Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler

Torsten Klein, Mirko Conrad, Ines Fey, Matthias Grochtmann

DaimlerChrysler AG, Forschung und Technologie, Methoden und Tools
Alt-Moabit 96a, 10559 Berlin
Torsten.Klein, Mirko.Conrad, Ines.Fey, Matthias.Grochtmann@DaimlerChrysler.com

1 Software im Automobil

Generelle Trends

Wer im Wettbewerb auf dem weltweiten Fahrzeugmarkt bestehen will, setzt auf Software im Automobil. Nach Aussagen der Mercedes-Benz Entwicklung treibt Elektronik künftig etwa 80 Prozent aller Innovationen voran, wobei diese wiederum zu 90 Prozent durch Softwarefunktionen bestimmt werden. Aktuelle Beispiele sind die *Sensotronic Brake Control (SBC)* in der aktuellen Mercedes-Benz E-Klasse, optimierte Motor- und Antriebssteuerungen sowie die erstmals in der Mercedes-Benz S-Klasse eingesetzte elektronische Abstandsregelung *Distronic*.

Der Trend geht aber nicht nur zu immer mehr Software im Auto, die Systeme werden darüber hinaus auch immer komplexer. Heute gibt es mehr und mehr Funktionen, die ihre Fähigkeiten erst im Zusammenspiel vieler Teilfunktionen und Steuergeräte entwickeln, wie z.B. das *Electronic Stability Program (ESP)*. Aktuell werden in der Mercedes-Benz S-Klasse mehr als 80 Steuergeräte verbaut, die insgesamt über 600.000 Programmzeilen beinhalten und über drei Bussysteme vernetzt sind, die über 100 Busnachrichten und mehrere 100 Signale transportieren.

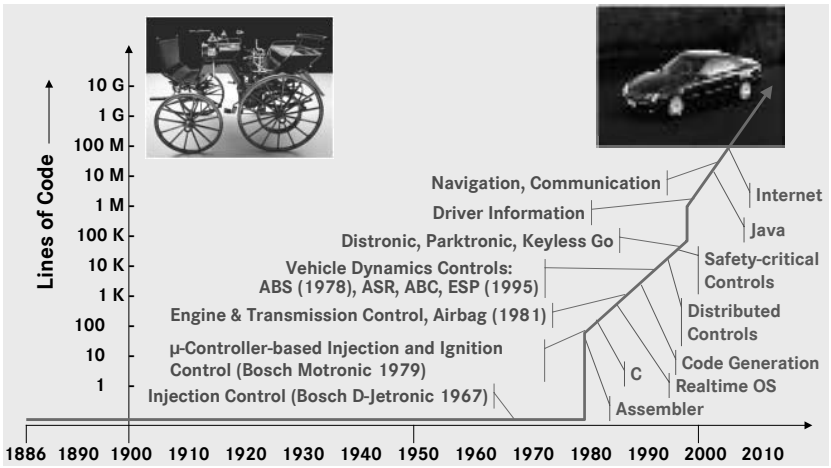

Abbildung 1: Historie softwarebasierter Funktionen im Automobil

Mit der Zahl elektronischer Steuerungen im Fahrzeug steigen zwangsläufig auch die durch die Elektronik und damit durch die Software verursachten Entwicklungskosten. In aktuellen Mercedes-Benz Fahrzeugen liegt der auf softwarebasierte Systeme entfallende Wertschöpfungsanteil bei rund 25 Prozent. Neuartige elektronische Assistenzsysteme erhöhen diesen Beitrag auf schätzungsweise 40 Prozent.

Es ist zu erwarten, dass die Bedeutung von Software im Automobil auch in Zukunft weiter zunehmen wird.

Paradigmenwechsel hin zur modellbasierten Entwicklung

Voraussetzungen für die effiziente und fehlerfreie Entwicklung derartiger softwarebasierter elektronischer Systeme sind ein definierter Entwicklungsprozess, der laufend weiterentwickelt werden sollte, sowie der Einsatz praxisgerechter, leistungsfähiger Methoden und Werkzeuge [SZ03]. Bei der Entwicklung von Softwareanteilen für Steuergeräte vollzieht sich seit einigen Jahren ein Paradigmenwechsel. Dieser ist durch einen Übergang von der klassischen Programmentwicklung hin zu modellbasierten Techniken gekennzeichnet, bei denen Modelle eine zentrale Rolle spielen.

Unserer Erfahrung nach ist es wichtig, darauf hinzuweisen, dass wir *Modelle* in diesem Kontext als konstruktive, meist ausführbare Beschreibungen von durch Software zu realisierenden Algorithmen einer Steuerung bzw. Regelung für eine Fahrfunktion unter Verwendung grafischer Notationselemente verstehen. In Abgrenzung zu Fahrzeugmodellen, Umgebungsmodellen, Fahrermodellen oder Verkehrsmodellen bezeichnen wir diesen Modelltyp als *Funktionsmodell*. Ein Funktionsmodell kann in verschiedenen Entwicklungsphasen jedoch unterschiedliche Eigenschaften haben, da mit dessen Erstellung unterschiedliche (Teil-)Ziele, wie z.B. Funktionsdesign oder Codegenerierung, verfolgt werden.

