Informatik Forsch. Entw. (2005) 19: 194–205
DOI 10.1007/s00450-005-0187-7

REGULÄRE BEITRÄGE

Andreas Bauer • Jan Romberg • Bernhard Schätz

Integrierte Entwicklung von Automotive-Software mit AutoFOCUS

Eingegangen am 27. September 2004 / Angenommen am 21. Februar 2005 / Online publiziert am 25. Mai 2005

Zusammenfassung Zur Beherrschung der komplexen vernetzten und verteilten Funktionen von Automotive-Software ist eine Beschreibung des zu erstellenden Systems auf verschiedenen Abstraktionsebenen und schrittweise Übergänge zwischen diesen Ebenen notwendig. Neben der Definition geeigneter Ebenen werden zur Unterstützung echtzeitkritischer Systemanteile ein einheitliches Berechnungsmodell, ebenenspezifische Beschreibungstechniken, sowie methodische Regeln für diese Abstraktionsebenen eingeführt und in den Werkzeugprototypen AutoFOCUS integriert.

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

Abstract For tackling the development of complex networked and distributed functionalities of Automotive software, a description of the system on different levels of abstraction, accompanied by stepwise transitions between those levels, is essential. In this article, a homogeneous model of computation, as well as notations and methodological rules corresponding to abstraction levels, are introduced for describing real-time software systems. The techniques described have been integrated into the tool prototype AutoFOCUS.

Keywords Automotive software engineering · Embedded software · Synchronous languages · AutoFocus

CR Subject Classification D.2.2

1 Einleitung

Die ständig zunehmende Gesamtkomplexität von Elektronikfunktionen im Automobil, verbunden mit dem steigenden Vernetzungsgrad von ursprünglich isolierten Funktionen, stellt hohe Anforderungen an eine Entwicklungsmethodik für Automotive-Software. Dabei sollte 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. Funktionsmodule sollten 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. Durch geeignete methodische Vorgaben und wohldefinierte Schnittstellen- und Moduldefinitionen sollte ein *hoher Wieder- und Mehrfachverwendungsgrad* von Funktionsmodulen erreicht werden. Darüber hinaus stellt sich mit dem Zusammenwachsen der Anwendungsgebiete diskreter Ereignissysteme (z. B. Komfortelektronik) und synchroner zeitgesteuerter Systeme (z. B. Antriebsstrang) 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 wird anhand einiger ausgewählter Beschreibungstechniken ein modellbasierter Ansatz zur Automotive Software-Entwicklung vorgestellt, der durch das AutoFOCUS-Werkzeug [6] unterstützt wird. In Abschn. 7 wird dabei ein aktueller Werkzeugprototyp für AutoFOCUS beschrieben. Konkret bietet AutoFOCUS dem Entwickler eine Anzahl graphischer und textueller Beschreibungstechniken (siehe Abschn. 4.2 ff.) für verschiedene Abstraktionsebenen (siehe Abschn. 3). Damit ist die Modellierung von Struktur und Verhalten von Software durchgängig möglich: angefangen von der Erfassung funktionaler Abhängigkeiten, bis hin zum *Deployment*, also der Verteilung von Applikationen auf Prozesse, Tasks und reale ECUs im Bordnetzverbund des Fahrzeugs (siehe Abschn. 4.5, 5). Zusätzlich existieren eine Reihe von Anbindungen für Techniken der Konsistenzanalyse, formalen Verifikation sowie Testfallgenerierung [4], sowie Code-Generatoren für die Programmiersprachen C und Ada.

A. Bauer · J. Romberg · B. Schätz
Institut für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching b. München

Etablierte Entwicklungsprozesse setzen häufig auf einer heterogenen Werkzeug- und Notationslandschaft auf. Probleme ergeben sich typischerweise beim Übergang zwischen Notationen sowie beim Versuch ganzheitlicher Analysen auf heterogenen Modellen, wie z. B. Konsistenzprüfungen. Die hier vorgestellte Arbeit versucht zu demonstrieren, wie ein nichttrivialer Anteil des automobilen Software-Entwicklungsprozesses auf einer *einheitlichen* notationellen und methodischen Basis unterstützt werden kann.

2 Verwandte Ansätze

Aufgrund der vielfältigen Bezüge von autombiler Softwaretechnik zu Maschinenbau und Regelungstechnik kommen im Kfz-Bereich regelungstechnisch motivierte Werkzeuge wie Matlab/Simulink [9] oder ASCET [7] mit geeigneten Blockbibliotheken für Elementarfunktionen wie Integratoren, Kennfeldern oder Filtern zur Anwendung. Als Konsequenz stehen in diesen Ansätzen Modellierungselemente wie synchrone Funktionsblöcke, Signale, und Perioden im Vordergrund. Ein gemeinsames Prinzip dieser Ansätze ist zum einen die explizite Modellierung von Datenflüssen in regelungstechnischen Blockdiagrammen. Häufig liegt solchen Modellen eine aus der physikalischen Umgebung motivierte synchrone, uniforme Zeitbasis zugrunde. Eine Betrachtung tieferliegender softwaretechnischer Problemstellungen wie nebenläufige Tasks, Schutz gemeinsamer Ressourcen, Typen auf Implementierungsebene usw. wird auf dieser Ebene geeigneterweise vermieden oder wie in [7] mit orthogonalen Aspekten der Beschreibung verknüpft.

