Informatik Forschung und Entwicklung manuscript No.
(will be inserted by the editor)

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

Das AutoMoDe-Projekt

Modellbasierte Entwicklung softwareintensiver Systeme im Automobil

14. August 2006

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 automobiler 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, D-85748 Garching b. München

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

U. Freund, N. Mata
ETAS GmbH
Borsigstr. 14, D-70469 Stuttgart

R. Sandner
BMW AG
D-80788 München

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

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

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

Die Komplexität stellt somit hohe Anforderungen an eine Entwicklungsmethodik für Automotive-Software. 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 von Funktionsmodulen erreicht werden. Über die genannten Punkte hinaus 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 abschließend 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 Ansatz wird unter anderem durch das AutoFOCUS-Werkzeug [7] unterstützt. Konkret bietet AutoFOCUS dem Entwickler eine Anzahl graphischer und textueller Beschreibungstechniken für verschiedene Abstraktionsebenen (siehe Abschnitt 3.1). Damit ist die Modellierung von Struktur und Verhalten von Software durchgängig möglich: angefangen von der Erfassung funktionaler Abhängigkeiten, bis hin zur Verteilung von Applikationen auf Prozesse, Tasks und reale ECUs im Bordnetzverbund des Fahrzeugs. Zusätzlich existieren eine Reihe von Anbindungen für Techniken der Konsistenzanalyse, formalen Verifikation sowie Testfallgenerierung [6], sowie Code-Generatoren für die Programmiersprachen C und Ada.

Um die praktische Umsetzung der Konzepte zu überprüfen wurde eine Werkzeugkette bestehend aus etablierten Werkzeugen der ETAS aufgebaut. Anhand dieser konnte gezeigt werden, dass AutoFOCUS-Modelle in effiziente Implementierungen für eingebettete Hardware umgesetzt werden können.

Aufbau. Abschnitt 2 diskutiert verwandte Arbeiten zu den in AutoMoDe geleisteten Beiträgen. In Abschnitt 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 Abschnitt 5 die eingesetzten AutoFOCUS-Notationen beschrieben werden. Als wesentlicher Bestandteil von AutoMoDe werden in Abschnitt 6 verschiedene Transformationen im Entwicklungsprozess beschrieben. Das Beispiel der Modellierung einer Antischlupfregelung in Abschnitt 7 dient dazu die Ergebnisse zu veranschaulichen und deren praktische Umsetzung zu überprüfen. Der Übergang zu der Implementierung des Beispiels ist Gegenstand des Abschnitts 8. Der Stand der Werkzeugunterstützung des AutoFOCUS-Werkzeugs wird in Abschnitt 9 kurz diskutiert, bevor dieser Artikel mit einer Zusammenfassung endet.

2 Verwandte Ansätze und Beiträge von AutoMoDe

Der Beitrag von AutoMoDe kann wie folgt in zwei Teilaspekte aufgespalten werden: Zum einen wurde als Ergebnis des Projekts ein *methodisches* Rahmenkonzept für die Entwicklung automobiler 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.

Methodischer Beitrag. Es existieren eine Reihe von verwandten Methoden für modellbasierten Entwurf automobiler Softwaresysteme [16, 17, 1, 27]. Neben Unterschieden im Detail verwenden alle zitierten Ansätze eine Anzahl von definierten Abstraktionsebenen sowie verschiedenen unterstützenden Werkzeugen und Notationen, mit dem Ziel, einer inkrementellen Entwurfsmethodik für automobile Software. Durch die Verwendung heterogener Notationen und Werkzeuge im Stadium des Entwurfs gelingt diesen vorangegangenen Ansätzen jedoch typischerweise keine enge Kopplung von Syntax- und Semantikkonzepten über Werkzeug-, Notations-, sowie Phasengrenzen hinweg: diese enge Kopplung ist in einem heterogenen Ansatz naturgemäß schwierig. AutoMoDe verwendet dagegen ein einheitliches und semantisch fundiertes Domänenmodell, das durch das AutoFOCUS-Werkzeug unterstützt wird [7]. Durch die Homogenität des Domänenmodells können neuartige technische Beiträge wie die semantikerhaltende Transformation von Modellen durch Transformationssprachen, eingebracht werden.

Von den genannten Ansätzen stützt sich AutoMoDe speziell auf Vorgängerprojekte Automotive [27] 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. 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 je-

doch 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 [20] konnte AutoMoDe zudem von den technischen Ergebnissen her auf die „Virtual Functional Bus"-Architektur von AUTOSAR abgestimmt werden.

Technischer Beitrag. Gegenüber regelungstechnisch motivierten Werkzeugen wie Matlab/Simulink [15] oder ASCET [9] sowie aus anderen Anwendungsgebieten (z. B. Telekommunikation) stammenden Ansätzen wie UML 2.0 Komponentendiagrammen [24] bietet der AutoFOCUS-gestützte Ansatz den Vorteil von domänenspezifische Konzepten wie z. B. Betriebsmodus, Funktion, 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. [14]. 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 Boolsche Ausdrücke (Clocks) zu definieren, sowie ein flankierendes Verfahren zur Inferenz und Überprüfung von Clocks, stammt aus dem Gebiet der synchronen Datenflusssprachen [4]. Das entworfene Inferenzverfahren unterscheidet sich in Details von den bekannten Verfahren, um die Skalierbarkeit des Verfahrens sowie die Anschaulichkeit der inferierten Clocks mit einer ausreichenden Ausdrucksmächtigkeit der Modellierungssprache zu verknüpfen.

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, die Editieren, Analyse, sowie Transformation von Modellen auf verschiedenen Abstraktionsebenen zulässt.

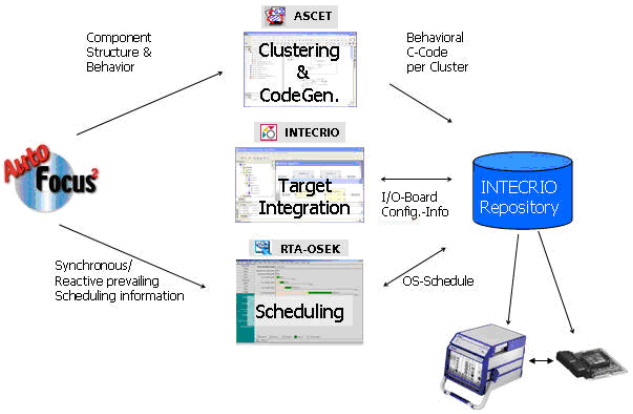

Abb. 1 AutoMoDe-Werkzeugkette

Die AutoMoDe-Entwurfsmethodik basiert stark auf einer einer Gliederung der Modellierungsartefakte in domänenspezifische Abstraktionsebenen. Diese sind an die in [1] vorgeschlagenen angelehnt (siehe Abb. 2). Zur Veranschaulichung des Gesamtansatzes werden für jede Abstraktionsebene jeweils die AutoMoDe-Werkzeuge erwähnt, die primär zur Repräsentation der jeweiligen Entwurfsartefakte eingesetzt werden. Die gesamte AutoMoDe-Werkzeugkette ist in Abb. 1 illustriert (siehe Abschnitt Abschnitt 8). Der Zusammenhang von Werkzeugen zu Abstraktionsebenen ist unkompliziert: Für die abstrakteren Modellierungsebenen wird das AutoFOCUS-Werkzeug verwendet, das die in Abschnitt 4 beschriebenen Notationen für die abstrakteren Ebenen unterstützt. Nebem dem Editieren und Validieren der so entstandenen Modelle unterstützt AutoFOCUS verschiedene Arten von Modelltransformationen, die in Abschnitt 6 näher erläutert werden. In Richtung auf die konkreteren und stärker implementierungsbezogenen Abstraktionsebenen schließt die AutoMoDe-Werkzeugkette die kommerziellen Werkzeuge ASCET [9], RTA OSEK Planner [13], sowie INTECRIO [10] mit ein. Die letztgenannten Werkzeuge zur implementierungsbezogenen Entwicklung und Synthese automobiler Softwaresysteme sind in dieser Domäne praktisch etabliert.