Fokus des Beitrags

Im Kontext softwarebasierter Systeme können im Fahrzeug vier Domänen unterschieden werden: *Telematik*, *Innenraum*, *Fahrwerk* sowie *Motor- und Antriebsstrang*. Bei den folgenden Betrachtungen wollen wir uns auf die drei letztgenannten Fahrzeugdomänen beschränken, da diese hinsichtlich des oben beschriebenen Paradigmenwechsels bei der Software-Entwicklung eine Reihe von Parallelitäten aufweisen. Dabei ist insbesondere die Rolle von Funktionsmodellen als die zentralen Entwicklungsartefakten sowie die Nutzung der integrierten Notation und Werkzeugumgebung *Matlab/Simulink/Stateflow* [MLSLSF] bestehend aus datenflussorientierten Blockdiagrammen und Zustandsautomaten ein gemeinsames, kennzeichnendes Merkmal.

Dem bei innovativen Fahrzeugherstellern entstehenden Know-how Bedarf auf dem vergleichsweise jungen Gebiet der modellbasierten Entwicklung wird bei DaimlerChrysler durch Forschungsvorhaben und entwicklungsbegleitende Projekte zu derartigen Fragestellungen Rechnung getragen. Der vorliegende Beitrag ist aus diesem Kontext entstanden und gibt die von den Autoren gesammelten Erfahrungen wieder. Er kann daher nur einen – unserer Einschätzung nach jedoch sehr wichtigen – Ausschnitt aus dem gesamten Spektrum der Software-Entwicklung für eingebettete Fahrzeugsysteme im DaimlerChrysler Konzern abdecken, ohne dabei einen Anspruch auf Vollständigkeit zu erheben.

2 Modellbasierte Software-Entwicklung bei DaimlerChrysler: Überblick und aktuelle Arbeiten

In diesem Abschnitt wird das typische Vorgehen bei der modellbasierten Entwicklung von eingebetteter Fahrzeugsoftware bei DaimlerChrysler dargestellt: von der Spezifikation über Funktionsdesign und -implementierung bis hin zu Verifikation und Validation. Zu jedem Themenfeld werden einige aktuelle Arbeitsschwerpunkte der DaimlerChrysler Forschung genannt. Abschließend werden wesentliche Herausforderungen angesprochen, die in der Integration und Feinabstimmung des modellbasierten Vorgehens innerhalb des gesamten, werkzeuggestützten Entwicklungsprozesses bestehen.

Anforderungsspezifikation

Die Entwicklung beginnt – wie allgemein üblich – damit, die geforderten Eigenschaften der zu erstellenden Software und ihr gewünschtes Verhalten zu spezifizieren: als realisierungsunabhängige Vorgabe für die weiteren Entwicklungsschritte. Dabei wird typischerweise eine textuelle Anforderungsspezifikation erstellt. In bestimmten Fällen kann diese um ein graphisches *Systemverhaltensmodell* ergänzt werden.

Die Anforderungsspezifikation beschreibt sowohl die geforderten funktionalen als auch nicht funktionalen Eigenschaften der zu erstellenden Software. Diese wird üblicherweise von den Fahrzeugsystementwicklern in Abstimmung mit anderen Experten erstellt. Wesentliches Ziel ist dabei eine detaillierte Beschreibung des Verhaltens der zu entwickelnden Funktion im Fahrzeug auf Systemebene und nicht nur der eingebetteten Software; d.h., es wird die später für den Fahrer erlebbare Funktion beschrieben. Dadurch wird die Spezifikation anschaulicher und der Kundennutzen rückt in den Vordergrund. Die ausschließliche Beschreibung des Softwareanteils erfolgt erst während des Funktionsdesigns. Die Anforderungsspezifikation ist im Sinne eines Lastenhefts bindende Vorgabe für die weiteren Entwicklungsschritte, gegebenenfalls auch für einen beteiligten Zulieferer.

Von der Form her ist die Anforderungsspezifikation in der Regel ein strukturierter Text, d.h., Anforderungen werden als vereinzelte Aussagen beschrieben. Dadurch wird die Anforderungsspezifikation in kleine, handhabbare Einheiten zerlegt. Das Dokument selbst wird nach Themenfeldern untergliedert, insbesondere nach funktionalen und nicht funktionalen Anforderungen. Verknüpfungen können genutzt werden, um Beziehungen zwischen Anforderungen explizit darzustellen. Ergänzend können z.B. Abbildungen und Use Cases verwendet werden. Die so formalisierte Struktur der Anforderungsspezifikation erlaubt es, Anforderungen zu priorisieren und ihre Umsetzung und Überprüfung zu verfolgen. Zur Erfassung und Verwaltung von Anforderungen hat sich bei DaimlerChrysler das Werkzeug DOORS der Firma Telelogic [DOORS04] durchgesetzt.

