebildet. In der Methodik aus [23] wird gezeigt, dass unverzögerte Kommunikation auf Modellebene nur in speziellen Situationen durch Inter-Task-Kommunikation realisiert werden kann: Dies ist ein Grund für die oben beschriebenen Einschränkungen an CCDs.

6 Beispiel: Antischlupfregelung

Am Beispiel einer Antischlupfregelung (traction control system/TCS) wird nun der AutoMoDe-Ansatz zur schrittweisen Entwicklung automotiver Software beschrieben.

Die Antischlupfregelung sorgt dafür, dass die Räder beim Beschleunigen nicht durchdrehen. Beim Anfahren mit zu viel Gas oder bei schlechtem Fahrbahnuntergrund können einzelne Räder durchdrehen. Die Antischlupfregelung vergleicht die Raddrehzahlen der Antriebsräder mit der Fahrzeuggeschwindigkeit. Raddrehzahlen, die einer Fahrzeuggeschwindigkeit über der tatsächlichen Fahrzeuggeschwindigkeit entsprächen, deuten auf Schlupf des entsprechenden Rads hin. Die Ursache für diesen Schlupf ist wahrscheinlich ein zu hohes Antriebsdrehmoment in Relation zum Fahrbahnuntergrund. Im Fall von Schlupf wird üblicherweise eine der folgenden Aktionen ausgeführt:

1. Wenn nur eines der beiden Antriebsrädern Schlupf hat, wird das zu schnelle Rad abgebremst.
2. Falls beide Antriebsräder Schlupf haben, wird das Antriebsdrehmoment reduziert. Im Fall eines Ottomotors mit „elektronischem Gaspedal“ kann die Leistungsreduzierung durch die im Beispiel verwendete Anpassung der Drosselklappenstellung herbeigeführt werden.

Üblicherweise interagiert die Antischlupfregelung eng mit dem Antiblockiersystem (ABS). Sowohl das TCS, wie auch das ABS verwenden die Raddrehzahlen und bremsen einzelne Räder getrennt ab.

Das TCS-Modell. Das Verhalten der `StabilityIntervention`-Komponente aus Abb. 3 ist als DFD definiert (siehe Abb. 6). Das DFD enthält die notwendigen Komponenten für die allgemeine Stabilitätskontrolle und die Fahrdynamik-Eingriffe. Neben der Kern-Komponente `Traction_Control` ist die `StabilityIntervention`-Komponente unterteilt in eine Referenzgeschwindigkeitsbestimmung, die Antischlupfregelung, der Brems-Schlupf-Regelung, der Brems-Verzögerungsregelung und einer Koordinierung der einzelnen Bremsanforderungen. Das Ergebnis der Koordinierung der Bremsanforderungen dient als