3.1 Abstraktionsebenen

Functional Analysis Architecture (FAA). Die abstrakteste der betrachteten Ebenen ist die Functional Analysis Architecture. Sie ist eine fahrzeugweite und bzgl. der Struktur

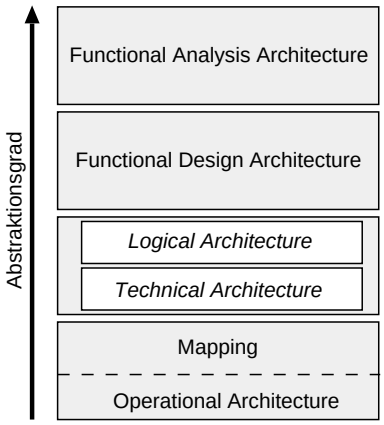

Abb. 2 AutoMoDe-Abstraktionsebenen

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. Die folgende Darstellung konzentriert sich zudem auf in Software realisierte Funktionalitäten, während die FAA insgesamt unabhängig von der Realisierungsentscheidung (HW/SW) ist.

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 die Längsdynamik beeinflussende Funktionen wie eine Traktionskontrolle. Im Vergleich zur FDA-Ebene kann dabei ein und dieselbe Komponente 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 Abschnitt 5 beschriebenen Systemstrukturdiagramme (SSDs) repräsentiert, und in untergeordneter Weise durch die AutoFOCUS-Notationen für detailliertes Verhalten. Mit AutoFOCUS können auch vielfältige, FAA-bezogene Konsistenzbedingungen überprüft und durchgesetzt werden: so kann z.B. die Strukturvollständigkeit auf FDA-Ebene als Vollständigkeit der Kommunikationsverbindungen interpretiert und automatisch überprüft werden: jeder eingehende Konnektor muss dabei z. B. mit einem Signal belegt sein. Verhaltensvollständigkeit bezeichnet in AutoFOCUS die Festlegung eines eindeutigen (deterministischen) Verhaltens für jede Komponente.

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 wiederverwendbare 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 auf FDA-Ebene Potenziale für Mehrfachverwendung aufgedeckt werden können, die sich beim Übergang von einer funktionsorientierten Strukturierung (FAA) auf eine an der Implementierung orientierten Struktur (LA/TA) ergeben. Ein häufig genanntes Beispiel für solche Potenziale 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 Abschnitt 5 beschriebenen Notationen zur Verhaltensdarstellung möglich. Die Überprüfung FDA-spezifischer Konsistenzbedingungen wie z. B. Verhaltensvollständigkeit kann ebenfalls durch AutoFOCUS vorgenommen werden.

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 sog. Cluster gruppiert: dabei repräsentiert ein Cluster eine kleinste verteilbare Einheit, z. B.

im Sinne einer Task 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*, 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 den Werkzeugen 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 [8]. 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, 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.

Modellierung von Echtzeitverhalten. Um die heterogenen (a)periodischen Takte bzw. Frequenzen typischer Automotive-Anwendungen komfortabel modellieren zu können, werden in AutoFOCUS sog. *Clocks* [5] verwendet. Jeder Nachrichtenstrom in AutoFOCUS ist mit einer Clock assoziiert: Die Clock ist dabei ein Boolsche Ausdruck, welcher die Anwesenheit bzw. Abwesenheit einer Nachricht eines Nachrichtenstrom beschreibt. Zur Laufzeit evaluiert der Ausdruck zu logisch wahr wenn der 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. Ähnlich zu anderen visuellen Architekturbeschreibungssprachen oder der UML-RT wird das System dabei als Netzwerk von *Komponenten* 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.

Die Strukturierung auf SSD-Ebene hat dabei auch semantische Auswirkungen: Kommunikation zwischen SSD-Komponenten erfolgt mit einer Verzögerung um eine Periode der jeweiligen Clock. Durch diese Festlegung werden bereits auf hoher Abstraktionsebene “Sollbruchstellen” für die spätere Partitionierung im Rahmen des Deployments eingeführt, ohne diese Partitionierung im Detail vorwegzunehmen.

Abb. 3 zeigt ein typisches SSD für die Darstellung der in Abschnitt 7 beschriebenen Antischlupfregelung. 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, wo ein möglicher Koordinationsbedarf zwischen den (funktional eng verwandten) Anfragen `DesiredThrottlePositionDriver` und `EngineIntervention` offensichtlich wird.

Für die Verwendung der SSD-Notation auf FDA-Ebene kann durch die in Abschnitt 9.2 beschrieben Konsistenzprüfungen die Struktur- und Verhaltensvollständigkeit für solche Modelle auf FDA-Ebene überprüft und sichergestellt werden.

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). Datenflussdiagramme beschreiben den algorithmischen Datenfluss von Komponenten und werden typischerweise auf allen Abstraktionsebenen eingesetzt. Auf FAA-Ebene überwiegen dabei prototypische Verhaltensbeschreibungen. DFDs selbst sind wiederum 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), ein State Transition Diagram (STD), oder direkt durch einen textuellen Ausdruck in der Basissprache von AutoFOCUS [12] gegeben sein. Hierarchische Blöcke werden wiederum durch DFDs definiert. Im Gegensatz zu SSDs, die eine präzise statische Typisierung an der Portschnittstelle vorschreiben, ist die Typisierung von Ports in DFDs dynamisch: elementare Blöcke wie z. B. arithmetische Operatoren schränken lediglich die Typklasse ein, aber legen nicht den genauen Datentyp eines Ports fest.

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.

Im Gegensatz zu der verzögerten Komposition in SSDs ist die Semantik der Komposition in DFDs grundsätzlich unverzögert im Sinne der synchronen Sprachen [4]. Im Auto-

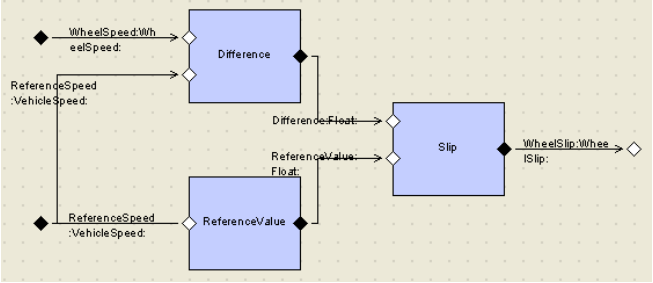

Abb. 4 Beispiel für ein DFD zur Radschlupferkennung

FOCUS-Werkzeug wird die Verwendung von unverzögerter Kommunikation von einer Kausalitätsprüfung begleitet, die Modelle mit unverzögerte Schleifen ablehnt, da diese nicht kausal implementierbar sind (d. h. die zugrundeliegende Berechnung hat keinen Fixpunkt). Zur Verwendung "unverzögerter" Kommunikation auf Modellebene ist zu bemerken, dass die Modellierung gleichzeitiger Aktionen im Modell auf FAA-, FDA- und LA-Ebene eine durchaus valide Abstraktion von sequenziell geordneten, zeitbehafteten Berechnungen und Nachrichtentransfers auf der Ebene der OA darstellt. Die Berechnungen einer Komponente werden durch die SSD-Festlegung spätestens an der Komponentengrenze mit einer Verzögerung beobachtet: diese Verzögerung definiert somit implizit die maximale Zeit (Deadline) für die Summe sequenzieller Berechnungen auf der OA-Ebene.

Mode-Transition-Diagramme (MTD). Die gegenüber [7] 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 graphische 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. Untergeordneten 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.