In den meisten Entwicklungsprojekten wird ausschließlich mit textuellen Anforderungen spezifiziert. Vorteile dieser Form der Spezifikation sind zum Beispiel die Flexibilität der natürlichen Sprache sowie die gute Lesbarkeit auch für Nicht-Techniker. Darüber hinaus erleichtert diese natürlichsprachliche Beschreibungsform den Einstieg in die Spezifikationsphase. Dennoch kann es sich lohnen, ergänzend oder teilweise sogar alternativ, das Verhalten der zu entwickelnden Funktion mit einer graphischen Modellierungssprache zu spezifizieren. Ein solches Systemverhaltensmodell ist zum Beispiel nützlich, wenn es bei einem großen, komplexen System vor allem auf die richtige Interaktion von Teilfunktionen ankommt, wenn man viele systeminterne Zustände hat, die verknüpft sind, oder wenn man

überprüfen will, ob das Systemverhalten vollständig und konsistent beschrieben ist; letztlich also vor allem dann, wenn es mehr auf die Gesamtsicht ankommt als auf Details. In solchen Fällen ist Graphik oft angemessener als Text. Weiterhin kann das Systemverhaltensmodell auch simulierbar gestaltet werden, so dass man das grundsätzliche Systemverhalten bereits früh erproben kann.

Deshalb werden in verschiedenen Projekten solche Systemverhaltensmodelle in Ergänzung zu den textuellen Anforderungen für die Spezifikation eingesetzt. Prinzipiell können dabei die verschiedensten Modellierungsansätze und -sprachen zum Einsatz kommen. Erfahrungsgemäß sind für Fahrzeugsysteme auf dieser Ebene vor allem zustandsbasierte Ansätze angemessen. Da mit Stateflow in Ergänzung zu Matlab/Simulink eine zustandsorientierte Notation für die spätere Modellierung bereits verfügbar ist, bietet es sich an, Zustandsübergangsdiagramme in Stateflow auch für das Systemverhaltensmodell zu verwenden.

Die Ein- und Ausgaben des Systemverhaltensmodells liegen auf der Ebene von Fahreraktionen, Ausgaben an den Fahrer, Fahrzeugzustand und Umgebung. Das Systemverhaltensmodell soll das generelle Ein- und Ausgabeverhalten sowie die verschiedenen Zustände des Systems und ihr Zusammenspiel beschreiben, jedoch noch nicht die Funktion vollständig realisieren. Folglich sind informelle Anteile, zum Beispiel textuell beschriebene Übergangsaktionen, und Vereinfachungen, zum Beispiel bevorzugt boolesche Ein- und Ausgaben, üblich. Trotzdem ist es oft möglich, das Systemverhaltensmodell simulierbar zu gestalten, so dass die oben genannten Aspekte der spezifizierten Funktion erprobt werden können.

Inwieweit das Systemverhaltensmodell die textuelle Anforderungsspezifikation nur ergänzen oder teilweise ersetzen sollte, ist projektspezifisch festzulegen. Stets notwendig ist eine separate Beschreibung für die nicht funktionalen Anteile, wie z.B. Anforderungen an Ressourcen oder zeitliches Verhalten.

Anforderungsspezifikation und gegebenenfalls Systemverhaltensmodell legen gemeinsam eindeutig fest, *was* entwickelt werden soll. Damit ist die folgende Phase *Funktionsdesign und -implementierung* (das *wie*) optimal vorbereitet.

Forschungsarbeiten bei DaimlerChrysler im Bereich der Spezifikation untersuchen beispielsweise die Verknüpfung von Anforderungen untereinander, sowie die Verknüpfung mit anderen Entwicklungsgegenständen, wie z.B. Modellen oder Tests. Ein weiteres Thema ist das Managen unterschiedlicher Varianten einer Funktion auf Anforderungsebene, um zum Beispiel Anforderungen verschiedener Fahrzeugbaureihen integriert und aufeinander abgestimmt verwalten zu können [EAST04]. Besondere Aufmerksamkeit richtet sich auch auf Systemverhaltensmodelle: hier wird beispielsweise der Bezug zwischen Anforderungen und Systemverhaltensmodell untersucht, sowie die Integration des Systemverhaltensmodells in das Entwicklungsvorgehen [BD03, WW03].

Funktionsdesign und -implementierung

Charakteristisch für die modellbasierte Entwicklung ist der durchgehende Einsatz von ausführbaren Modellen in den Phasen Funktionsdesign und -implementierung. Die zu realisierende Steuergerätefunktion tritt dabei in verschiedenen, aufeinander aufbauenden Repräsentationsformen auf: Ein zunächst logisches Funktionsmodell wird dabei unter Realisierungsaspekten ergänzt, überarbeitet und schließlich mittels Codegeneratoren in Programmcode einer imperativen Programmiersprache (meist C) überführt.