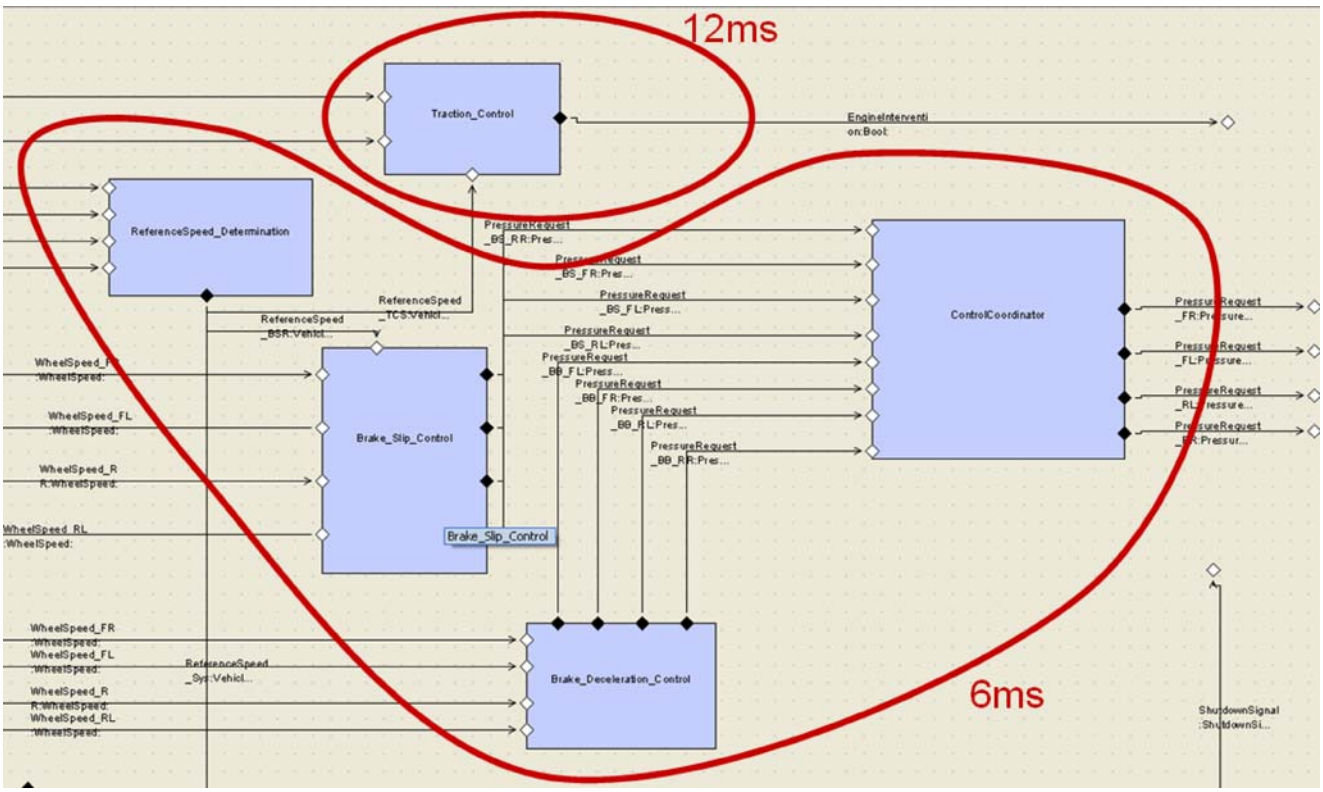

Abb. 6 DFD der Komponente `StabilityIntervention`

Druckvorgabe für die einzelnen Radbremszylinder. Die Abbildung zeigt zudem schematisch die Ausführungsfrequenz der verschiedenen Komponenten, die durch Clocks der entsprechenden Kanäle spezifiziert wurde (in der Abbildung sind die Clocks nicht zu sehen).

Abbildung Abb. 7 zeigt die hierarchische Dekomposition der `Traction_Control`-Komponente (vgl. Abb. 6). Basierend auf den Radgeschwindigkeitssignalen wird eine Referenzgeschwindigkeit errechnet und mit den einzelnen Radgeschwindigkeiten verglichen. Der daraus ermittelte Radschlupf wird mit der Referenzgeschwindigkeit normalisiert. Für jedes Antriebsrad (im Beispiel die Hinterräder) wird bestimmt, ob der Wert über dem Grenzwert (`AboveLambda2`) oder unter dem Grenzwert (`BelowLambda1`) liegt. Auf Basis dieser Klassifikation wird bestimmt, ob eine Leistungsanpassung des Motors über die Drosselklappe erfolgen soll.

Abbildung Abb. 4 zeigt schematisch die Radschlupferkennung (`WheelSlip`), wie sie konkret in den Komponenten `TCS_Compute_WheelSlip_RX` vorgenommen wird. Auf deren Basis wird die oben beschriebene Klassifikation, wie in Abb. 8 gezeigt, definiert. Die Komponente zur Regelung eines Eingriffs in die Motorsteuerung aus Abb. 7 beeinflusst die Stellung der Drosselklappe, falls der

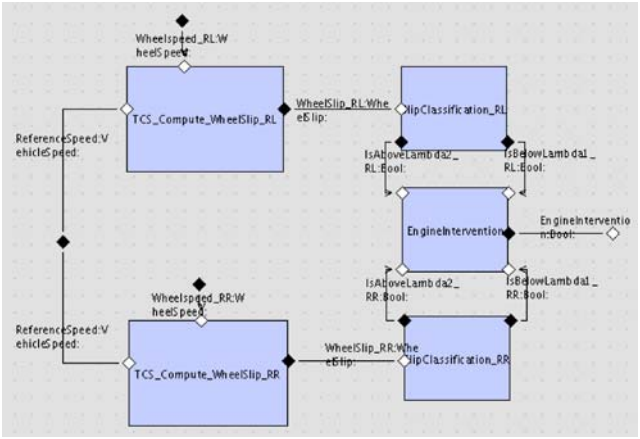

Abb. 7 DFD der Komponente `Traction_Control`

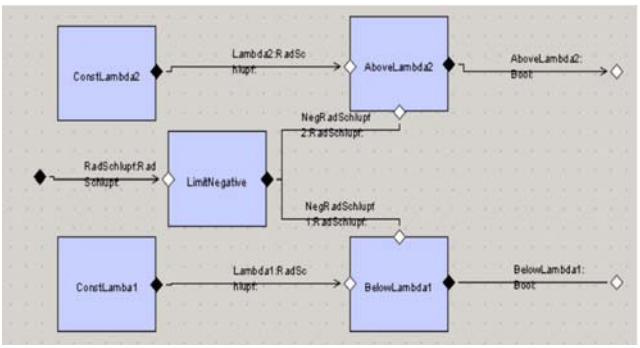

Abb. 8 DFD der Komponente `SlipClassification_RX`

Schlupf beider Antriebsräder außerhalb der Grenzwerte ist.