In anderen Anwendungsgebieten (z. B. Telekommunikation) wird Software vorherrschend zur Abwicklung diskreter, ereignisgesteuerter Aufgaben (z. B. Vermittlung) eingesetzt, mit Werkzeugen wie Telelogic Tau oder ObjecTime [15] und Modellierungselemente wie kommunizierende Komponenten, Nachrichten, Zustände, sowie Sende- und Empfangsereignisse. Diese Konzepte wurden wegen der Möglichkeit, reaktive Systemen im Zusammenhang mit einigen Echtzeitprimitiven zu beschreiben, unter dem Begriff UML-RT (UML for Real-Time) zusammengefasst und schließlich in die UML (z. B. [10]) aufgenommen. Als Ergänzung zur UML-RT wurde mit einem zusätzlichen UML-Profil [11] ein Metamodell entwickelt, um die in der Echtzeitprogrammierung eingesetzten Konzepte und Begriffe (z. B. Ereignis, Aktion, Ressource, Schedule) innerhalb des Standards abbilden zu können.

Gegenüber diesen etablierten Ansätzen bietet unser AutoFocus-gestützter Ansatz die folgenden wesentlichen Neuerungen:

- wie in den Abschn. 4 und 5 beschrieben, werden domänenspezifische Konzepte (z. B. Betriebsmodus, Funktion, Periode; aber auch Prozessor, Kommunikationspfad) in ein ganzheitliches, für die Domäne maßgeschneidertes Modell integriert
- wie in Abschn. 4 gezeigt, werden diese neu eingeführten domänenspezifische Konzepte zu einzelnen Teilansätzen (z. B. DFD, MSD) zusammengefasst und geeignete einfache Ausführungsmodelle für diese unterschiedlichen domänenspezifischen Modelle in AutoFocus definiert, um so die Komplexität der einzelnen Sichten zu beschränken
- die eingeführten Modelle werden den verschiedenen Abstraktionsebenen bei der Modellierung verteilter eingebetteter Software zugeordnet (z. B. die in Abschn. 3 erwähnten Ebenen). Somit wird ein domänenspezifischen Entwicklungsprozess (z. B. ITEA EAST/EEA [1]) um konkrete, domänenspeifische Beschreibungsmittel erweitert.
- wie in Abschn. 6 beschrieben, werden methodisch sinnvolle Übergänge zwischen diesen Modellen anhand des Domänenmodells definiert und die werkzeugtechnische Unterstützung für solche Übergänge aufgezeigt.

3 Abstraktionsebenen

Ähnlich zu verwandten Ansätzen [5, 1] basiert die AutoFocus-gestützte Entwurfsmethodik auf einer Gliederung in domänenspezifische *Abstraktionsebenen*. Unsere Gliederung ist dabei an die von [1] angelehnt (siehe Abb. 1).

Die abstrakteste der betrachteten Ebenen ist dabei 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. Im folgenden werden speziell die durch Software realisierte Funktionalitäten betrachtet.

Systeme sind auf FAA-Ebene typischerweise nach *funktionalen Gesichtspunkten* oder auch 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

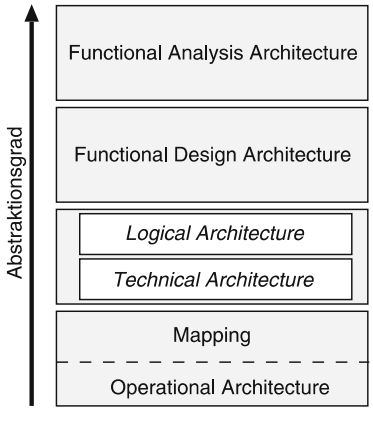

Abb. 1 AutoFocus-Abstraktionsebenen für den Automotive Software-Entwicklungsprozess

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).

Die *Functional Design Architecture (FDA)* stellt bereits eine bzgl. der *Struktur* und des *Verhaltens* vollständige Beschreibung von Software-Systemen im Automobil dar. Werkzeugseitig kann in AutoFOCUS die Strukturvollständigkeit dabei als Vollständigkeit der Kommunikationsverbindungen angesehen 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.

Bezüglich der Mehrfachheit 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 Vorverarbeitung und Aufbereitung von Sensorsignalen, die verteilt kommuniziert und verwendet werden.

FDA-Komponenten können wiederum hierarchisch durch andere Komponenten aufgebaut sein, und sind über typisierte, gerichtete Kanäle und Konnektoren miteinander verbunden.

Die detaillierteste Abstraktionsebene gliedert sich in *Logical Architecture* und *Technical Architecture*. In der Logical Architecture (LA) werden die Software-Komponenten in sog. Cluster gruppiert und dabei Variationspunkte aufgelöst. 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 behandelten *Operational Architecture*.

4 AutoFOCUS – Notationen und Semantik

4.1 Berechnungsmodell

Das Berechnungsmodell von AutoFOCUS beruht auf einer *taktsynchronen* Ausführung der Einzelkomponenten mit uniformem, systemweitem Takt. Dabei werden Rechenschritte innerhalb eines Taktes nicht weiter zeitlich aufgelöst. Um die heterogenen (a)periodischen Takte bzw. Frequenzen typischer Automotive-Anwendungen komfortabel modellieren zu können, werden in AutoFOCUS sog. *Clocks* [3] verwendet. Mit jedem Datenfluss innerhalb eines AutoFOCUS-Designs ist ein Boolescher Ausdruck (Clock) assoziiert, der die Anwesenheit bzw. Abwesenheit einer Nachricht angibt. Mit Hilfe von strukturell 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). Mit expliziten *Sampling-Operatoren* zwischen Komponenten bzw. Clustern kann ein Datenfluss von einer Clock auf eine andere überführt werden (siehe Abschn. 5).

4.2 System Structure Diagrams

System Structure Diagrams (SSD) 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.