Am Anfang dieser *Modellevolution* steht in der Regel ein sog. *physikalisches Modell*[1]. Es definiert die Schnittstelle der zu entwickelnden Software-Funktion und beschreibt deren Außenverhalten in ausführbarer Form. Insbesondere enthält es bereits die notwendigen Steuerungs- und Regelalgorithmen. Das physikalische Modell kann nach beliebigen Kriterien strukturiert sein. Zweck des physikalischen Modells ist es, die zu entwickelnden Algorithmen darzustellen, ohne dabei bereits Realisierungsdetails beachten zu müssen. Die Beschreibung der Algorithmen erfolgt daher üblicherweise unter Verwendung von Gleitkomma-Arithmetik.

Aus Effizienzgründen und aufgrund der Tatsache, dass im physikalischen Modell ggf. von der Zielplattform abstrahiert wurde, kann das physikalische Modell nicht direkt als Basis für die Ableitung von Code für Seriensteuergeräte dienen. Daher wird es unter Realisierungsgesichtspunkten überarbeitet (z.B. Aufteilung von Funktionsteilen auf unterschiedliche Tasks) und um die notwendigen Implementierungsdetails (z.B. Datentypen und Skalierungsinformationen) angereichert. Ergebnis ist ein *Implementierungsmodell*, das alle für die Codegenerierung notwendigen Informationen enthält und als Basis für die Codierung der eingebetteten Software dient. Diese ist durch den Einsatz von Werkzeugen zur automatischen *Codegenerierung* (z.B. TargetLink [TargetLink] oder Real-Time Workshop [MLSLSF]) bereits weitgehend automatisierbar, der Übergang von der grafischen Modellierung zum Programmcode verläuft ohne Brüche. Ist der Einsatz dieser Techniken in einem bestimmten Projektkontext nicht möglich, kann alternativ auch eine vollständig manuelle Implementierung anhand des Funktionsmodells bzw. eine Integration mit handcodierten Teilfunktionen erfolgen. Die aus den Funktionsmodellen gewonnenen Codeteile werden in der Regel mit Standard-Softwaremodulen und der notwendigen Rahmensoftware integriert und sind unter Standard-Betriebssystemen wie OSEK [OSEK] lauffähig.

In einem konkreten Entwicklungsprojekt kann es beliebige Zwischenstufen bei der Modellierung geben. Die beiden hier skizzierten Modellarten repräsentieren die typischerweise im von uns betrachteten modellbasierten Entwicklungsprozess erstellten Modelle.

Spezifisch für diese Art der Entwicklung ist, dass ausführbare Modelle sowohl die gewünschte Funktion der Software spezifizieren (wenn auch detaillierter als ein Systemverhaltensmodell), ein Design für die Umsetzung der Funktion vorgeben und schließlich auch die Grundlage für die Implementierung bilden. Somit haben die für die Systemverhaltens- und Funktionsmodellierung verwendeten Simulink/Stateflow-Modelle sowohl *Spezifikations-* als auch *Design- und Implementierungscharakter*. Im Vergleich zur klassischen Software-Entwicklung mit einer klaren Phasentrennung ist bei der modellbasierten Entwicklung ein stärkeres Zusammenwachsen der konstruktiven Entwicklungsphasen Spezifikation, Design und Implementierung zu erkennen.

Aktuelle Arbeiten bei DaimlerChrysler befassen sich beispielsweise mit der Modellierung vernetzter Systeme [HWK+03], der Absicherung der Codegenerierung [SC03] und der Gestaltung von flexiblen Kooperationsszenarien zwischen Fahrzeugherstellern (OEMs) und Zulieferern bei der Modellierung und Codegenerierung.

---

[1] Der Begriff physikalische Modell wird hier *nicht* für die physikalisches Abbildung der Umgebung im Streckenmodell, sondern für ein frühes Funktionsmodell verwendet.

Verifikation und Validation

Auch im Rahmen der modellbasierten Entwicklung ist es notwendig, die entstehende Software einer geeigneten Kombination von Verifikations- und Validationsmaßnahmen zu unterziehen, um Fehler zu finden und Vertrauen in die korrekte Funktionsweise der Software zu gewinnen, wobei der dynamische Test im Mittelpunkt der Verifikations- und Validationsaktivitäten steht. Die modellbasierte Entwicklung eröffnet hier neue Möglichkeiten und Synergiepotenziale für den Testprozess, die unter Effizienzgesichtspunkten konsequent genutzt werden sollten. Ein solcher, mit der modellbasierten Entwicklung eng verzahnter Testprozess, der eine Kombination unterschiedlicher, sich gut ergänzender Testverfahren umfasst und dabei das ausführbare Modell in seinen verschiedenen Ausprägungen als Informationsquelle für den Test benutzt, wird im hier betrachteten Kontext als *modellbasierter Test* bezeichnet.