Der Umfang der Drosselklappenkorrektur wird in der Drosselklappensteuerung (`ThrottleControl`) bestimmt. Diese ist Teil der `Powertrain`-Komponente, die oben rechts in der Abb. 3 zu sehen ist. Die Ansteuerung der Drosselklappe ist ein Regelungssystem. Dieses besteht aus einem PID-Regler, der den Drosselklappenstellmotor ansteuert.

Entsprechend der zeitlichen Auflösung der Sensoren und Aktuatoren werden Teile der Regelung mit unterschiedlichen Frequenzen ausgeführt. Im Beispiel läuft die Radgeschwindigkeits- und die Referenzgeschwindigkeitsbestimmung in 6 ms Tasks, während die Antischlupfregelung in einer 12 ms Task läuft. Die Drosselklappenansteuerung erfolgt im 1 ms Raster (vgl. Abb. 6). In AutoFocus werden die verschiedenen Ausführungsraster durch verschiedene Clocks beschrieben. In ASCET bzw. INTECRIO werden Cluster mit unterschiedlichen Ausführungsrastern auf Tasks abgebildet. Dabei wird jede Task mit einem der Clock entsprechenden Ausführungsraster versehen.

7 Übergang nach ASCET/INTECRIO

Die abstrakte Modellierung basierend auf Nachrichtenströmen und diskreter Zeit in AutoFocus und die Validierung der Regelungsalgorithmen auf FDA- und LA-Ebene ist nur ein Aspekt bei der Entwicklung automotiver Software. Um die Eigenschaften der Modelle auf den abstrakten Ebenen zu erhalten, ist es notwendig, dass die Übersetzung in echtzeitfähige Systeme die Semantik der Modelle erhält. Die Synthese eines echtzeitfähigen Executables aus ASCET besteht aus Tasks, die ihrerseits `void(void)`-Routinen aufrufen (d.h. keine Rückgabe- oder Funktionsargumente auf dem Stack). Diese entsprechen dem Prozess-Konzept in ASCET. Kommunikation zwischen ASCET-Prozessen und -Tasks erfolgt entweder über Inter-Prozess-Kommunikationsnachrichten (*IPC-Nachrichten*) oder über *globale Variablen*. ASCET-*Module* gruppieren Prozesse und Nachrichten. Module können wiederum in einem ASCET-*Projekt* zusammengefasst werden.

Das Verhalten von ASCET-Prozessen kann zum einen durch *Zustandsmaschinen*, mit Hilfe der C ähnlichen Sprache *ESDL*, mittels *C-Code* oder in Form von *Blockdiagrammen* erfolgen. Die AutoMoDe-Werkzeugkette verwendet in erster Linie Blockdiagramme, da so die AutoFocus-Modelle auf die natürlichste Weise dargestellt werden können. Jedes ASCET-Blockdiagramm besteht aus elementaren *Operatoren*, *Variablen* und einer totalen Ordnung von Zuweisungen bezüglich eines Prozesses, die durch sogenannte „*Sequence Calls*“ angegeben wird. ASCET-Blockdiagramme sind in ihrer Semantik sehr ähnlich zu imperativen Programmier-

Name	ConstLambda2
Clock	
Function	MakeWheelSlip(-0.2)

Abb. 9 Eigenschaften des atomaren Blocks `ConstLambda2`

Name	AboveLambda2
Clock	
Function	wheel_slip(WheelSlip)<=wheel_slip(Lambda2)

Abb. 10 Eigenschaften des atomaren Blocks `AboveLambda2`

sprachen. Ihre Syntax und Semantik unterscheidet sich daher von AutoFocus-DFDs.

Die AutoMoDe-Werkzeugkette wurde entwickelt, um AutoFocus-Modelle in echtzeitfähige Software zu übersetzen. Die Kette besteht aus einem AutoFocus-Plugin und den ETAS Werkzeugen ASCET [12], RTA-OSEK [17] und INTECRIO [13] (siehe Abb. 1). Das Plugin überführt Cluster eines AutoFocus-Modell in ASCET-Module und -Prozesse. Mit dem ASCET-Codegenerator wird dann C-Code für die Cluster erzeugt. Anschließend werden die Cluster auf ein „Rapid Development System" überführt. Die Integration auf der Zielplattform umfasst die Festlegung eines Betriebssystem-Schedules. Dieses wird aus den Clocks der AutoFocus-Modelle abgeleitet. Zusätzlich müssen die hardware-spezifischen Schnittstellen manuell hinzugefügt werden. Im Folgenden werden der Reihe nach die verschiedenen Transformationsschritte beschrieben.