Abbildung 2 zeigt typische SSDs für die FAA- und FDA-Ebene. Das SSD in Abb. 2a zeigt die FAA-Abstraktion einer einfachen Längsdynamiksteuerung, die sowohl auf Anfragen des Fahrers (Signal `driverAccReq`) als auch für jedes Rad spezifische Eingriffe der Traktionskontrolle `TractionIntervention` (Signal `momentumReq`) reagieren muss. Dabei stehen die (auf jedes Rad spezifisch wirkende) Bremse `Brake` sowie der Antrieb `Propulsion` zur Verfügung. 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 `LongitudinalDynamicsController` repräsentierte Stelle im Modell, wo ein möglicher Koordinationsbedarf zwischen den (funktional eng verwandten) Anfragen `driverAccReq` und `momentumReq` offensichtlich wird. AutoFOCUS-SSDs bieten derzeit zur Modellierung und Kennzeichnung solcher Stellen die Möglichkeit des Einfügens einer Komponente mit mehr oder weniger definiertem Verhalten und einem Stereotypen zur Markierung (im Beispiel `<<coordination>>`). Eine stärkere notationelle und methodische Trennung zwischen elementaren

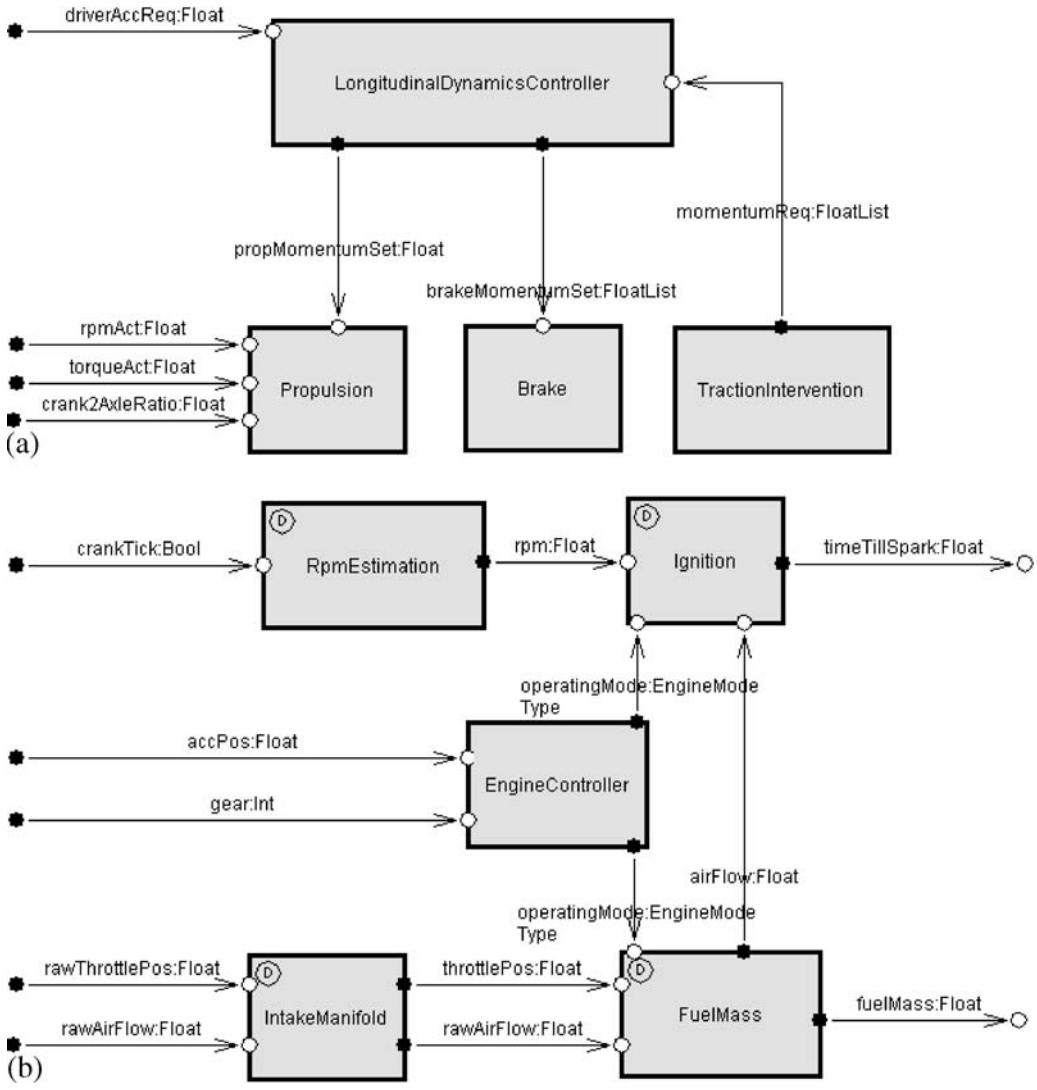

Abb. 2 Beispiele für SSDs auf FAA- **(a)** und FDA-Ebene **(b)**

Funktionalitäten einerseits und Koordinationsaufgaben andererseits im Sinne von [8] ist Gegenstand laufender Arbeiten.

Abbildung 2b zeigt die Verwendung des SSD-Formalismus auf FDA-Ebene am Beispiel einer Motorsteuerung (Ausschnitt). Durch die in Abschn. 7.2 beschrieben Konsistenzprüfungen kann die Struktur- und Verhaltensvollständigkeit für solche Modelle auf FDA-Ebene überprüft und sichergestellt werden.

4.3 Data Flow Diagrams

Data Flow Diagrams (DFD) 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 (durch DFD definierten) *Blöcken* aufgebaut, die ähnlich zu SSDs über gerichtete und typisierte Kanäle verbunden sind. Weiterhin können DFDs einem speziellen *Modus* eines *Mode Switch Diagramms* zugeordnet sein, der über die Aktivierung einer Berechnung entscheidet.

In Abb. 3 ist ein DFD für eine einfache Motorvortriebs-Regelung abgebildet. Anders als bei SSDs sind die DFD-Ports diamantförmig, was im Berechnungsmodell einem unverzögerten Nachrichtenaustausch entspricht, d. h. die Bereitstellung einer Nachricht an einem Ausgang und ihre Ankunft an einem Eingang werden in Bezug auf den Modelltakt als gleichzeitig angenommen. Aufgrund des taktsynchronen Berechnungsmodells müssen z. B. in Rückkopplungsschleifen explizite Verzögerungs-Operatoren verwendet werden, um kausale Zyklen zu vermeiden.