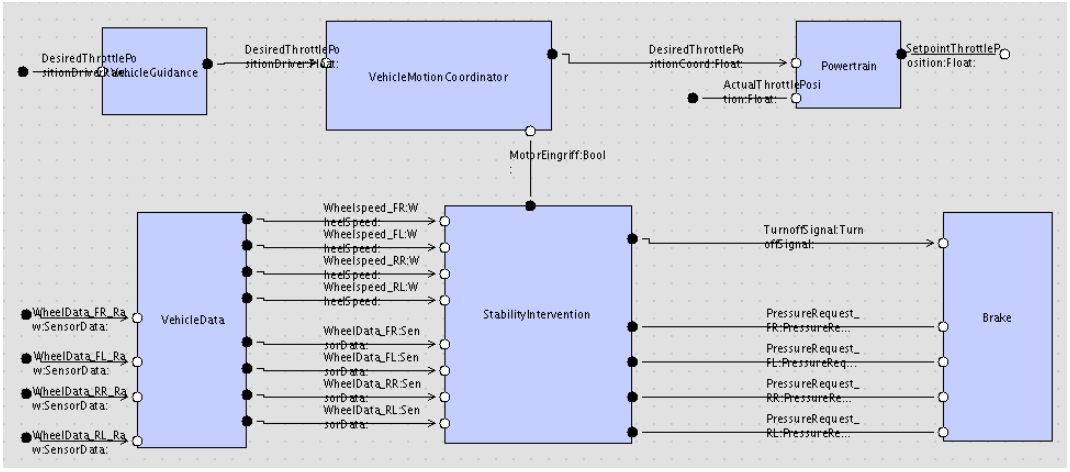

Abb. 3 Beispiel für ein SSD auf FAA-Ebene

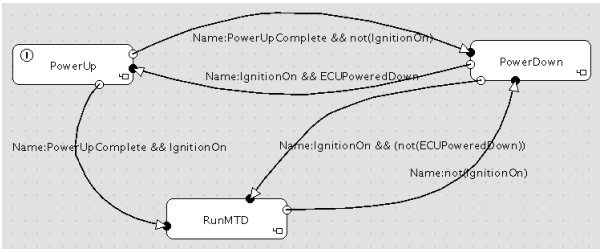

Abb. 5 Beispiel für ein MTD mit Betriebsmodi einer Motorsteuerung

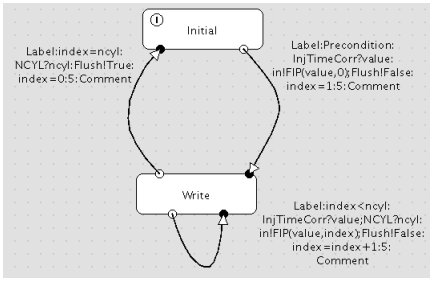

Abb. 6 Beispiel für ein STD zur Zündzeitpunktsteuerung für einen Benzinmotor

Abb. 5 zeigt ein Beispiel aus einer AutoMoDe-Fallstudie für die Modellierung von Betriebszuständen einer Motorsteuerung. Dabei wird zwischen den Modi `PowerDown` (ausgeschalteter Zustand), `PowerUp` (Hochlauf der Steuerung) sowie `RunMTD` (betriebsfähiger Zustand) unterschieden und je nach Wertbelegung der Boolschen Eingänge `PowerUpComplete` (Beendigung der Hochlaufphase) und `IgnitionOn` (Stellung des Zündschlosses) zwischen den Modi umgeschaltet.

State-Transition-Diagramme (STD). State Transition Diagrams sind erweiterte endliche Zustandsautomaten mit Zuständen und Transitionen zwischen Zuständen (siehe Abb. 6). Transitionen werden anhand von Bedingungen über den aktuellen Werten der Eingabeports der STD-Komponente sowie ihrer lokalen Variablen ausgewählt. Auf jeder Transition können im Falle der Aktivierung neue Werte für die Ausgabeports der STD-Komponente sowie neue Werte für die lokalen STDs geschrieben werden. STDs sind somit ähnlich zu den bekannten Statecharts [11], sind aber diesen gegenüber mit bestimmten syntaktischen Einschränkungen verbunden. So gibt es in STDs z. B. keine AND-Zustände, keine Inter-Level-Transitionen, und eingeschränkte Möglichkeit der Abbruchbehandlung (Präemption). Durch die gewählten Einschränkungen werden einige der semantischen Zweideutigkeiten, die bei vielen Statecharts-ähnlichen Sprachen auftreten [26], vermieden.

Obschon sowohl MTDs als auch STDs Kontrollfluss definieren, und auf den ersten Blick sehr ähnlich wirken, unterliegen die beiden Notationen verschiedenen syntaktischen Beschränkungen, und passen auf voneinander verschiedene methodische Einsatzbereiche. In STDs kann auf Ausgabeports und lokale Variablen geschrieben werden, während MTDs zwischen verschiedenen untergeordneten Verhaltensdefinitionen hin- und herschalten, die wiederum die Werte der Ausgabeports sowie der lokalen Variablen bestimmen. Nach ersten praktischen Erfahrungen ist die STD-Darstellung besser für sporadisches, ereignisgesteurtes Verhalten auf Subsystemebene geeignet, während MTDs besser zu umgeschalteten periodischen Berechungen passen. Im Gegensatz zu MTDs sind STDs im Allgemeinen nicht dafür geeignet, eine Grobstrukturierung größerer Systemmodelle durchzuführen.

Cluster-Communication-Diagramme (CCD). Cluster Communication Diagrams werden auf der obersten Hierarchieebene der Logical Architecture 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 ist nie auf

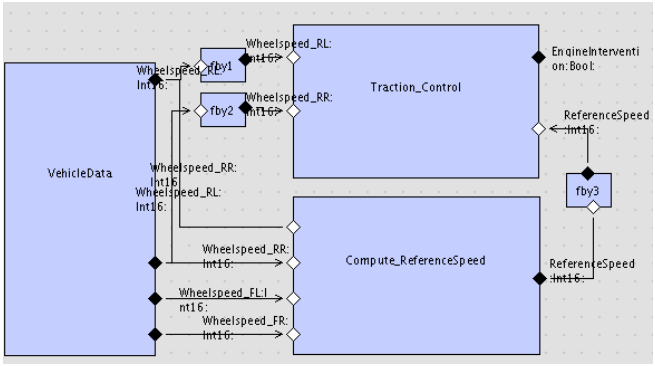

Abb. 7 Beispiel für ein CCD für eine Antischlupfregelung

mehrere Tasks einer Applikation verteilt. Die Partitionierung der FDA-Komponenten in Cluster kann sich dabei beispielsweise an den Clocks bzw. Frequenzen des Designs orientieren. Je nach Scheduling-Strategie [19] 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 graphische 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.

Abb. 7 zeigt ein Beispiel für ein CCD für eine Antischlupfregelung.

Das Typsystem wird in AutoMoDe für die LA-Ebene durch ein Konzept für Implementierungstypen erweitert, das die mehr oder weniger stark ausgeprägten plattformbezogenen Eigenschaften des Modells in Bezug auf die Implementierung abbildet. Prinzipiell ist ein Implementierungstyp 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, wie z. B. Überlauf von Variablen, können mit Hilfe des AutoFOCUS-Werkzeugs in Simulationen validiert 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 [19] 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) [18] 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 [19] wird gezeigt, dass unverzögerte Kommunikation auf Modellebene nur in speziellen Situationen durch Inter-Task-Kommunikation realisiert werden kann: Dies ist der Grund für die oben beschriebenen Einschränkungen für Verwendung unverzögerter Kommunikation in CCDs.

6 Transformationen und Übergänge

Für die Modellierung von Systemen innerhalb einzelner Abstraktionsebenen stellt AutoFOCUS Funktionalität zum Modellieren, zur dynamischen Validierung (Simulation), sowie zur statischen Validierung (z. B. Vollständigkeit der Kommunikationsbeziehungen, Typkorrektheit, Aufdecken kausaler Zyklen, Clock-Checks) zur Verfügung. Als wesentliche Anforderung für die methodische Eignung des Werkzeugs ist jedoch außerdem die Unterstützung für den *Übergang zwischen Modellen verschiedener Abstraktionsebenen* sowie den *Übergang zwischen Modellen innerhalb einer Abstraktionsebene* zu nennen.

Grundsätzlich betrachtet, wird im ersteren Fall ein vorhandenes Modell des Systems mit Informationen angereichert, und es werden z. B. Umstrukturierungen aufgrund einer zunehmenden Orientierung an technischen Randbedingungen der Implementierung vorgenommen. Im zweiteren Fall steht die reine Umstrukturierung im Vordergrund. Solche Übergänge sind oft nichttrivial und nach heutigem Kenntnisstand nur begrenzt automatisierbar. Die Anforderung an den vorliegenden Ansatz ist es deshalb, zumindest diejenigen Tätigkeiten zu identifizieren, die anhand der in AutoFOCUS-Modellen vorliegenden Informationen sinnvoll formuliert und im Rahmen eines Werkzeugs zur Verfügung gestellt werden können.

6.1 Übergang zwischen Modellen verschiedener Abstraktionsebenen

Da AutoFOCUS insbesondere auf die Unterstützung der oberen und mittleren Architekturebenen ausgerichtet ist, betrachten wir dabei zwei Übergänge:

– Übergang *FAA→FDA*. Hier steht die Abbildung von Funktionalitäten auf logische Komponenten im Vordergrund. Dabei sollten zum einen Mechanismen vorhanden sein, welche die Umstrukturierung von funktional

gruppierten nach architekturmäßig gruppierten Strukturdarstellungen erleichtern. Zum anderen wird die Identifikation und Auflösung von Konflikten betrachtet, welche durch die Integration von Teilsystemen auf FDA-Ebene entstehen, deren Verhalten auf FAA-Ebene jeweils isoliert bzw. weniger detailliert betrachtet wurde.