Modulidentifizierung und Sequenzialisierung. Der AutoMoDe-Übersetzungsalgorithmus transformiert CCD-Cluster in ASCET-Module, wobei jedes Modul einen ASCET-Prozess beinhaltet. Wie in Abschn. 5 beschrieben ist, entsprechen die Cluster den „kleinsten verteilbaren Einheiten". Jeder Cluster wird durch eine Verhaltensbeschreibung in AutoFocus, wie beispielsweise einem möglicherweise hierarchischen DFD, definiert.

Jeder Cluster entspricht einem flachen Blockdiagramm in ASCET[1]. Aus den hierarchisch strukturierten DFDs werden die atomaren AutoFOCUS-Blöcke in eine Anzahl von Operatoren, Verbindungen, IPC-Nachrichten und den entsprechenden Ausführungssequenznummern in ASCET übersetzt. Die Sequenzialisierung kann dabei direkt aus der kausalen Ordnung, die der AutoFocus-Semantik zugrunde liegt, abgeleitet werden.

Als ein einfaches Beispiel einer Blockdiagrammübersetzung wird die Grenzwertüberprüfung in der Komponente `SlipClassification_RX` betrachtet (vgl. Abb. 8). `RX` steht dabei für `RL` (Rear Left) oder `RR` (Rear Right). Die Komponente überprüft, ob der tatsächliche Schlupf über dem Grenzwert `lambda2` liegt. Der Block `ConstLambda2` stellt die Konstante `lambda2` zur Verfügung und ist durch einen einfachen textuellen Ausdruck definiert (vgl. Abb. 9). Der Block `AboveLambda2` liefert das Ergebnis des Vergleichs von `Lambda2` und `WheelSlip`. Die Selector-Funktion `wheel_slip` wird benötigt, da der Typ

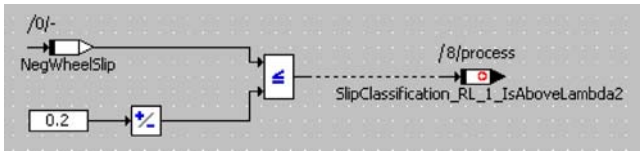

Abb. 11 ASCET-Blockdiagramm für `AboveLambda2`

`WheelSlip` als ein „Wrapper"-Datentyp in AutoFocus realisiert ist: `data WheelSlip = MakeWheelSlip (wheel_slip:Float)`. Abbildung Abb. 11 zeigt die Übersetzung dieser beiden Blöcke in ein ASCET-Blockdiagramm.

Die vollständige Übersetzung in das ASCET-Blockdiagramm des Clusters `SlipClassification_RX` aus Abb. 8 wird in Abb. 12 gezeigt. Das DFD zum `SlipClassification_RX`-Cluster wird in ein Blockdiagramm mit drei Ausführungsschritten innerhalb eines Prozesses übersetzt. Die Anzahl der notwendigen ASCET-Prozesse ist abhängig von der Anzahl der Clocks, die dem AutoFocus-Modell zugrunde liegen. Im Beispiel von `SlipClassification_RX` enthält das Modell nur eine Clock. Daher wird für die Übersetzung ein ASCET-Prozess benötigt. Die Zuweisung von Ausführungsreihenfolgen folgt den Kausalitäten, die inhärent in einem DFD festgelegt sind. Im Beispiel werden drei Ausführungsschritte benötigt. Zuerst wird der Variable `NegWheelSlip` als Teil des Ausdrucks des IF-Anweisung bestimmt (`/5/process`). Anschließend werden `BelowLambda1` und `AboveLambda2` bestimmt (`/7/process` and `/8/process`).

Die Übersetzung ist optimiert, so dass nur eine minimale Anzahl von internen Hilfsvariablen, die die Kanäle repräsentieren, benötigt wird. Im Beispiel wird die in-

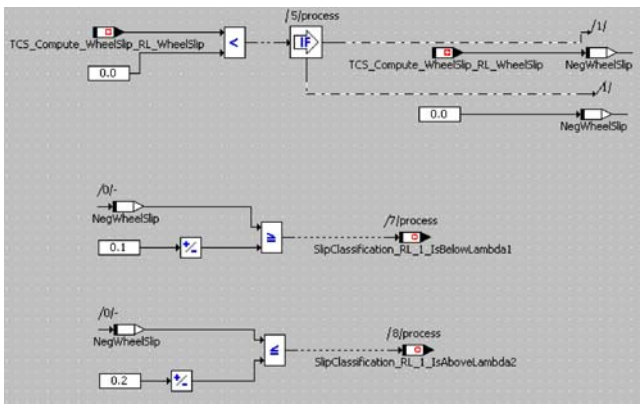

Abb. 12 ASCET-Blockdiagramm `SlipClassification_RX`

[1] Die Übersetzung der Hierarchie ist technisch zwar möglich, wurde aber in der aktuellen Werkzeugkette nicht verwirklicht.