4.4 Mode Switch Diagrams

Die gegenüber [6] neu eingeführten *Mode Switch Diagrams* (MSD) werden auf allen Abstraktionsebenen dazu verwen-

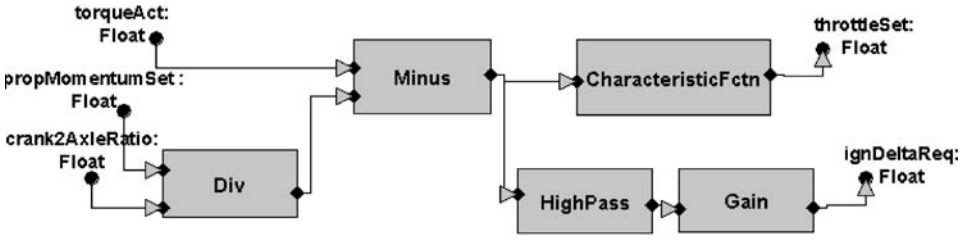

Abb. 3 DFD für eine Vortriebsmoment-Steuerung

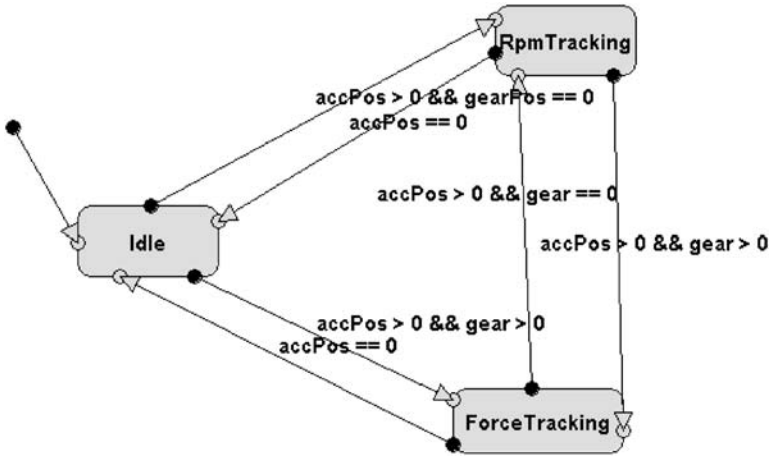

Abb. 4 MSD für `EngineController`

det, zwischen alternativen Betriebsmodi oder Konfigurationen einer Komponente zur Laufzeit hin- und herzuschalten.

Abhängig vom Zustand eines MSDs können so unterschiedliche, den Zuständen zugeordnete DFDs als Berechnungsvorschrift einer Komponente aktiviert sein. Dabei ist die graphische Darstellung ähnlich der eines endlichen Automaten: Knoten des Graphen entsprechen *Modi*, Kanten entsprechen Transitionen oder *Switches*. Jeder Switch 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 einen Switch gewechselt, falls die Auswertung der Booleschen Bedingung logisch wahr ergibt.

Abb. 4 zeigt ein Beispiel für die Modellierung von Betriebszuständes einer Motorsteuerung: dabei wird zwischen den Modi `Idle` (Leerlaufsteuerung), `RpmTracking` (Steuerung der Drehzahl) und `ForceTracking` (Steuerung des Kurbelwellenmoments) unterschieden und je nach Wertbelegung der Eingänge `gear` (eingelegter Gang) und `accPos` (Gaspedalstellung) umgeschaltet und ein den jeweiligen Modi `Idle`, `RpmTracking` und `ForceTracking` zugeordnetes DFD aktiviert.

4.5 Cluster Communication Diagrams

Cluster Communication Diagrams (CCD) werden auf der Ebene 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 mehrere Tasks einer Applikation verteilt. Für den Übergang FDA→LA ist angestrebt, dass trotz der in Abschn. 6.1 beschriebenen Umstrukturierungen das Verhalten des Modells bis auf elementare Transformationen (z. B. Ersetzen der abstrakten Typen auf FDA-Ebene durch Implementierungstypen) erhalten bleibt.

Die graphische Notation für CCDs entspricht der von DFDs. Auf CCDs gelten jedoch weitere Einschränkungen: Cluster können keine Subcluster enthalten. Ausserdem werden die in Abschn. 5 näher beschriebenen Einschränkungen für Verzögerungen auf CCDs überprüft und durchgesetzt.

5 Explizite Zeitmodellierung

Die Modellierung von zeitlichen Abhängigkeiten und Verhalten geschieht in AutoFOCUS explizit durch Verwendung von *Sampling-Operatoren* und *Unit-Delays*. Die Sampling-Operatoren dienen im Wesentlichen zur gegenseitigen Abtastung von verschieden getakteten Signalströmen, wohingegen Unit-Delays lediglich Verzögerungen einführen.

Im Beispiel Motorsteuerung treten beispielsweise periodische Signale unterschiedlicher Frequenzen auf (z. B. periodische Übermittlung der aktuellen Drehzahl), azyklische Ereignisse (z. B. Benutzereingriffe, Überschreitung von Schwellwerten), sowie winkel- bzw. kurbelwellensynchrone Signale (z. B. Zündzeitpunktberechnung, Einspritzung).