– Übergang *FDA→LA/TA*. Für den Übergang von FDA auf die LA-Darstellung, die zusammen mit der (separat zu entwickelnden) TA die LA/TA-Abstraktionsebene beschreibt, muss eine Abbildung bzw. Umstrukturierung von FDA-Komponenten auf LA-Cluster vorgenommen werden. Die Zielebene LA/TA spiegelt die Einschränkungen aus der verteilten Implementierung dabei in höherem Maße wieder als die FDA. In den AutoFOCUS-Notationen entspricht dieser Übergang dem von hierarchischen SSDs und DFDs auf „flache“ CCDs, wobei die Cluster die kleinsten verteilbaren Einheiten eines Software-Systems darstellen.

Übergang FAA→FDA. Zentrale Aufgabe beim Übergang von der funktionalen Architektur der FAA zur logischen Architektur der FDA ist die Integration von einzelnen, teilweise isolierten Funktionalitäten zu logischen Komponenten. AutoFOCUS bietet dazu Mechanismen zur Integration ebenso auf der Struktur- ebenso wie auf der Verhaltensebene an.

Wie in Abschnitt 3.1 gezeigt, lassen sich Funktionalitäten und Komponenten mit den gleichen Beschreibungsmitteln darstellen; die Funktionalitäten (z. B. Bremsfunktion) entsprechen dabei eher Komponententypen, die logischen Komponenten (z. B. Bremsfunktion des rechten Vorderrads) eher Komponenteninstanzen. Entsprechend bietet AutoFOCUS für eine Integration auf der Strukturebene die Möglichkeit, zwischen diesen Komponentenformen zu unterscheiden, sowie diese jeweils ineinander umzuwandeln.

Daneben unterstützt AutoFOCUS Mechanismen zum gezielten Umbau (*Refactoring*) von der FAA- zur FDA-Ebene. Neben generischen Schritten (z. B. Umordnen der Komponentenhierarchie inkl. Kanalanpassung), die auch in der FDA Anwendung finden – sind hier insbesondere spezifische Transformationen (z. B. Bündeln von Kanälen) denkbar.

Im Gegensatz zur Strukturintegration werden bei der Verhaltensintegration mehrere Funktionalitäten zu einer logischen Komponente „verschmolzen“. Diese Integration ist besonders bei der Kombination von Anwendungsfunktionen (z. B. Zentralverriegelung) mit Standardfunktionalität (z. B. Konfiguration von Parametern wie Dauer von Hup- oder Lichtsignal, Protokollierung von Fehlverhalten) sinnvoll. Wie in [22] beschrieben, werden dazu zu verschmelzende Elemente der einzelnen Funktionen vom Benutzer interaktiv identifiziert. Dabei können unterschiedliche Funktionsanteile wie Schnittstellenelemente, lokale Daten (z. B. Applikationsvariable „Lichtsignaldauer“ mit Parameter der Konfigurationsfunktion), und Zustände bzw. Modi (z. B. Modus „Betrieb“ der Zentralverrieglung mit Modus „Betrieb“ der Konfigurationsfunktion) verschmolzen werden. Elemente der zugebundenen Funktionen, die nicht zur Verschmelzung ausgewählt wurden (z. B. generisches Kommando zum Setzen eines Parameters), werden – unter Berücksichtigung der Verschmelzung (z. B. „Lichtsignaldauer“ mit generischem Konfigurationsparameter) – der kombinierten Komponente hinzugefügt (z. B. Kommando zum Setzen der Lichtsignaldauer).

Übergang FDA→LA/TA. AutoFOCUS schränkt den Entwickler grundsätzlich nicht hinsichtlich einer Abbildung von FDA-Komponenten auf LA-Cluster ein. Ausschlaggebend für die Formierung von Clustern können z. B. sein

– die vorhandenen Strukturgrenzen zwischen SSD-Komponenten der Ebene der FDA,
– die Platzierung von Delay- und Sampling-Operatoren auf FDA-Ebene, oder auch
– gemeinsame Signalfrequenzen innerhalb von Teilen des Modells mit dem Ziel, dass jeder Cluster innerhalb der LA eine möglichst einheitliche Clock/Frequenz aufweist.

In der Praxis wird es allerdings häufig eine Hybridstrategie geben, da z. B. die SSD-Grenzen in vielen Fällen mit den Frequenzübergängen zwischen Funktionen zusammenfallen können.

Beispiel (Delays). Im vorgestellten Ansatz ist festgelegt, dass zwischen zwei SSDs auf der Modellierungsebene FAA implizit stets ein Verzögerungsglied pro Kommunikationskanal liegen muss. Dies ist sinnvoll aus zwei Gründen:

1. Damit werden die SSDs zu logischen Einheiten mit vordefinierten Sollbruchstellen, den Unit Delays.
2. Im späteren Mapping auf Cluster ist an allen funktionalen Schnittstellen, die oft einem Sampling zwischen verschiedenen Frequenzen zusammenfallen, bereits eine Verzögerung entsprechend der Einschränkungen für CCDs vorhanden.

Insbesondere Punkt 2. ermöglicht es, dass Verzögerungen nicht implementierungsgetrieben, also bottom-up, im Design verteilt werden müssen, sondern dass unter Kenntniss der Ausführungsarchitektur (O/S, Scheduling) eine Middleware (bzw. deren Eigenschaften) generiert werden kann, die die verzögerte Semantik realisiert (siehe [3]). Ein nachträgliches Einfügen von Verzögerungen mit abschließender Revalidierung entfällt dadurch.

Beispiel (Typen). AutoFOCUS unterstützt generell zwei Arten von Typen: *abstrakte* Modellierungstypen und *Implementierungstypen*. Dies ist sinnvoll, da im Entwicklungsprozess besonders in den frühen Phasen häufig keine Festlegung auf eine Ausführungsarchitektur besteht, so dass eine Systemmodellierung auf abstrakten Datentypen ermöglicht werden muss. Zur Verfügung stehen beispielsweise `Int`,

`Float`, oder auch zusammengesetzte benutzerdefinierte Typen.

Der Übergang FDA→LA behandelt somit nicht nur die Aufteilung in Cluster, sondern zieht im Allgemeinen auch einen Übergang von abstrakten Datentypen zu konkreteren Implementierungstypen nach sich, z. B. `Int` → `Int8`.

6.2 Übergang innerhalb einer Abstraktionsebene

Die Einführung von MTDs (vgl. Abschnitt 5) und damit die explizite Modellierung von Modus-Informationen in AutoFOCUS-Designs eignet sich insbesondere zur Modelltransformation *innerhalb* einer Abstraktionsebene (Refactoring). MTDs werden typischerweise eingesetzt, um Betriebsmodi zu modellieren, eignen sich aber auch „bottom-up“, beispielsweise um Gemeinsamkeiten von Modellen zum besseren Verständnis und erhöhter Konsistenz unter einem Modusdiagramm zusammenzufasen.

Sind die Modi einer Komponente bzw. eines Systemteils „top-down“ vordefiniert, so lassen sich ideal Refactorings beispielsweise zur technischen Optimierung durchführen. Dazu ist in Abb. 8 ein abstrahiertes Beispiel aus der Karosserie-Elektronik dargestellt, welches typische Betriebsmodi unterscheidet: `Sleep`, `Awake`, `StarterEngaged`, `PowerOn`. Die Pfeile symbolisieren Übergänge zwischen den Modi. Abhängig vom aktiven Modus werden nun verschiedene Innenraumkomponenten, dargestellt als DFDs, zur Ausführung gebracht: während sich Beleuchtung und Schiebe-Hebe-Dach (`InteriorLight`, `SunRoof`) nur im angeschalteten Zustand bzw. während des Startvorgangs bedienen lassen, muss die Zentralverriegelung (`PowerLock`) aus sicherheitstechnischen Gründen zusätzlich während `Awake` aktiv sein. Im Sleep-Modus ist keiner der Verbraucher aktiv.

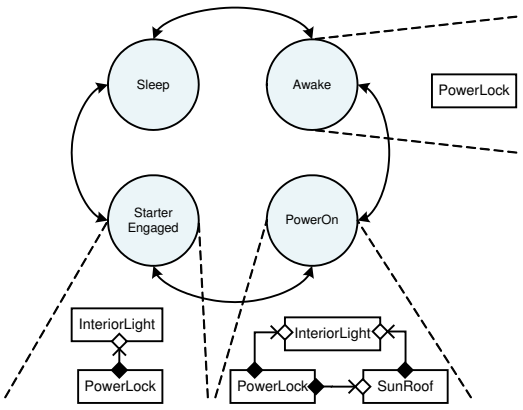

Abb. 8 Innenraum-Bedienelemente abhängig vom Betriebsmodus

Obwohl die Modus-Zuordnung im Beispiel durchaus sinnvoll ist, hätte eine naive Umsetzung solcher und ähnlicher Designs zur Folge, dass redundant modellierte Systemanteile wie beispielsweise `PowerLock` den Umfang einer Implementierung unnötig vergrößern würden. Aus Sicht der Modellierung kommt hinzu, dass `PowerLock` im engeren Sinn gar nicht nach vier bzw. drei Betriebszuständen unterscheiden müsste, sondern lediglich danach, ob sich die Innenraumelektronik im Energiesparmodus `Sleep` befindet, oder nicht.