terne Variable `NegWheelSlip` eingeführt. Diese Variable speichert das Ergebnis des `WheelSlip`-DFD-Blocks und wird für die Berechnung von `BelowLambda1` und `AboveLambda2` benötigt.

Jeder externe Port eines Clusters wird in eine IPC-Nachricht übersetzt. Im Beispiel wird der Cluster `Slip-Classification_RX` in das Modul aus Abb. 13 übersetzt. CCDs werden noch nicht explizit durch die grafischen AutoFocus-Editoren unterstützt. Daher entsprechen Cluster speziellen AutoFocus-Komponenten, die durch den Stereotype ≪`cluster`≫ markiert wurden (dieser ist in den Abbildungen nicht sichtbar). Im Beispiel werden eine „Receive"-IPC-Nachricht für den Wert von „wheel slip" und zwei „Send"-IPC-Nachrichten des ASCET-Typs `logic` benötigt.

Der Algorithmus für die Erstellung einer Clusterung wird ebenso wie weitere Modelltransformationen in AutoFocus mit Hilfe der sogenannten operation definition language (ODL) [25] angegeben. ODL ist eine logikbasierte Sprache, basierend auf Funktionen und Prädikaten, die für die Definition von Konsistenzchecks und Transformationen benutzt werden kann. Ein ODL-Ausdruck kann Benutzerinteraktionen beinhalten. Zum Beispiel fragt der Clusterungs-Algorithmus den Benutzer nach einer Clock und einer Teilmenge von Komponenten, die diesen Clock benutzen. Basierend auf diesen Informationen transformiert der Algorithmus das AutoFocus-Modell, so dass anschließend die CCD-Cluster enthalten sind und

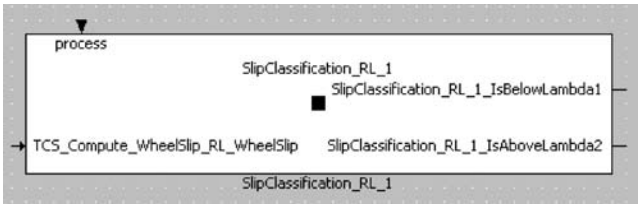

Abb. 13 ASCET-Modul `SlipClassification_RL`

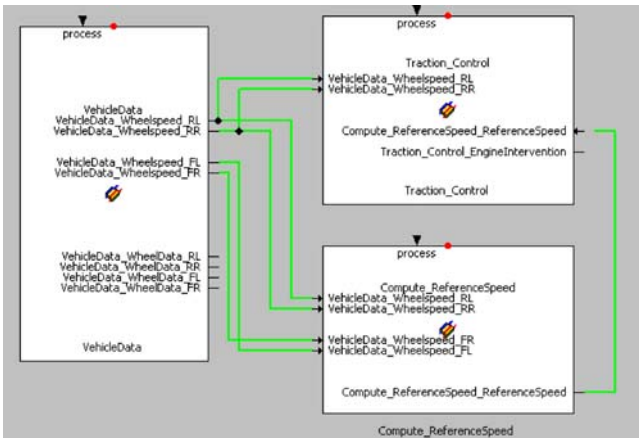

Abb. 14 ASCET-Module, die drei AutoFocus-Clustern entsprechen

Kommunikationsbeziehungen entsprechend umstrukturiert wurden.

Cluster-Definition. Bei der Übersetzung für die Cluster `VehicleData`, `Traction_Control` und `Compute_ReferenceSpeed` werden drei ASCET-Module (siehe Abb. 14) erzeugt. Im Beispiel sind zudem die Konnektoren zwischen den ASCET-Modulen angeben. Diese Konnektoren repräsentieren ASCET-Nachrichten.

Im nächsten Verfeinerungsschritt wird die Gruppierung von ASCET-Modulen in Projekte angegeben. Module, die in einem Projekt gruppiert wurden, werden auf einer ECU ausgeführt. Die Gruppierung von Modulen zu Projekten folgt daher der Abbildung von Clustern auf der LA-Ebene auf ECUs der TA-Ebene. Ist eine entsprechende Abbildung vorgegeben, so kann die Verteilung der Module auf ASCET-Projekte automatisiert werden. Die AutoMoDe-Werkzeugkette betrachtet jedoch diese Abbildung nicht.