Im Gegensatz zur klassischen Softwareentwicklung, bei der für eine Funktionalität in der Regel nur ein ausführbares Ergebnis, der Programmcode, erstellt wird, ergeben sich in der modellbasierten Entwicklung durch die Modellevolution zu einer Funktionalität routinemäßig mehrere ausführbare Artefakte (z.B. Funktionsmodelle, Implementierungsmodelle, Autocode). Kombiniert man diese mit den unterschiedlichen Ausführungsplattformen (z.B. Desktop-PC, Evaluierungsboard, Steuergerät) und Umgebungen (z.B. simuliertes Umgebungsmodell, Prüfstand, Fahrzeug) ergeben sich eine Vielzahl sogenannter *Testmöglichkeiten*. Dabei müssen für einen gründlichen Test der Funktionalität nicht alle Testmöglichkeiten zwangsläufig ausgeschöpft werden.

Eine Teststrategie für die modellbasierte Entwicklung sollte daher möglichst komplementäre Testverfahren vorschlagen und die Auswahl einer geeigneten Teilmenge von Testmöglichkeiten unterstützen, so dass eine hohe Fehleraufdeckungswahrscheinlichkeit erreicht wird.

Den Ausgangspunkt einer solchen effektiven und effizienten Teststrategie bildet die systematische Ableitung von funktionalen Tests und deren Anwendung auf das physikalische Modell. Falls notwendig, sind solange zusätzlich Strukturtests zu ermitteln, bis ein geeignetes Strukturtestkriterium auf Modellebene erfüllt ist. Ist so eine ausreichende Testabdeckung für das physikalische Modell gewährleistet, können die Funktions- und Strukturtestszenarien im Rahmen von Back-to-back-Tests für den Test des Implementierungsmodells und des Autocodes wiederverwendet werden, um die funktionale Äquivalenz zwischen dem physikalischen Modell und den daraus abgeleiteten Repräsentationsformen nachzuweisen.

Die systematische Ermittlung von Testszenarien unter funktionalen wie unter strukturellen Gesichtspunkten sowie der vergleichende Test verschiedener Repräsentationsformen ermöglichen eine effektive Aufdeckung softwarebezogener Fehlertypen und prüfen die fehlerfreie Überführung des Modells in C-Code und dessen Einbettung in das Steuergerät.

Aktuelle Forschungsarbeiten bei DaimlerChrysler beschäftigen sich mit der Entwicklung von funktionalen Testverfahren für den modellbasierten Test, wie der Klassifikationsbaum-Methode für eingebettete Systeme CTM/ES [Con01] und Time-Partition-Testing TPT [Leh00], der Untersuchung struktureller Überdeckungskriterien auf Modellebene [BCS+03], der Automatisierung der Testauswertung im Rahmen von Back-to-back-Tests [CFP+03], dem Einsatz evolutionärer Testverfahren [BPS03] sowie der durchgängigen Automatisierung des modellbasierten Testprozesses mit der Testumgebung MTest [LBE+04].

Herausforderungen der modellbasierten Entwicklung

Mit dem Übergang von der klassischen Softwareentwicklung hin zu einem modellbasierten Vorgehen werden in der Praxis häufig Artefakte der frühen Entwicklungsphasen als „im Modell enthalten“ angesehen. Von uns in realen Entwicklungsprojekten gesammelte Erfahrungen zeigen jedoch, dass dieser ausschließlich modellzentrierte Ansatz wichtige Aspekte der Entwicklung nicht ausreichend berücksichtigt.

Soll ein systematischer Entwicklungsprozess mit nachvollziehbaren und ggf. wiederverwendbaren Erkenntnissen und Artefakten verfolgt werden, ist unserer Erfahrung nach die separate Erfassung von textuellen Anforderungen (vgl. Abschnitt „Anforderungsspezifikation“) auch im Kontext der modellbasierten Entwicklung unverzichtbar. Darüber hinaus erzwingen Randbedingungen aus externen Vorgaben in vielen Fällen ohnehin eine separate Anforderungsphase, aus der ein oder mehrere textbasierte Anforderungsspezifikationen als verbindliche Artefakte hervorgehen.

Zur Erhöhung der Qualität der Funktionssoftware hat sich außerdem die Ergänzung der Tests, die ausschließlich unter Betrachtung von Modellen auf verschiedenen Abstraktionsebenen erfolgen, durch weitere aus dem Kontext der klassischen Software-Entwicklung bekannte Testverfahren als sinnvoll erwiesen (vgl. Abschnitt „Verifikation und Validation“).

Innerhalb eines integrierten Entwicklungsprozesses mit einem wohldefinierten Management der Anforderungen bietet sich weiterhin die Möglichkeit, Verknüpfungen zwischen textuellen Anforderungen und Modellelementen explizit zu definieren. Vorteil dieser Links ist die Verfolgbarkeit von den Anforderungen bis zur Umsetzung, die sowohl die Einschätzung lokaler Änderungen ermöglicht als auch bei der Definition und der Erkennung der Notwendigkeit von Wiederholungen anforderungsbasierter Testfälle unterstützt.