Das in Abb. 9 dargestellte Refactoring greift beide Punkte auf, indem es einen Block bzw. Komponente `PowerLock` aus einem gegebenen MTD heraustrennt. Intern unterscheidet `PowerLock` dann lediglich zwischen zwei Zuständen. Die Information aus einer solcherart getrennten Darstellung kann dann dazu verwendet werden, `PowerLock` bei der Implementierung getrennt zu berücksichtigen, z.B. bezüglich des Energiemanagements. Die weiterhin bestehenden kausalen- und kommunikativen Abhängigkeiten zwischen den DFDs sind in der Darstellung durch die Ein- und Ausgabekanäle der neuen Komponenten dargestellt.

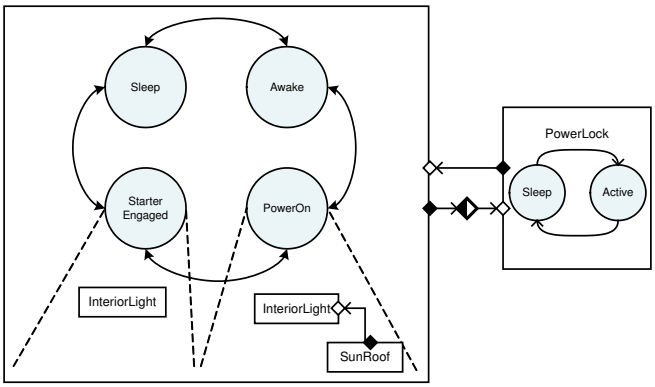

Abb. 9 Eine auf technische Umsetzung optimierte Modellierung des Systems aus Abb. 8

Solche Refactorings können neben technischen Gesichtspunkten auch einem anderen Zweck dienen, beispielsweise der Überführung einer eher kontrollflussorientierten Sicht wie in Abb. 8 (d. h. viele Komponenten ordnen sich genau einem Modusdiagramm unter), hin zu einer eher datenflussorientierten Darstellung ähnlich zu Abb. 9 (d. h. die einzelnen Komponenten unterscheiden individuell bzw. intern nach dem Modus – ohne ein übergeordnetes MTD).

Die dazu notwendigen Arbeitsschritte sind in AutoFOCUS einerseits durch die Editoren, andererseits durch die integrierte Logiksprache ODL (siehe Abschnitt 9.2) unterstützt, die es ermöglicht, solche Transformationsschritte abstrakt zu definieren und automatisch für verschiedene Modelle beliebig oft wiederzuverwenden.

7 Beispiel: Antischlupfregelung

Am Beispiel einer Antischlupfregelung (oder Traction Control System/TCS) wird im Folgenden der AutoMoDe-Ansatz

zur schrittweisen Entwicklung automobiler 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. Die Verhaltensbeschreibung der `StabilityIntervention`-Komponente aus Abb. 3 ist in Form eines DFDs in Abb. 10 zu sehen. Das DFD definiert 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 eigentliche Antischlupfregelung, der Brems-Schlupf-Regelung, der Brems-Verzögerungsregelung und einer Koordinierung der einzelnen Bremsanforderungen. Das Ergebnis der Koordinierung der Bremsanforderungen dient als Druckvorgabe für einzelnen Radbremszylinder. Die Abbildung zeigt zudem schematisch die Ausführungsfrequenz der verschiedenen Komponenten, die durch Clocks der entsprechenden Kanäle spezifiziert wurde (in der Abbildung sind diese Clocks nicht zu sehen).

Abb. 11 zeigt die hierarchische Dekomposition der `Traction_Control`-Komponente. Basierend auf den Radgeschwindigkeitssignalen wird eine Referenzgeschwindigkeit errechnet. Diese wird mit den einzelnen Radgeschwindigkeiten verglichen. Der daraus ermittelte Radschlupf wird mit der Referenzgeschwindigkeit normalisiert und klassifiziert. 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.

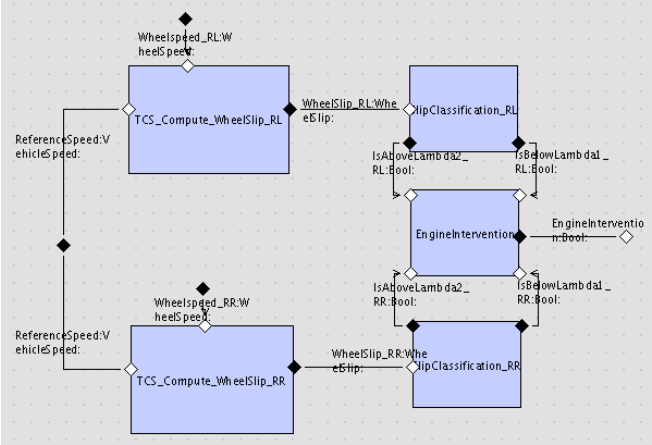

Abb. 11 DFD der Komponente `Traction_Control`

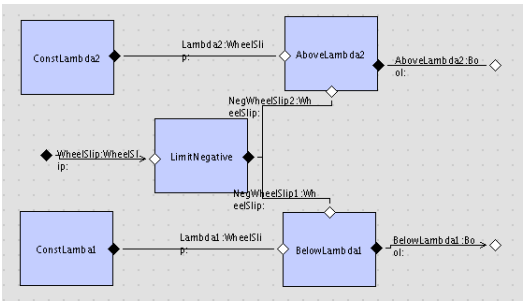

Abb. 12 DFD der Komponente `SlipClassification_RX`

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. 12 gezeigt, definiert. Die Komponente zur Regelung eines Eingriffs in die Motorsteuerung aus Abb. 11 beeinflusst die Stellung der Drosselklappe, falls der Schlupf beider Antriebsräder außerhalb der Grenzwerte ist.

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

Entsprechend der unterschiedlichen zeitlichen Auflösung der Sensoren und Aktuatoren werden Teile der Regelung mit unterschiedlichen Frequenzen ausgeführt. Im Beispiel läuft die Radgeschwindigkeits- und die Referenzgeschwindigkeitsbestimmung in 6ms Tasks, während die Antischlupfregelung in einer 12ms Task läuft. Die Drosselklappenansteuerung erfolgt im 1ms Raster (vgl. Abb. 10). 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