Software-System-Konstruktion. Nachdem alle ASCET-Module in ASCET-Projekte gruppiert wurden, wird der ASCET-Code-Generator benutzt, um C-Code als Eingabe für INTECRIO zu generieren. Für jeden Cluster, also für jedes ASCET-Modul, wird dazu C-Code und eine Beschreibung im SCOOP-IX-Format generiert. Zusammengenommen repräsentieren diese ein INTECRIO-Modul (nicht zu verwechseln mit einem ASCET-Modul). Für die Software-System-Konstruktion in INTECRIO werden alle Cluster als INTECRIO-Module in INTECRIO importiert. Anschließend könnten diese weiter mit den INTECRIO-Funktionalitäten geclustert werden. Das Ergebnis aller Clustering-Schritte ist in Abb. 15 zu sehen. Dabei sind Sensor- und Aktuator-Module am linken und rechten Rand des INTECRIO-Diagramms abgebildet. Die Ports am Rand des Diagramms stellen die Schnittstelle der Module zu den I/O-Boards des ES 1000 Rapid-Prototyping-Systems dar.

Target-Integration. Das ETAS-Rapid-Development-System ES 1000 ist eine VME-Bus-basierte Prototyping-Hardware. Für das Beispiel der Antischlupfregelung wurde ein Microprozessor-Board (ES 1135), ein A/D-Konverter-Board (ES 1303) und ein PWM-Board (ES 1330) eingesetzt. Die Software-Schnittstelle des A/D-Konverter-Boards in Form eines INTECRIO-Moduls ist die Grundlage für die Anbindung der PWM-Signale an den Drosselklappenstellmotor und die Bremszylinderansteuerung.

OS-Konfiguration. Auf der Ebene der AutoFocus-Modelle besteht die Antischlupfregelung aus Nachrichtenströmen, die mit unterschiedlichen Clocks getaktet sind. Aus der modellbasierten Sicht läuft die Komponente für die Drosselklappenansteuerung sechs mal so schnell wie die Komponente für das Antiblockiersystem und zwölf mal so schnell wie die Komponente für die Antischlupfregelung. Dieses auf Clocks basierende Modell wird in ein echtzeitfähiges System übersetzt. Dabei wird die Drosselklappenansteuerung im 1 ms Raster ausgeführt. Aus der

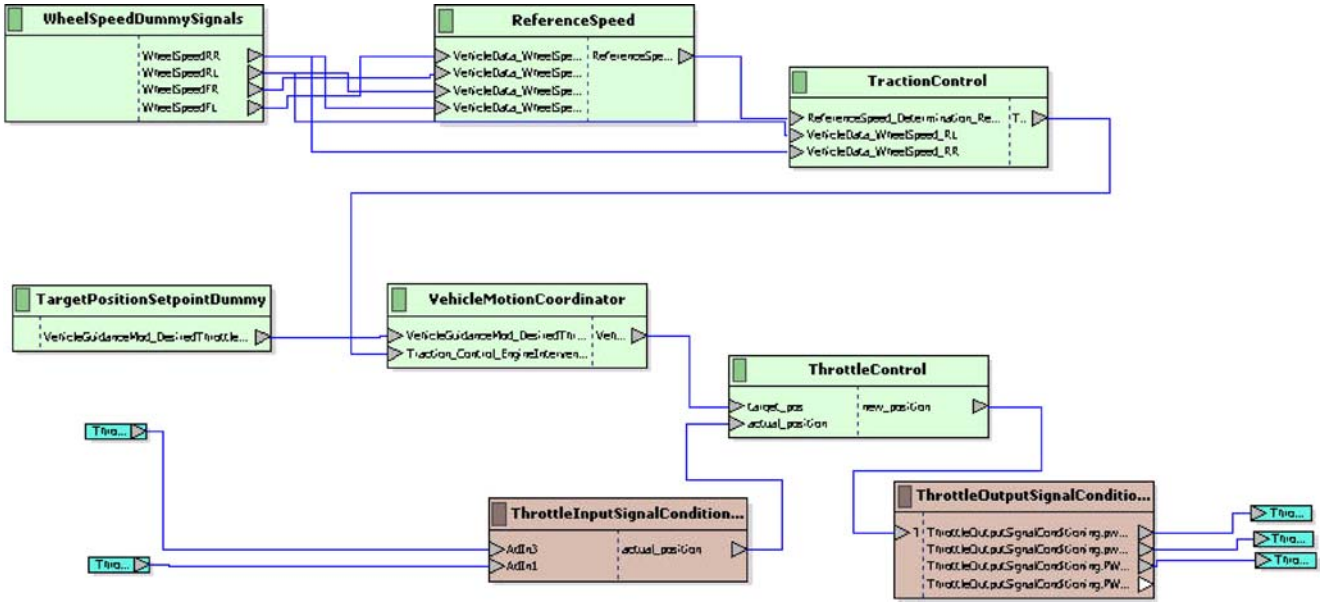

Abb. 15 Die Antischlupfregelung in INTECRIO

Kombination mit den Clocks des AutoFocus-Modells ergeben sich dann also Tasks im 1 ms, 6 ms und 12 ms Raster. Die Prozesse der INTECRIO-Module der einzelnen Komponenten werden dann den entsprechenden Tasks zugeordnet.