Eine Herausforderung liegt in der effizienten Erstellung und Nutzung der Verlinkung und damit der engen Integration der Entwicklungsphasen. Obwohl einige technische Lösungen dafür bereits verfügbar sind, fehlt eine methodische Herangehensweise, die den Prozess der Verknüpfungsfindung klärt. Unserer Erfahrung nach hat sich hier ein Ansatz bewährt, der die explizite Identifizierung einzelner Informationseinheiten des spezifischen Entwicklungsprozesses sowie deren Beziehung untereinander in den Mittelpunkt stellt. Wir nutzen für diese Herangehensweise ein Informationsmodell.

Als *Informationsmodell* bezeichnen wir die Darstellung der in den verschiedenen Entwicklungsphasen entstehenden Entwicklungsobjekte sowie deren Abhängigkeiten und Ordnungen (z.B. Hierarchien). Instanzen eines Informationsmodells liefern demzufolge eine statische Sicht auf den zu einem bestimmten Zeitpunkt existierenden Zustand einer Systementwicklung, d.h. auf die in einem Projekt vorhandenen Entwicklungsdaten.

Für die modellbasierte Entwicklung werden beispielsweise Entwicklungsobjekte, wie Systemelement, Funktionsmodell und dessen Subsysteme bis hin zu Basisblöcken und Signalen, die in Subsystemen enthalten sind, betrachtet. Diese Art der hierarchischen Strukturierung ist für die Definition von Metamodellen durchaus üblich (vgl. [UML20]).

Informationsmodelle wurden in der Vergangenheit hauptsächlich im Kontext des Managements von Anforderungen untersucht [JHN+99, WW02], und auf diesen Anwendungskontext beschränkt punktuell in der Praxis eingeführt. Darüber hinaus wurden Informationsmodelle verwendet, um den Austausch von Modellinformationen zwischen verschiedenen Modellierungs- und Simulationswerkzeugen zu erleichtern [SM01]. Der Tatsache, dass

sich zunehmend die modellbasierte Funktionsentwicklung etabliert, kann durch die Erweiterung dieses Ansatzes auf *alle* im Entwicklungsprozess relevanten Artefakte Rechnung getragen werden. Dies hat als angenehmen Nebeneffekt außerdem die Verbindung der genannten Arten von Teil-Informationsmodellen zur Folge.

Neben einer methodischen Fundierung der Integration verschiedener Entwicklungsartefakte kann ein derartiges Informationsmodell auch für die Evaluierung technischer Lösungen zur Abbildung des Entwicklungsprozesses dienen bis hin zu einer Bewertung von Werkzeuglösungen.

Wesentliche Voraussetzung für den praktischen Einsatz von Verknüpfungen ist zum Einen eine geeignete Werkzeugunterstützung und zum Anderen eine explizite und von allen beteiligten akzeptierte Integration der Erstellung und des Änderungsmanagements von Verknüpfungen zwischen Anforderungen und dem Modell in den Entwicklungsprozess.

Wie bereits erwähnt, stellt die Verknüpfung von Anforderungen und Modellen eine nach wie vor nicht zufriedenstellend gelöste Herausforderung dar. Exemplarisch für diesen Anwendungsfall haben wir aus dem Informationsmodell der modellbasierten Entwicklung die Randbedingungen und Anforderungen einer entsprechenden Werkzeugunterstützung identifiziert und anhand dieser Vorgaben die Kopplung der Werkzeuge zur Modellierung (Simulink/Stateflow) und zum Management von Anforderungen (DOORS [DOORS04]) über das Requirements-Management-Interface [RMI04] evaluiert. Das Ergebnis dieser Untersuchung zeigt sehr anschaulich, dass die Teilbereiche des Informationsmodells, die den einzelnen Werkzeugdomänen entsprechen, also die Verwaltung textueller Anforderung bzw. die Modellerstellung, sehr gut abgedeckt werden. Große Lücken zeigten sich aber in der werkzeugseitigen Abbildung von Verknüpfungen zwischen diesen Domänen, dem Kernziel des Requirements-Management-Interface.

3 Modellbasierte Systementwicklung in der Automobilindustrie: Eine domänenspezifische Ausprägung des MDA der OMG?

Das modellgetriebene Entwicklungsparadigma wird international derzeit stark von der Initiative der *Object Management Group (OMG)* bzgl. der *Model Driven Architecture (MDA)* in Verbindung mit der *Unified Modeling Language (UML)* geprägt. Oberflächlich betrachtet scheint dieses Paradigma nur wenig mit der Entwicklung von Software in der Automobilindustrie zu tun zu haben – insbesondere wenn man auf die Begriffswelten schaut, die gänzlich verschieden sind.