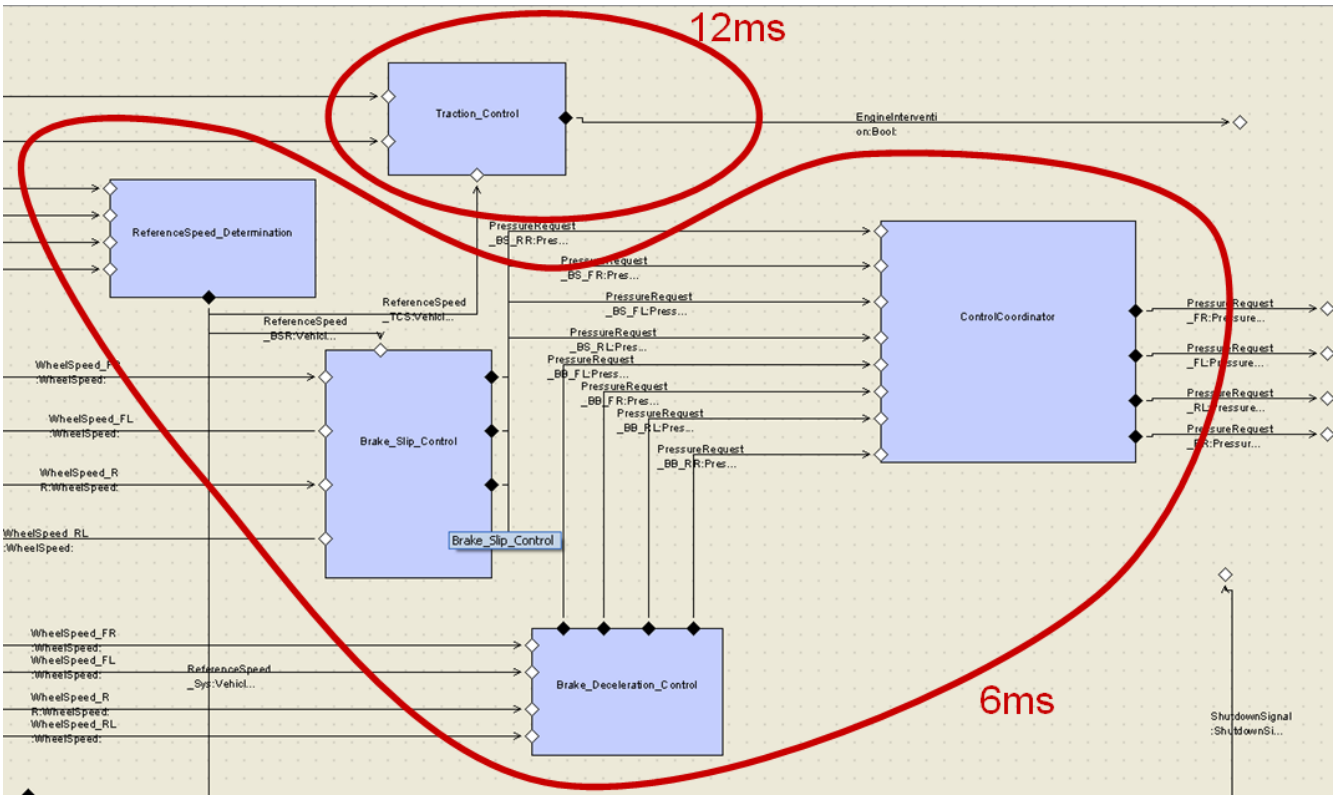

Abb. 10 DFD der Komponente `StabilityIntervention`

Task mit einem entsprechenden Ausführungsraster, die der Clock im Modell entspricht, versehen.

8 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 eine Seite bei der Entwicklung automobiler 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 Programmiersprachen. Ihre Syntax und Semantik unterscheidet sich daher von AutoFOCUS-DFDs.

Zusätzlich zur Beschreibung der Software-Anteile in einem Modell auf LA-Ebene müssen weitere hardwarespezifische Schnittstellen berücksichtigt werden. Dazu gehören Treiber für I/O-Schnittstellen oder Interrupt-Service-Routinen (ISRs).

Die AutoMoDe-Werkzeugkette wurde entwickelt, um AutoFOCUS-Modelle in echtzeitfähige Software in ASCET zu übersetzen. Diese besteht aus einem AutoFOCUS-Plugin und den ETAS Werkzeugen ASCET [9], RTA-OSEK [13] und INTECRIO [10]. Die Werkzeugkette ist schematisch in Abb. 1 zu sehen. Im Folgenden wird zuerst das Vorgehen bei der Übersetzung kurz erläutert. Die Werkzeugkette überführt Cluster, die im AutoFOCUS-Modell definiert wurden, in Module, Prozesse und Nachrichten in ASCET. Mit Hilfe des ASCET-Codegenerators werden diese Cluster in C-Code übersetzt. Die Cluster werden dann auf ein „Rapid Development System“ überführt. Die Integration auf dieser Zielplattform umfasst die Festlegung eines

Name	ConstLambda2
Clock	
Function	MakeWheelSlip(-0.2)

Abb. 13 Eigenschaften des atomaren Blocks `ConstLambda2`

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

Abb. 14 Eigenschaften des atomaren Blocks `AboveLambda2`

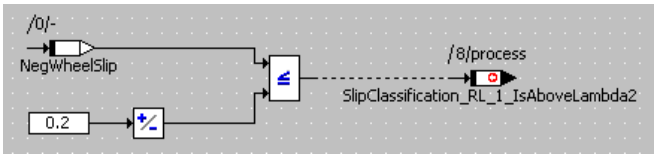

Abb. 15 ASCET-Blockdiagramm für `AboveLambda2`

Betriebssystem-Schedules. Der Schedule wird dabei direkt aus den Clocks in den AutoFOCUS-Modellen abgeleitet. Die Hardware-spezifischen Schnittstellen müssen zusätzlich manuell hinzugefügt werden.

Im Folgenden werden die verschiedenen Transformationsschritte detailliert beschrieben: *Modulidentifizierung*, *Sequenzialisierung*, *Cluster-Definition*, *Software-System-Konstruktion*, *Target-Integration* und *OS-Konfiguration*.

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

Jedes Cluster entspricht einem flachen Blockdiagramm in ASCET[1]. Aus den hierarchisch strukturierten DFDs werden die atomaren AutoFOCUS-Blöcke, deren Verhalten durch Ausdrücke in einer textuellen Sprache angegeben ist, 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 im Folgenden die Grenzwertüberprüfung in der Komponente `SlipClassification_RX` betrachtet (vgl. Abb. 12). `RX` steht dabei für `RL` (Rear Left - hinten links) oder `RR` (Rear Right - hinten rechts). Die Komponente überprüft, ob der tatsächliche Schlupf über dem Grenzwert `lambda2` liegt. Der Block `ConstLambda2` stellt die Konstante `lambda2` zur Verfügung. Der Block selbst ist dabei durch einen einfachen textuellen Ausdruck definiert (vgl. Abb. 13). Der Block `AboveLambda2` vergleicht seine zwei Eingaben, `Lambda2` und `WheelSlip`, und liefert das Ergebnis des Vergleichs. Die Selector-Funktion `wheel_slip` wird benötigt, da der Typ `WheelSlip` als ein „Wrapper“-Datentyp in AutoFOCUS realisiert ist: `data WheelSlip = MakeWheelSlip(wheel_slip:Float)`. Die Übersetzung dieser beiden Blöcke in ein ASCET-Blockdiagramm ist in Abb. 15 zu sehen.

Die vollständige Übersetzung in das ASCET-Blockdiagramm des Clusters `SlipClassification_RX` aus Abb. 12 wird in Abb. 16 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 dabei abhängig von der Anzahl der Clocks, die dem AutoFOCUS-Modell zugrunde liegen. Im Beispiel von `SlipClassification_RX` enthält das Modell lediglich eine Clock. Daher wird für die Übersetzung nur ein ASCET-Prozess benötigt. Die Zuweisung von Ausführungsreihenfolgen folgt den Kausalitäten, die inhärent in einem AutoFOCUS-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 interne Variable `NegWheelSlip` eingeführt. Diese Variable speichert das Ergebnis des `WheelSlip`-DFD-Blocks und wird anschließend für die Berechnung von `BelowLambda1` und `AboveLambda2` benötigt.

Jeder externe Port eines Clusters wird in eine IPC-Nachricht übersetzt. Im Beispiel wird der Cluster `SlipClassification_RX` in das Modul aus Abb. 17 übersetzt. CCDs werden derzeit noch nicht explizit durch die graphischen AutoFOCUS-Editoren unterstützt. Daher entsprechen Cluster speziellen AutoFOCUS-Komponenten, die durch den Stereotype `<<cluster>>` markiert wurden (der Stereotype 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 so genannten Operation Definition Language (ODL) [22] 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 dabei Benutzerinteraktionen beinhalten. Zum Beispiel könnte

[1] Die Übersetzung der Hierarchie aus einem AutoFOCUS-Modell in eine entsprechende Hierarchie in ASCET ist technisch zwar möglich, wurde aber in der aktuellen Werkzeugkette nicht verwirklicht.

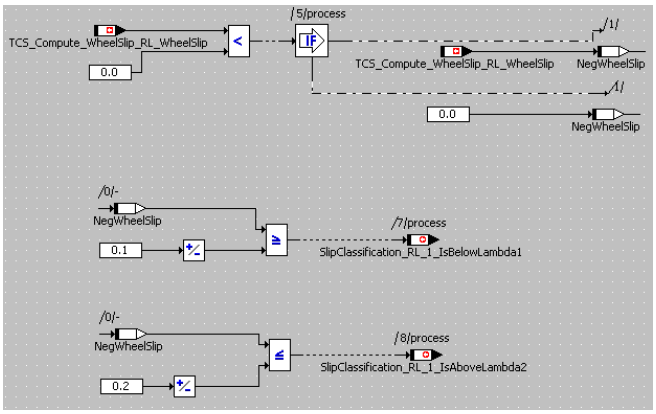

Abb. 16 ASCET-Blockdiagramm `SlipClassification_RX`

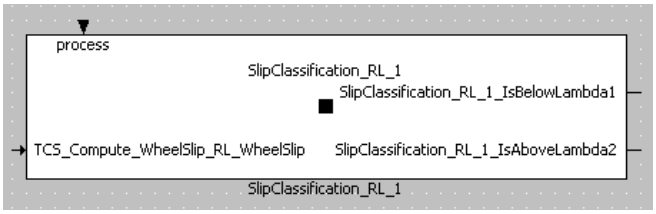

Abb. 17 ASCET-Modul `SlipClassification_RL`

der Clusterungs-Algorithmus den Benutzer nach einer Clock und nach einer Teilmenge von Komponenten, die diesen Clock benutzen, fragen. Basierend auf diesen Informationen transformiert der Algorithmus das AutoFOCUS-Modell, so dass anschließend die CCD-Cluster enthalten sind und Kommunikationsbeziehungen entsprechend umstrukturiert wurden.