Wie in Abschn. 5 besprochen, beschreibt [4] ein „correct-by-construction"-Verfahren zur Umsetzung von zeitsynchronen AutoFocus-Modellen basierend auf einem ratenmonotonen Scheduling. Der Ansatz nutzt die oben beschriebene Doppel-Puffer-Technik für die Kommunikation von niederfrequenten zu hochfrequenten Tasks. Für die Analyse dieser Default-Konfiguration kann die Planner-Funktionalität des Werkzeugs RTA-OSEK [17] eingesetzt werden, das auf in [27] beschriebenen Algorithmen basiert.

Im dem Fall, dass der einfache ratenmonotone top-down Ansatz aus [4] nicht ausreicht, können die Algorithmen aus [3] eingesetzt werden, um bottom-up sicherzustellen, dass ein Multi-Clock-System ordnungsgemäß durch ein gegebenes Scheduling umgesetzt wurde. Die Grundidee der Algorithmen ist die Überprüfung, dass alle Signale im entsprechenden Zyklus gelesen werden und keine schreibende Komponente durch eine lesende Komponente überholt wird. Über den Gebrauch derartiger Checkalgorithmen hinaus ist es derzeit die gängige Praxis, dass Scheduling-Analysen auf der Messung der Ausführungszeiten beruhen [26].

8 Zusammenfassung und Ausblick

Der hier abschließend vorgestellte AutoMoDe-Ansatz zur modellbasierten Entwicklung softwareintensiver Systeme im Automobil basiert auf einem integrierten Domänenmodell mit alternativen Sichten, Notationen und einem einheitlichen Berechnungsmodell. Der Entwicklungsprozess für Automotive-Software wird dabei auf verschiedenen Abstraktionsebenen unterstützt, die jeweils für gegebene methodische Zwecke und Entwicklungsphasen maßgeschneidert sind. Gegenüber vorausgegangenen Arbeiten im AutoFocus-Umfeld wurden spezifisch für die Domäne Automotive neue Beschreibungstechniken eingeführt, insbesondere zur Darstellung von regelungstechnischem Datenfluss, zur expliziten Modellierung von Betriebsmodi, sowie zur Unterstützung später Entwicklungsphasen in Hinblick auf eine Implementierung von Modellen als Echtzeitsysteme.

Als wichtige methodische Ergänzung zu den reinen Notationen wurden in diesem Artikel eine Reihe von Transformationen innerhalb sowie zwischen Abstraktionsebenen diskutiert. Einige der angesprochenen Transformationen sind bereits im AutoFocus2-Werkzeugprototypen implementiert und automatisiert, wie z.B. die im vorigen Abschnitt beschriebene Sequenzialisierung von Modellen, die Übersetzung nach ASCET, oder die OS-Konfiguration. Andere hingegen, wie die Modulidentifizierung oder Cluster-Definition verlangen ein größeres Maß an Benutzerinteraktion, da hier bewusst Freiheitsgrade ausgenützt werden können, die aus der konkreten Anwendung heraus motiviert sind. Dank der ODL ist es jedoch möglich, auch solche Schritte bei z.B. häufig wiederkehrenden Abläufen zu automatisieren.

Die in AutoMoDe entwickelten Konzepte wurden darüber hinaus in vielen Punkten mit den aktuellen Entwicklungen der AUTOSAR-Entwicklungspartnerschaft [24] abgestimmt, im Hinblick auf eine bessere Übertragbarkeit von

Ergebnissen in die Praxis mit der anstehenden Einführung von AUTOSAR in die Serienentwicklung bei den beteiligten Industriepartnern.

Der hier beschriebene Ansatz wurde mit einer umfangreichen Fallstudie aus dem Bereich der Fahrdynamiksteuerung abschließend erprobt und auf allen Abstraktionsebenen validiert. Im Ergebnis entstand auf konkreter Implementierungsebene ein Demonstrator, der die im abstrakten Modell definierte Funktionalität des Softwaresystems als konkrete Echtzeitanwendung mit physikalischen Sensoren und Aktuatoren umsetzt. Somit konnte gezeigt werden, dass in AutoMoDe durch einen phasenübergreifenden, homogenen Ansatz unter Einsatz von Transformationstechniken und Konsistenzprüfungen auch anspruchsvolle Softwareanwendungen effektiv, konsistent, und mit engem Bezug zu kommerziell etablierten Werkzeugketten entwickelt werden können.