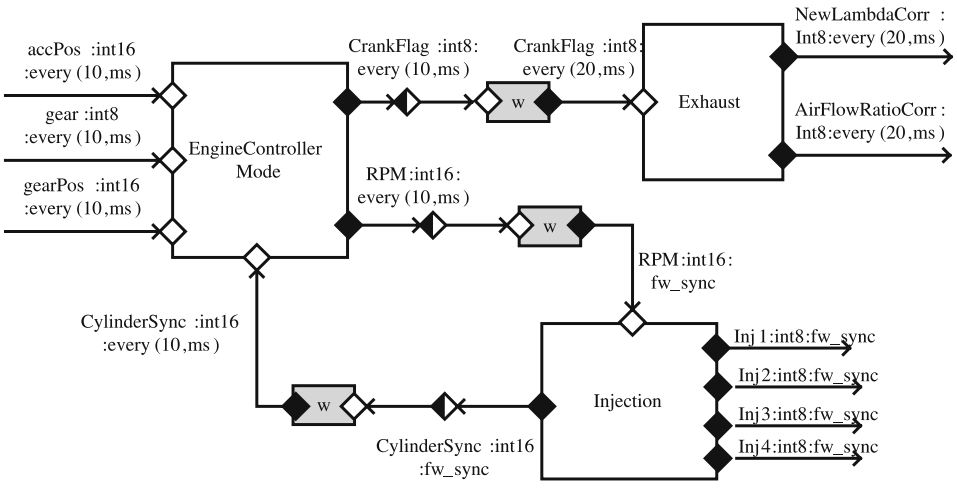

Abb. 5 CCD einer Motorsteuerung (abstrahiert)

In Abb. 5 wird dies anhand von drei abstrakten Clustern verdeutlicht, wie sie typischerweise in einer elektronischen Motorsteuerung eines Verbennungsmotors zu finden sind: die Abläufe zur Einspritzmengenberechnnung für die einzelnen Zylinder innerhalb von `Injection` werden durch kurbelwellenabhängige Interrupts aktiviert, während `Exhaust`, d. h. vorwiegend Vorgänge zur Katalysatorkalibrierung, im 20 ms und `EngineControllerMode` (vgl. Abb. 4) im schnelleren 10 ms Raster aktiviert wird. Somit existieren zwischen den einzelnen CCDs zeitliche Schnittstellen, die durch Sampling-Operatoren (grau) und Unit-Delays (diamantförmige Operatoren) in der Abbildung gekennzeichnet sind. Der zentrale Cluster `EngineControllerMode` hat zu allen anderen Clustern Kommunikationsschnittstellen: er schreibt beispielsweise alle 10 ms `CrankFlag` (Identifiziert den Anlassvorgang) und `operatingMode` (Betriebsmodus) und liest von Cluster `Injection` das durch Sampling verarbeitete `CylinderSync` (Lokale Synchronisierung der Zylinderstellungen).

Die unterschiedlichen Frequenzen lassen sich anhand der Clocksignaturen eindeutig identifizieren, z. B. repräsentiert der Ausdruck `every(10,ms)` jeden zehnten Untertakt des Millisekunden-Takts, und `fw_sync` den kurbelwellensynchronen Takt. Jede der Clocksignaturen legt also explizit die Periodizität eines Kommunikationskanals fest. *Sampling-Operatoren* werden immer dann benötigt, wenn die Clocks zwischen zwei zusammenhängenden Ports nicht übereinstimmen. In AutoFOCUS wird dazu das Sampling mittels der Operatoren `when` (in der Abbildung durch „w" abgekürzt) bzw. `sample` durchgeführt. Die Semantik von `when` kann exemplarisch wie folgt definiert werden:

	a	1	2	3	4	5	6	7	8
	b	*tt*	*ff*	*tt*	*ff*	*tt*	*ff*	*tt*	*ff*
a when b		1	⊥	3	⊥	5	⊥	7	⊥

In diesem Beispiel wird ein Signalstrom `a` mit dem booleschen Signalstrom `b` synchronisiert. `a` und `b` liegen dabei beide auf der sog. Basis-Clock (z. B. ein Millisekunden-Raster), d. h. zu jedem Takt liegt ein Wert an, doch die zum Sampling verwendeten Werte von `b` sind relativ nur jeden zweiten Takt *tt* (bzw. *ff*). Deshalb ist das Signal `a when b` ebenfalls nur jeden zweiten Takt anwesend.

Das Delay-Glied hält den letzten Eingangswert jeweils für eine Periode zurück. In einer Implementierung ist deshalb mit jedem Delay mindestens eine Speicherstelle assoziiert, die einen Wert bestimmten Typs speichert.

Für eine Implementierung des CCDs mit mehreren Tasks und einer vorgegebenen Scheduling-Policy ergeben sich anhand der in [2] definierten Kriterien Einschränkungen für das Vorhandensein von Delays zwischen Clustern. So muss z. B. für Rate Monotonic Scheduling für die Kommunikation von Cluster mit großer Periode zu Clustern mit kleiner Periode mindestens ein Delay vor dem Sampling-Operator eingefügt werden. Als Beispiel ist dazu in Abb. 6 ein detaillierterer Ausschnitt der Motorsteuerung dargestellt. Er konzentriert sich lediglich auf die Schnittstelle zwischen der zentralen Moduskomponente `EngineControllerMode` und dem CCD `Exhaust`, welches die Abgasregelung übernimmt. An der Schnittstelle wird zunächst das Ausgangssignal mit der Signatur `CrankFlag:int8:every(10,ms)` durch den Unit-Delay um eine Periode verzögert und anschließend durch „when" auf `every(20,ms)` verlangsamt. Neben dem eigentlichen Eingangssignal, `CrankFlag`, benötigt der when-Operator dazu einen zusätzlichen Strom, dessen Clock die Ausgangsfrequenz festlegt. In diesem Fall wird dazu ein Event-Strom als Basis-Takt verwendet und durch die Anweisung `every(20,ms)` ein boolescher Sampling-Strom erzeugt.