Dies wäre nachvollziehbar, da der MDA seinen Ursprung in der Entwicklung kommerzieller großer – nicht zwingend eingebetteter – verteilter Systeme hat, bei denen die Aspekte der Verteilung und damit einhergehend die Middleware von zentraler Bedeutung sind. Betrachtet man jedoch die Bedeutung der in den vorhergehenden Abschnitten besprochenen Artefakte losgelöst von den mit Ihnen verknüpften Begrifflichkeiten, lassen sich unseres Erachtens eine Reihe von interessanten Parallelen aufzeigen, die darauf hindeuten, dass in der Automobilindustrie wesentliche Gedanken des MDA [MDA01] bereits seit geraumer Zeit operativ umgesetzt werden:

- Das physikalische Modell hat zum Ziel, Steuerungs- und Regelungsalgorithmen unabhängig von der im Serienfahrzeug letztendlich verfügbaren Steuergeräte-Infrastruktur zu beschreiben. Dieser Ansatz ist vergleichbar mit den Eigenschaften, die durch die

Betrachtung von *Platform Independent Models (PIM)* im Rahmen des MDA verfolgt werden.

- Das Implementierungsmodell berücksichtigt die speziellen Eigenschaften von Ressourcen und realen Schnittstellen (CPU, Busse, Speicher, Kommunikationsinfrastruktur) und ist auf deren Verwendung und optimale Nutzung ausgerichtet. Hier lassen sich Parallelen zu den typischen Eigenschaften eines *Platform Specific Models (PSM)* erkennen. Inwieweit es für die im MDA genannten - hauptsächlich auf den Middleware-Aspekt bezogenen – „Plattformen" heutzutage unmittelbare Entsprechungen im Automotive Umfeld gibt, ist Gegenstand aktueller Untersuchungen.
- Automatische bzw. teil-automatisierte Modelltransformationen sind zentraler Gegenstand des MDA Gedankens, von dem sich die OMG einen entscheidenden Effizienzgewinn verspricht. Im durch Matlab/Simulink/Stateflow getriebenen modellbasierten automotive Entwicklungsprozess sind *Modelltransformationen* heute gängige Praxis. Dies betrifft sowohl die Transformation zwischen Modellen untereinander (z.B. zwischen Implementierungs- und physikalischem Modell) als auch die automatische Generierung von Programmcode für Steuergeräte – letztendlich auch eine Modelltransformation.
- Die von der OMG getriebene formale Fundierung von Notationen durch Meta-Modelle wird derzeit von der Automobilindustrie als relevantes Tätigkeitsfeld erkannt in Form von allgemeinen Informations-Meta-Modellen für die Elektronik-Entwicklung umgesetzt [EAST04].

In Zukunft stellt sich die Frage, inwiefern die Automobilindustrie von den Entwicklungen des MDA und der UML profitieren kann. Mögliche Vorteile könnten aus unserer Sicht folgende sein:

- im Hinblick auf Funktionalität und semantische Fundierung ausgereifte Entwicklungswerkzeuge (durch eine Erweiterung des Marktpotenzials für die Werkzeughersteller im Vergleich zu rein durch die Automobilindustrie getriebenen Speziallösungen)
- Zugang zu einem größeren Kreis von Experten (durch Verwendung einer weit verbreiteten Notation, Methodik und Werkzeuglandschaft)
- Erschließung von Synergiepotenzialen über verschiedene Anwendungsdomänen für eingebettete Systeme (z.B. Luft- und Raumfahrt, Verteidigungstechnik, Bahntechnik, Telekommunikation)

Aktuell sehen wir einen nicht zu vernachlässigenden Aufholbedarf für die MDA-bezogenen Technologien, da der von uns beschriebene modellbasierte Entwicklungsansatz für softwarebasierte Fahrzeugfunktionen bei DaimlerChrysler stark auf einer hochspezialisierten um Simulink/Stateflow herum aufgebauten Methoden und Werkzeuglandschaft basiert. Dies zeigt sich in speziellen Ansätzen zur Codegenerierung, in speziellen Frameworks zur Reglerentwicklung, Messtechnik und verteilten Modellierung sowie in spezifischen Funktionsblockbibliotheken, die auf die besonderen Bedürfnisse der Steuerungs- und Regelungstechnik abgestimmt sind. Um MDA-bezogene Technologien in diesem Anwendungsumfeld zu etablieren, ist eine Berücksichtigung dieser Bedürfnisse zwingend erforderlich.

Ob und in welcher Form eine Abstimmung der allgemeinen, von der OMG vertretenden modellbasierten Vorgehensweisen, mit der modellbasierten Entwicklung von Fahrzeugsoftware stattfinden wird, ist schwer vorherzusagen und wird unserer Einschätzung nach maßgeblich durch den Einfluss organisatorischer und betriebswirtschaftlicher Randbedin-

gungen entschieden werden. Eine kurzfristige Übernahme der OMG/MDA/UML-basierten Ansätze für die Serienentwicklung von eingebetteten, softwarebasierten Fahrfunktionen im DaimlerChrysler Konzern sehen wir derzeit nicht.