Literatur

1. Das Projekt EAST-EEA – Eine middlewarebasierte Softwarearchitektur für vernetzte Kfz-Steuergeräte. In: VDI-Kongress Elektronik im Kraftfahrzeug. Number 1789 in VDI Berichte. Baden-Baden, 2003
2. Amálio N, Polack F (2003) Comparison of formalisation approaches of UML class constructs in Z and Object-Z. In: ZB 2003. volume 2651 of LNCS. Springer
3. Baleani M, Ferrari A, Mangeruca L, Sangiovanni-Vincentelli AL, Freund U, Schlenker E, Wolff HJ (2005) Correct by construction transformations across design environments for model-based embedded software development. In: DATE 05
4. Bauer A, Romberg J (2004) Model-based deployment in automotive embedded software. In: MOMPES 2004
5. Bauer A, Romberg J, Schätz B (2005) Integrierte Entwicklung von Automotive-Software mit AutoFocus. Informatik Forsch Entw 19(4):194–205
6. Benveniste A, Caspi P, Edwards S, Halbwachs N, Guernic PL, Simone RD (2003) The Synchronous Languages Twelve Years Later. Proc IEEE 91(1):64–83
7. Benveniste A, Caspi P, Guernic PL, Halbwachs N (1993) Data-Flow Synchronous Languages. In: REX School/Symposium pp 1–45
8. Braun P, Lötzbeyer H, Schätz B, Slotosch O (2000) Consistent integration of formal methods. In: TACAS 2000 number LNCS 2280. Springer
9. Broy M, Huber F, Schätz B (1999) AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. Informatik Forsch Entw 14(3):121–134
10. Broy M, Stølen K (2001) Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement. Springer
11. Damm W (2006) Embedded system development for automotive applications: trends and challenges. In: EMSOFT. ACM. ed by Min SL, Yi W
12. ETAS GmbH (2001) ASCET-SD Benutzerhandbuch
13. ETAS GmbH (2005) INTECRIO User Guide V 1.0
14. France R, Evans A, Lano K, Rumpe B (1998) The UML as a formal modeling notation. Comput Stand Interf 19(7):325–334
15. Giese H, Burmester S, Schäfer W, Oberschelp O (2004) Modular design and verification of component-based mechatronic systems with online-reconfiguration. In: FSE-12. ACM
16. Huber F, Schätz B, Einert G (1997) Consistent Graphical Specification of Distributed Systems. In: FME'97, LNCS 1313. Springer
17. LiveDevices York (2005) RTA-OSEK User Guide V 4.0
18. Maraninchi F, Raymond Y (2003) Mode-automata: a new domain-specific construct for the development of safe critical systems. Sci Comput Program 46(3):219–254
19. The MathWorks Inc. (2000) Using Simulink
20. Müller-Glaser KD, Frick G, Sax E, Kühl M (2004) Multiparadigm Modeling in Embedded Systems Design. IEEE Trans Control Syst Technol 12(2):279–292, March
21. Mutz M, Huhn M, Goltz U, Krömke C (2003) Model based system development in automotive. In: SAE World Congress
22. Poledna S, Mocken T, Scheimann J, Beck T (1995) Ercos: An operationg system for automotive applications. In: SAE World Congress
23. Romberg J (2006) Synthesis of distributed systems from synchronous dataflow programs. PhD thesis TU-München
24. Scharnhorst T, Heinecke H, Schnelle KP, Fennel H, Bortolazzi J, Lundh L, Heitkämper P, Leflour J, Mate J, Nishikawa K (2005) Autosar – challenges and achievements. In: Elektronik im Kraftfahrzeug 2005. VDI, October
25. Schätz B, Braun P, Huber F, Wisspeintner A (2005) Checking and transforming models with AutoFOCUS. In: ECBS 2005. IEEE
26. Schäuffele J Zurawka T (2003) Automotive Software Engineering. Vieweg Verlag, Wiesbaden
27. Tindell K, Clark J (1994) Holistic schedulability analysis for distributed hard real-time systems. Eur J 40:117–134
28. von der Beeck M, Braun P, Rappl M, Schröder C (2003) Automotive UML. In: Selic B, Martin G, Lavagno L (eds), UML for Real: Design of Embedded Real-Time Systems, number ISBN 1-4020-7501-4. Kluwer Academic Publishers