Die Clocks stellen somit ein wichtiges Modellierungswerkzeug dar, insbesondere um unterschiedliche Frequenzen und Reaktionszeiten von Komponenten abzubilden. Die Abbildung von Frequenzen im Domänenmodell ist insbesondere wichtig, um die Zusammenfassung in Clustern und das Mapping von Clustern auf Tasks technisch effizient zu gestalten, und um z. B. die Konsistenzbedinungen bezüglich Delay-Gliedern zwischen Clustern werkzeugseitig überprüfen zu können.

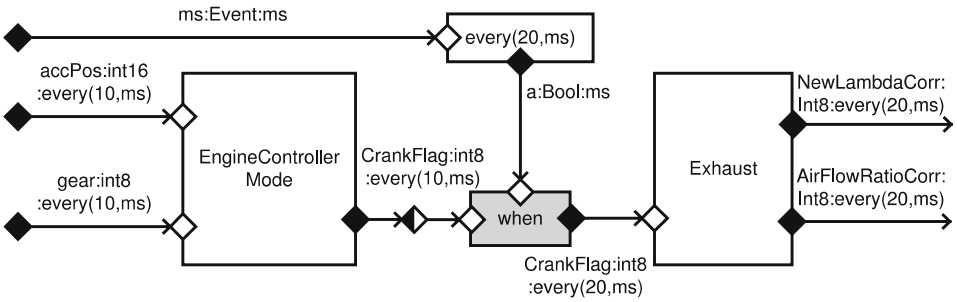

Abb. 6 Detailansicht von Abb. 5 mit Clustern `EngineControllerMode` und `Exhaust`

Mit Hilfe der AutoFOCUS-Konstrukte für Zeitmodellierung ist zumindest lokal für einzelne ECUs eine verhaltenskonforme Implementierung mit einer oder mehreren Tasks möglich [2]. Für Anwendungen mit harten Echtzeitanforderungen kann so (in Verbindung mit einer genauen Laufzeitanalyse für die einzelnen Tasks) die Korrektheit der Implementierung bezüglich des Modells sicher gestellt werden. Für die Implementierung taktsynchroner Modelle auf verteilten Steuergerätenetzwerken bieten kommende Bussysteme wie FlexRay mit Uhrensynchronisierung und deterministischen (time-triggered) Protokollsegmenten gute Voraussetzungen. Die Untersuchung von verhaltenskonformer Implementierung von AutoFOCUS-Modellen auf der Basis von ereignisorientierten Netzwerken wie CAN ist Gegenstand laufender Arbeiten [12].

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 Abschn. 3 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) vorhanden.

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 [14] 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 ist.

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 [2]). 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 MSDs (vgl. Abschn. 4.4) und damit die explizite Modellierung von Modus-Informationen in AutoFOCUS-Designs eignet sich insbesondere zur Modelltransformation *innerhalb* einer Abstraktionsebene (Refactoring). MSDs 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. 7 ein abstrahiertes Beispiel aus

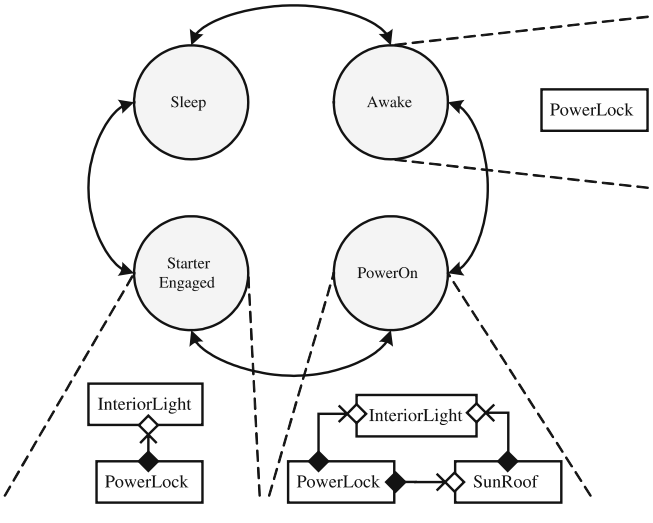

Abb. 7 Innenraum-Bedienelemente abhängig vom Betriebsmodus

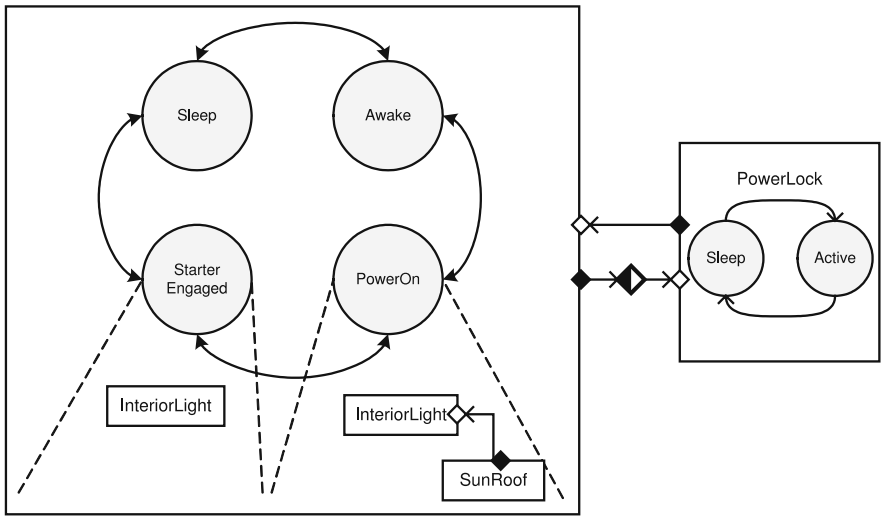

Abb. 8 Eine auf technische Umsetzung optimierte Modellierung des Systems aus Abb. 7

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.

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. 8 dargestellte Refactoring greift beide Punkte auf, indem es einen Block bzw. Komponente `PowerLock` aus einem gegebenen MSD 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.