Cluster-Definition. Entsprechend dem CCD aus Abb. 7 werden bei der Übersetzung für die Cluster `VehicleData`, `Traction_Control` und `Compute_ReferenceSpeed` drei ASCET-Module, wie sie in Abb. 18 zu sehen sind, 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, könnte die Verteilung der Module auf ASCET-Projekte leicht 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

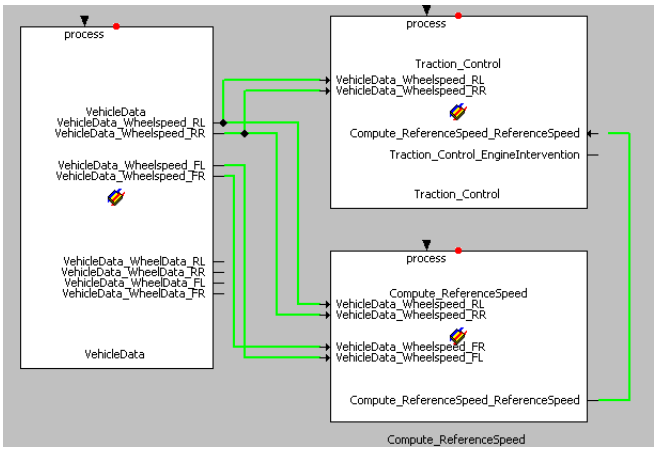

Abb. 18 ASCET-Module, die drei AutoFOCUS-Clustern entsprechen

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. 19 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 Traktionskontrolle 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 Traktionskontrolle 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 Traktionskontrolle.

Dieses auf Clocks basierende Modell wird in ein echtzeitfähiges System übersetzt. Dabei wird die Drosselklappenansteuerung im 1ms Raster ausgeführt. Aus der Kombination mit den Clocks des AutoFOCUS-Modells ergeben sich dann also Tasks im 1ms, 6ms und 12ms Raster. Die Prozesse der INTECRIO-Module der einzelnen Komponenten werden dann den entsprechenden Tasks zugeordnet.

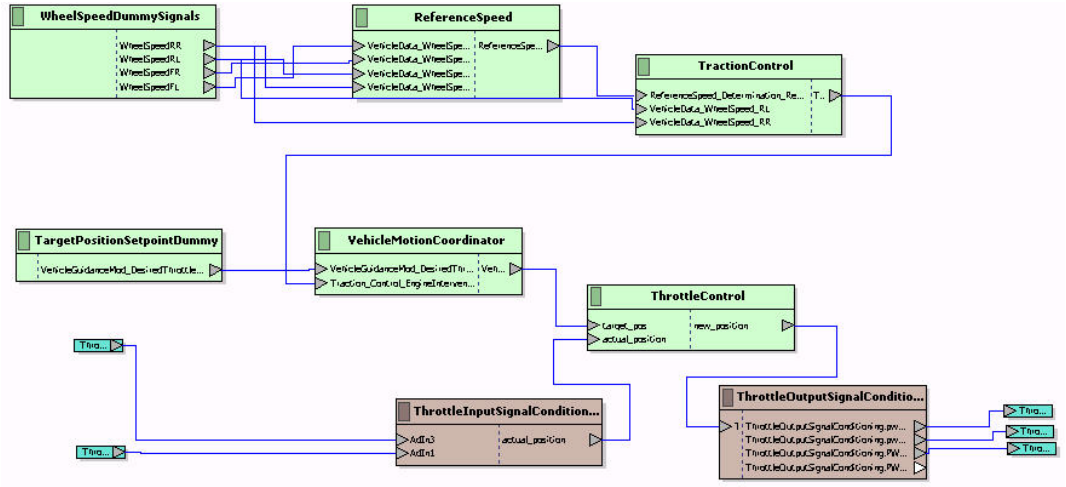

Abb. 19 Die Traktionskontrolle in INTECRIO

Wie detaillierter in Abschnitt 5 besprochen, beschreibt [3] ein „correct-by-construction"-Verfahren zur Umsetzung von zeitsynchronen AutoFocus-Modellen basierend auf einem rate-montonic Scheduling. Der Ansatz nutzt die oben beschrieben Doppel-Puffer-Technik für Kommunikation von niederfrequenten zu hochfrequenten Tasks. Für die Analyse dieser Default-Konfiguration kann die Planner-Funktionalität des Werkzeugs RTA-OSEK [13] eingesetzt werden, das auf Algorithmen basiert, wie sie in [25] beschrieben wurden.

Im dem Fall, dass der einfache rate-montonic, top-down Ansatz aus [3] nicht ausreicht, können die Algorithmen aus [2] 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 [23].

9 Werkzeugunterstützung

In diesem Abschnitt soll abschließend auf den Werkzeugprototyp AutoFocus 2 selbst eingegangen werden, mit dem die Modelle in den vorgestellten Beschreibungstechniken editiert und analysiert werden können.

9.1 Editoren

Der AutoFocus 2-Prototyp unterstützt das Editieren der in diesem Artikel beschriebenen Beschreibungstechniken, insbesondere die im Vergleich zu [7] neu hinzugekommenen

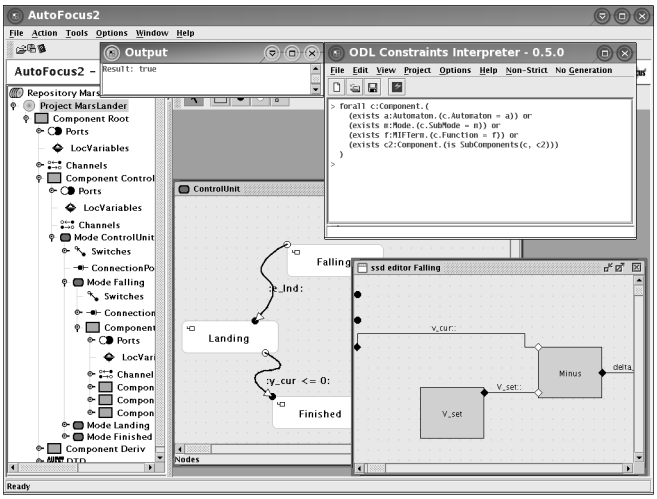

Abb. 20 Screenshot AutoFocus 2-Werkzeugprototyp

Notationen wie DFDs, CCDs und MTDs. Die in [7] genannten State Transition Diagrams (STD), Extended Event Traces (EET), sowie Data Type Definitions (DTD) werden weiterhin unterstützt und können mit den neuen Beschreibungstechniken sinnvoll kombiniert werden. Speziell die DTD-Sprache wurde dabei um Konstrukte für die in Abschnitt 6.1 genannten Implementierungstypen erweitert. Abb. 20 zeigt einen Screenshot von AutoFocus 2.

9.2 Transformationen und Konsistenzprüfungen

Neben der Unterstützung verschiedener Beschreibungstechniken und Sichten stehen mit der Ausführungsumgebung AQuA für die Logiksprache ODL [21] in AutoFocus umfassende Möglichkeiten für die automatische Durchführung von Konsistenzüberprüfungen sowie die interaktive Transformation von Modellen zur Verfügung.

Die aktuellen, domänenspezifischen Konsistenzüberprüfungen und Modelltransformationen sind dabei jeweils zugeschnitten auf die eingangs erwähnten Abstraktionsebenen FAA, FDA, sowie LA/TA. Dazu wird ODL in zwei Varianten eingesetzt. Zum einen kann mit Hilfe von automatisch überprüfbaren ODL-Bedingungen die Einhaltung der für eine gegebene Abstraktionsebene definierten Einschränkungen hinsichtlich der zulässigen Modelle durch das Werkzeug gewährleistet werden. So kann beispielsweise sichergestellt werden, dass innerhalb der LA tatsächlich Verzögerungen an den Clustergrenzen vorhanden sind. Zum anderen stehen in ODL beschriebene Standardtransformationen zur Verfügung, um effizient häufige Entwicklungsschritte durchzuführen. So wird beispielsweise das Einbinden von Grundfunktionen (z. B. Fehlerprotokoll, Reset, Parametrierung) auf der Ebene der FAA/FDA angepasst an die entwickelte Anwendungsfunktion (z. B. Fensterheber).