4 Referenzen

[BCS+03] A. Baresel, M. Conrad, S. Sadeghipour, J. Wegener: The Interplay between Model Coverage and Code Coverage. Proc. of 11. European Int. Conf. on Software Testing, Analysis and Review (EuroSTAR '03), Amsterdam (NL), Dez. 2003

[BD03] K. Buhr, H. Dörr: Requirements Driven Quality Assurance. Proc of 9th Int. Council on Systems Engineering (INCOSE 2003), Washington (USA), 2003.

[Con01] M. Conrad: Beschreibung von Testszenarien für Steuergerätesoftware – Vergleichskriterien und deren Anwendung. In: [ElKfz01], S. 381-398

[CFP03] M. Conrad, I. Fey, H. Pohlheim: Automatisierung der Testauswertung für Steuergerätesoftware. In: [ElKfz03], S. 299-315

[BPS03] A. Baresel, H. Pohlheim, S. Sadeghipour: Structural and Functional Sequence Testing of Dynamic and State-Based Software with Evolutionary Algorithms. Proc. of Genetic and Evolutionary Computation Conference (GECCO 2003), Part 2, Chicago (US), LNCS Band 2724, S. 2428-2441, Springer, Juli 2003

[CWM98] M. Conrad, M. Weber, O. Müller: Towards a Methodology for the Design of Hybrid Systems in Automotive Electronics. Proc of 31st Int. Symposium on Automotive Technology and Automation (ISATA'98), Düsseldorf, 1998.

[DOORS04] DOORS, Telelogic AB at www.telelogic.com/products/doorsers.

[EAST04] U. Freund et.al. The EAST-ADL: A Joint Effort of the European Automotive Industry to Structure Distributed Automotive Embedded Control Software DOORS, 2nd European Congress ERTS, Embedded Real Time Software - January 2004, Toulouse.

[ElKfz01] 10. Internationaler Kongress "Elektronik im Kraftfahrzeug" (Tagungsband), VDI-Berichte, Band 1646, VDI Verlag, Düsseldorf 2001

[ElKfz03] 11. Internationaler Kongress "Elektronik im Kraftfahrzeug" (Tagungsband), VDI-Berichte, Band 1789, VDI Verlag, Düsseldorf 2003

[HWK+03] B. Hardung, M. Wernicke, A. Krüger, G. Wagner, F. Wohlgemuth: Entwurfsprozess für vernetzte Elektroniksysteme. In [Elkfz03]

[JHNTW99] G. John, M. Hoffmann, M. Nagel, C. Thomas, M. Weber: Using a Common Information Model as a Methodological Basis for a Tool-Supported Requirements Management Process. Proc. of 9th Int. Symposium of the Int. Council on Systems Engineering (INCOSE'99), Brighton, 1999.

[LBE+04] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, I. Fey: Model-based Testing of Embedded Automotive Software using MTest. SAE World Congress 2004, Detroit (US), März 2004 (in Vorbereitung)

[Leh00] E. Lehmann: Time Partition Testing: A Method for Testing Dynamic Functional Behaviour. Proc. of TEST2000, London (GB), 2000

[MDA01] Model Driven Architecture (MDA) - Whitepaper. Edited by Joaquin Miller and Jishnu Mukerji, OMG Architecture Board ORMSC1, July 9, 2001

[MLSLSF] The MathWorks, Simulink, Stateflow and Real-Time Workshop at http://www.mathworks.com/products

[OSEK] OSEK Specification at http://www.osek-vdx.org

[Ra02] A. Rau: Integrated Specification and Documentation of Simulink Models. Proc. of Int. Automotive Conference 2002 (IAC'02), Stuttgart, 2002.

[RCKFD00] A. Rau, M. Conrad, H. Keller, I. Fey, C. Dziobek: Integrated Model-based Software Development and Testing with CSD and MTest. Proc. of Int. Automotive Conference 2000 (IAC '00), Stuttgart, 2000.

[RMI04] Requirements Management Interface, The MathWorks Inc. at www.mathworks.com/products/rmi.

[SC03] I. Stürmer, M. Conrad: Test Suite Design for Code Generation Tools. 18. IEEE Int. Conf. Automated Software Engineering (ASE '03), Montreal (CDN), Okt. 2003

[SM01] Sax, E.; Müller-Glaser, K.-D.: A Seamless, Model-based Design Flow for Embedded Systems in Automotive Applications. Proc. of 1st Int. Symposium on Automotive Control (ISAC), Shanghai, China, 2001.

[SZ03] J. Schäuffele, T. Zurawka: Automotive Software Engineering. ATZ-MTZ-Fachbuch. Friedr. Vieweg & Sohn Verlag/GWV Fachverlage GmbH. Wiesbaden 2003.

[TargetLink] TargetLink by dSPACE at http://www.dspaceinc.com.

[UML20] UML 2.0 specifications of the OMG at http://www.omg.org/uml/.

[WW03] M. Weber, J. Weisbrod: Requirements Engineering in Automotive Development - Experiences and Challenges. In: IEEE Software, Jan/Feb. 2003, pp. 16-24.