Solche Refactorings können neben technischen Gesichtspunkten auch einem anderen Zweck dienen, beispielsweise der Überführung einer eher kontrollflussorientierten Sicht wie in Abb. 7 (d. h. viele Komponenten ordnen sich genau einem Modusdiagramm unter), hin zu einer eher datenflussorientierten Darstellung ähnlich zu Abb. 8 (d. h. die einzelnen Komponenten unterscheiden individuell bzw. intern nach dem Modus – ohne ein übergeordnetes MSD).

Die dazu notwendigen Arbeitsschritte sind in AutoFOCUS einerseits durch die Editoren, andererseits durch eine integrierte Logiksprache, ODL (siehe Abschn. 7.2), unterstützt, die es ermöglicht, solche Transformationsschritte abstrakt zu definieren und automatisch für verschiedene Modelle beliebig oft wiederzuverwenden.

7 Werkzeugunterstützung

In diesem Abschnitt soll kurz der Werkzeugprototyp AutoFOCUS 2 vorgestellt werden, mit dem Modelle in der aktuellen AutoFOCUS-Beschreibungstechnik editiert und analysiert werden können.

7.1 Editoren

Der AutoFOCUS 2-Prototyp unterstützt das Editieren der in diesem Artikel beschriebenen Beschreibungstechniken, insbesondere die im Vergleich zu [6] neu hinzugekommenen Notationen wie DFDs, CCDs und MSDs. Die in [6] 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 Abschn. 6.1 genannten Implementierungstypen erweitert. Abb. 9 zeigt einen Screenshot von AutoFOCUS 2.

7.2 Transformationen und Konsistenzprüfungen

Neben der Unterstützung verschiedener Beschreibungstechniken und Sichten stehen mit der Ausführungsumgebung

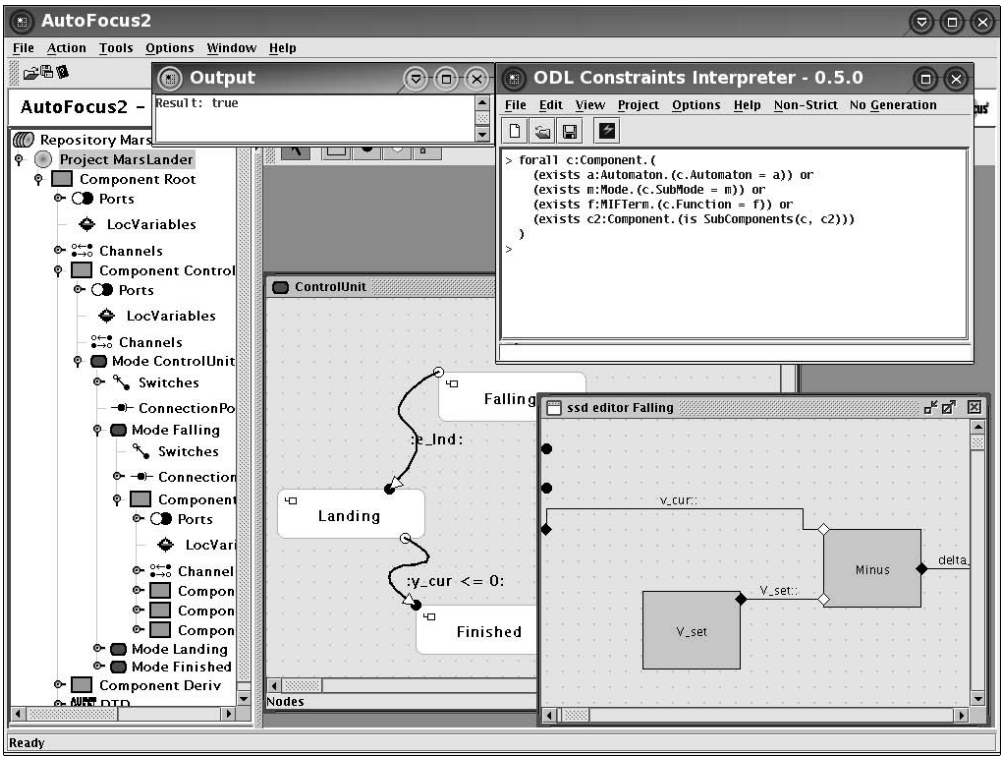

Abb. 9 Screenshot AutoFOCUS 2-Werkzeugprototyp

AQUA für die Logiksprache ODL [13] in AutoFOCUS nun auch 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.

7.3 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 als verschieden behandelt würden.

Die Funktionsweise des Clock-Checkers ist dabei ähnlich der eines Typcheckers in funktionalen Sprachen: dabei besitzt jeder auf DFD-Ebene verwendete Elementaroperator auf eine vorgegebene Clock-Signatur. Typisch für arithmetische und logische Operatoren ist dabei, dass alle Eingangsparameter sowie das Ergebnis der Operation auf derselben Clock sein müssen. 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. So ist z. B. der bereits erwähnte `every`-Operator ein Boolescher Operator; in dem in Abb. 6

gezeigten CCD wird der Boolesche Ausdruck `every(20, ms)` mittels des Operators `when` in die Clock-Domäne aufgenommen.

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.

7.4 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.

8 Zusammenfassung und Ausblick

Der hier vorgestellte Ansatz zur integrierten Automotive Software-Entwicklung mit AutoFOCUS basiert auf einer integrierten Beschreibungstechnik mit alternativen Sichten und einem einheitlichen Berechnungsmodell, die den Entwicklungsprozess für Automotive-Software auf drei ausgewählten Abstraktionsebenen unterstützt. Jede Abstraktionsebene wird dabei mit geeigneten Beschreibungstechniken in Bezug gesetzt. Gegenüber vorausgegangenen Publikationen wurden spezifisch für die Domäne Automotive einige neue Beschreibungstechniken eingeführt, insbesondere um die Darstellung regelungstechnischer Blockdiagramme, die explizite Modellierung von Betriebsmodi, sowie die späten Phasen im Prozess in Richtung Deployment zu unterstützen.