Darüber hinaus wird die gezielte Bereitstellung von Transformationsschnitten in Abhängigkeit der jeweiligen Abstraktionsebene durch eine phasenorientierte Benutzerführung von AutoFOCUS realisiert. Dazu werden dem Benutzer aus einer umfassenden Bibliothek von Standardtransformationen jeweils nur solche Operationen angeboten, die in der aktuellen Entwicklungsphase auch sinnvoll sind (z. B. Einbinden von Funktionen in der FAA/FDA, Refactoring in FDA, Aufspaltung in LA, usw.). Neben der besseren Strukturierung der Entwicklungsschritte ist hierbei die Erhaltung von Konsistenzbedingungen (z. B. Verbot von Refactoring unverzögerter Komponenten über Clustergrenzen) ein wesentlicher Faktor.

9.3 Inferenz und Überprüfung von Clocks

Im AutoFOCUS 2-Prototypen werden während der Bearbeitung eines Modells mittels eines *Clock Checkers* ständig die Clocks auf sämtlichen Datenflüssen überprüft und anhand der Struktur des Modells berechnet. Eine Clock wird dabei in AutoFOCUS 2 als Boolescher Ausdruck verwaltet: die Gleichheit zweier Clocks wird lediglich anhand einfacher syntaktischer Kriterien entschieden, so dass semantisch äquivalente Clocks wie z. B. `a` und `not(not(a))` in AutoFOCUS 2 jeweils als eigenständige, aber symbolisch unterschiedliche Clocks behandelt würden.

Die Funktionsweise des Clock-Checkers ist dabei ähnlich der eines Typcheckers in funktionalen Sprachen: dabei besitzt jeder auf DFD-Ebene verwendete Elementaroperator eine vorgegebene Clock-Signatur. Typisch für arithmetische und logische Operatoren ist dabei, dass alle Eingangsparameter sowie das Ergebnis der Operation derselben Clock zugehören. Im Gegensatz dazu haben Sampling-Operatoren verschiedene Clocks auf Ein- und Ausgängen: mit einem Sampling-Operator wird außerdem immer einer der Eingangsparameter vom Typ `Bool` in die Domäne der Clock-Ausdrücke überführt.

Falls die Clock eines Datenflusses durch die Kombination aus Modell und Umgebung nicht vollständig festgelegt ist, wird (analog zu polymorphen Typsignaturen in funktionalen Sprachen) eine *Clockvariable* inferiert. Über ihre interne Bedeutung für das Werkzeug hinaus sind Clockvariablen als Element der Notation nützlich, um an der Schnittstelle einer wiederverwendbaren Komponente jeweils gleich und unterschiedlich getaktete Datenflüsse unterscheiden zu können, ohne diese mit konkreten Frequenzen belegen zu müssen.

9.4 Simulation

AutoFOCUS 2 verfügt über einen eingebauten graphischen Simulator, mit dem (Teil-)Modelle mit vollständig festgelegtem Verhalten interaktiv ausgeführt werden können. Wie in grafischen CASE-Werkzeugen üblich, bleibt die vom Benutzer festgelegte Diagrammsicht dabei erhalten und wird zusätzlich mit Informationen über den simulierten Zustand des Modells zur Laufzeit (z.B. Aktivität eines Modus/Kontrollzustands, Datenzustand) angereichert.

10 Zusammenfassung und Ausblick

Der mit diesem Artikel 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 von Abstraktionsebenen sowie ausgewählte Übergänge zwischen Abstraktionsebenen diskutiert. Einige der angesprochenen Transformationen sind bereits im AutoFOCUS 2-Werkzeugprototypen implementiert. Mittels automatisierter Konsistenzprüfungen auf dem homogen definierten AutoMoDe-Domänenmodell kann die Konsistenz von Designs gegenüber Ansätzen mit heterogenen Modellierungstechniken wesentlich erhöht werden.

Der 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. M. Baleani, A. Ferrari, L. Mangeruca, A. L. Sangiovanni-Vincentelli, U. Freund, E. Schlenker, and H.-J. Wolff. Correct by construction transformations across design environments for model-based embedded software development. In *DATE 05*, 2005.
3. A. Bauer and J. Romberg. Model-based Deployment in Automotive Embedded Software: From a High-Level View to Low-Level Implementations. In *Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, Hamilton, Ontario, Canada, June 2004.
4. A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. LeGuernic, and R. D. Simone. The Synchronous Languages Twelve Years Later. *Proceedings of the IEEE*, 91(1):64–83, 2003.
5. A. Benveniste, P. Caspi, P. L. Guernic, and N. Halbwachs. Data-Flow Synchronous Languages. In *REX School/Symposium*, pages 1–45, 1993.
6. P. Braun, H. Lötzbeyer, B. Schätz, and O. Slotosch. Consistent Integration of Formal Methods. In S. Graf and M. Schwartzbach, editors, *Tool and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, number LNCS 2280. Springer Verlag, 2000.
7. M. Broy, F. Huber, and B. Schätz. AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. *Informatik Forschung und Entwicklung*, (14(3)):121–134, 1999.
8. M. Broy and K. Stølen. *Specification and Development of Interactive Systems:* Focus *on Streams, Interfaces, and Refinement.* Springer, 2001.
9. ETAS Engineering Tools GmbH. *ASCET-SD Benutzerhandbuch*, 2001.
10. ETAS GmbH Stuttgart. *INTECRIO User Guide V 1.0*, 2005.
11. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8, 1987.
12. F. Huber, B. Schätz, and G. Einert. Consistent Graphical Specification of Distributed Systems. In *Industrial Applications and Strengthened Foundations of Formal Methods (FME'97)*, LNCS 1313, pages 122–141. Springer Verlag, 1997.
13. LiveDevices York. *RTA-OSEK User Guide V 4.0*, 2005.
14. F. Maraninchi and Y. Raymond. Mode-automata: a new domain-specific construct for the development of safe critical systems. *Science of Computer Programming*, 46(3):219–254, 2003.
15. The MathWorks Inc. *Using Simulink*, 2000.
16. K. D. Müller-Glaser, G. Frick, E. Sax, and M. Kühl. Multiparadigm Modeling in Embedded Systems Design. *IEEE Transactions on Control Systems Technology*, 12(2):279–292, March 2004.
17. M. Mutz, M. Huhn, U. Goltz, and C. Krömke. Model Based System Development in Automotive. In *Proceedings of the SAE World Congress*, Detroit, MI, 2003. Society of Automotive Engineers.
18. S. Poledna, T. Mocken, J. Scheimann, and T. Beck. Ercos: An operationg system for automotive applications. In *SAE International Congress*, 1995.
19. J. Romberg. *Synthesis of distributed systems from synchronous dataflow programs*. PhD thesis, Technische Universität München, 2006.
20. T. Scharnhorst, H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, P. Heitkämper, J. Leflour, J. Mate, and K. Nishikawa. Autosar – challenges and achievements. In *Elektronik im Kraftfahrzeug 2005*. VDI, October 2005.
21. B. Schätz. The ODL operation definition language and the AutoFocus/Quest application framework AQuA. Technical Report TUM-I0111, TU München, 2001.
22. B. Schätz, P. Braun, F. Huber, and A. WIsspeintner. Checking and Transforming Models with AutoFocus. In *12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*. IEEE Computer Society, 2005.
23. J. Schäuffele and T. Zurawka. *Automotive Software Engineering*. Vieweg Verlag, Wiesbaden, 2003.
24. B. Selic, G. Gullekson, and P. Ward. *Real-Time Object Oriented Modelling*. Wiley, 1994.
25. K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming - Euromicro Journal*, 40:117–134, 1994.
26. M. v. d. Beeck. Comparision of statecharts variants. In *Third Int'l Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, pages 128–148, 1994.
27. M. von der Beeck, P. Braun, M. Rappl, and C. Schröder. Automotive UML. In B. Selic, G. Martin, and L. Lavagno, editors, *UML for Real: Design of Embedded Real-Time Systems*, number ISBN 1-4020-7501-4. Kluwer Academic Publishers, 2003.