Die Integration von Clocks und Sampling-Operatoren als Beschreibungsmittel in AutoFOCUS erlaubt statische Aussagen über die Frequenz (Clocks) von Signalen, die mit einem komfortablen Inferenzmechanismus erzielt werden. Über die Bedeutung als reines Mittel der Modellierung hinaus sind Clocks wesentliche Informationen für den Übergang auf die LA/TA-Ebene, da zum einen für die Kommunikation zwischen Clustern von Clocks abhängige Konsistenzbedinungen für Verzögerungen existieren können, zum anderen eine effiziente Partitionierung des Modells in Cluster die Kenntnis der Frequenzen voraussetzt.

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. Als technische Basis ist dabei insbesondere die Transformations- und Logiksprache ODL zu nennen: auf ODL basierende Transformationen können den Entwickler bei vielen Standardaufgaben (Aufspalten von komplexen Kanälen, Einbinden von Bibliotheksfunktionen, usw.) entlasten. Mittels automatisierter Konsistenzprüfungen auf dem homogen definierten AutoFOCUS-Metamodell kann die Konsistenz von Designs gegenüber heterogenen Ansätzen wesentlich erhöht werden.

Der Werkzeugprototyp AutoFOCUS 2 bietet bereits ausreichende Möglichkeiten für die Bearbeitung von Fallstudien. Als Erweiterung des Werkzeugs ist aktuell die Hinzunahme komplexerer Modelltransformation geplant. Ein mögliches Anwendungsfeld sind dabei Reengineering-Schritte, die insbesondere im Umgang mit importierten Modellen aus anderen Werkzeugen (z. B. MATLAB Simulink, ASCET) auftreten.

Literatur

1. Das Projekt EAST-EEA – Eine middlewarebasierte Softwarearchitektur für vernetzte Kfz-Steuergeräte (2003) In: VDI-Kongress Elektronik im Kraftfahrzeug, VDI Berichte, vol 1789, Baden-Baden
2. Bauer A, Romberg J (2004) 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
3. Benveniste A, Caspi P, Guernic PL, Halbwachs N (1993) Data-Flow Synchronous Languages. In: REX School/Symposium, pp 1–45
4. Braun P, Lötzbeyer H, Schätz B, Slotosch O (2000) Consistent Integration of Formal Methods. In: Graf S, Schwartzbach M (eds) Tool and Algorithms for the Construction and Analysis of Systems (TACAS 2000), LNCS, vol 2280. Springer Verlag
5. Braun P, von der Beeck M, Rappl M, Schröder C (2002) Automotive Software Development: A Model-Based Approach. In: Congress of Automotive Engineers, SAE Transactions Paper
6. Broy M, Huber F, Schätz B (1999) AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. Informatik Forschung und Entwicklung 14(3):121–134
7. ETAS Engineering Tools GmbH (2001) ASCET-SD Benutzerhandbuch
8. Gelernter D, Carriero N (1992) Coordination languages and their significance. Communications of the ACM 35(2):97–107
9. The MathWorks Inc. (2000) Using Simulink
10. Object Management Group OMG (2004) `www.uml.org`. Unified Modeling Language: Superstructure. Version 2.0. OMG Adopted Specification ptc/03-08-02
11. OMG Object Management Group (2003) `www.uml.org`. UML Profile for Schedulability, Performance, and Time Specification. Version 1.0 formal/03-09-01
12. Romberg J, Bauer A (2004) Loose Synchronization of Event-Triggered Networks for Distribution of Synchronous Programs. In: ACM Conference on Embedded Software (EMSOFT)
13. Schätz B (2001) The ODL operation definition language and the AutoFocus/Quest application framework AQuA. Technical Report TUM-I0111, TU München
14. Schätz B, Braun P, Huber F, Wisspeintner A (2005) 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
15. Selic B, Gullekson G, Ward P (1994) Real-Time Object Oriented Modelling. Wiley

Andreas Bauer ist wissenschaftlicher Mitarbeiter der Technischen Universität München und arbeitet am Lehrstuhl fuer Software & Systems Engineering von Prof. Manfred Broy. Neben Verfahren zur effizienten Uebersetzung von Programmiersprachen richten sich seine Forschungsinteressen vorwiegend auf die Entwicklung und Diagnose eingebetteter, fehlertoleranter Software-Systeme.

Jan Romberg hat sein Studium der Elektrotechnik an den Universitäten Dresden und Karlsruhe im Jahr 2000 mit Dipl.-Ing. abgeschlossen. Nach einem einjährigen Forschungsaufenthalt bei Bell Labs, Naperville, USA ist er seit 2002 als wissenschaftlicher Angestellter am Lehrstuhl für Systems & Software Engineering (Prof. Broy) der TU München mit dem Schwerpunkt Entwurf eingebetteter Systeme tätig.

Bernhard Schätz Nach dem Studium der Informatik an der Technischen Universität München 1983–1989 arbeitete Bernhard Schätz im Rahmen eines Stipendiats der Fa. Siemens AG, sowie als wiss. Mitarbeiter am Sonderforschungsbereich 342 „Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen" der DFG im Bereich Spezifikation und Verifikation reaktiver Systeme. Seit seiner Promotion 1998 zum Doktor rer. nat. an der Technischen Universität München arbeitet er als wiss. Assistent am Lehrstuhl für Software & Systems Engineering (Prof. Broy) als Leiter des Kompetenzzentrums „Modellbasierte Entwicklung" mit dem Schwerpunkt „Eingebettete Systeme". Neben seinen akademischen Tätigkeiten ist er Mitgründer und Aufsichtsrat der Validas AG.