Modellbasierte Steuergerätesoftwareentwicklung für den Automobilbereich

Holger Schlingloff[1,2], **Mirko Conrad**[3], **Heiko Dörr**[3], **Carsten Sühl**[4]

Abstract: Ein großer Teil der Neuerungen im Automobilbereich entsteht durch den Einsatz softwareintensiver Systeme im Fahrzeug. Die klassischen Methoden der Softwaretechnik zur Entwicklung großer Softwaresysteme werden dabei den spezifischen Herausforderungen der Entwicklung eingebetteter Steuergeräte im Automobil mittlerweile nicht mehr gerecht. Als Alternative wird daher seit einiger Zeit bei Herstellern und Zulieferern die modellbasierte Entwicklung verfolgt. Dabei wird auf Basis des ursprünglichen Lasten-/ Pflichtenheftes ein ausführbares Modell erstellt, welches als zentraler Ausgangspunkt für alle nachfolgenden Entwicklungsschritte dient. In verschiedenen Verfeinerungsschritten werden aus einem frühen Systemverhaltensmodell, ein physikalisches Modell, ein Implementierungsmodell und daraus der ausführbare Seriencode (teilweise vollautomatisch) erzeugt sowie entsprechende Testbeschreibungen abgeleitet. Wesentliche Fragen bei der modellbasierten Entwicklung betreffen die Definition und Verwaltung der Anforderungen in Relation zum Modell, die Absicherung der Codegenerierung bei hochoptimierenden Codegeneratoren, geeignete Strategien für den entwicklungsbegleitenden modellbasierten Test, sowie eine integrierte Methodik, welche auf domänenspezifischen Informationsmodellen basiert.

1. Einleitung

Im Bereich sicherheitsrelevanter eingebetteter Steuergeräte ist die Automobilbranche ein wesentlicher Innovationsmotor. Viele Verbesserungen in der Sicherheit des Fahrzeugs und im kundensichtbaren Mehrwert werden heutzutage durch neuartige Steuergeräte erreicht. Beispiele für solche Systeme sind elektronische Motor- und Getriebesteuerungen, Fahrdynamiksysteme wie ABS und ESP, Telematik- und Navigationssysteme, sowie Fahrerassistenz- und Insassenkomfortsysteme. Strategische Schwerpunkte der aktuellen Forschung und Entwicklung großer Hersteller und Zulieferer liegen dabei derzeit bei der Realisierung von Funktionen für sicheres und unfallfreies Fahren. Diese Funktionen können Unfälle verhüten, indem sie den Fahrer aktiv vor Gefahrensituationen warnen, oder können in bestimmten Notfällen sogar autonom die Fahrzeugführung übernehmen. Es ist dabei zu erwarten, dass die „Intelligenz“ der entsprechenden Steuergeräte und somit der Anteil der Software bei ihrer Entwicklung weiter drastisch zunimmt. Bereits heute werden in einem durchschnittlichen Oberklasse-Fahrzeug 50-80 Steuergeräte mit durchschnittlich jeweils über zehntausend Zeilen Programmcode eingesetzt. Diese kommunizieren über 3-5 verschiedene Bussysteme mit Hunderten von Nachrichten und Tausenden von Einzelsignalen miteinander. Auf Grund der hohen Komplexität der Interaktion können Probleme in einer einzelnen Komponente unübersehbare Auswirkungen auf das Gesamtsystem haben.

[1] Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik FIRST, Kekuléstr. 7, 12489 Berlin, ✉ Holger.Schlingloff@first.fhg.de
[2] Humboldt-Universität zu Berlin, Institut für Informatik, Rudower Chaussee 25, 12489 Berlin
[3] DaimlerChrysler AG, Research Information and Communication, Alt-Moabit 96A, 10559 Berlin, ✉ {Mirko.Conrad | Heiko.Doerr}@DaimlerChrysler.com
[4] S.E.S.A. Software und Systeme AG, Alt Moabit 91a, D-10559 Berlin, ✉ Carsten.Suehl@sesa.de

Diese Arbeit wurde unterstützt durch das vom BMBF im Rahmen der Forschungsinitiative „Software Engineering 2006“ geförderte Projekt „IMMOS - Integrierte Methodik für die modellbasierte Steuergeräteentwicklung“ [27], http://www.immos-projekt.de

Weitere spezifische Probleme der Softwareentwicklung im Automobilbereich sind die stetig verkürzten Entwicklungszeiten durch die immer schnellere Abfolge der Fahrzeuggenerationen. Betrug die durchschnittliche Generationsfolge in den 1980er Jahren noch ungefähr 50 Monate, so sind heute Zyklen von ca. 36 Monaten die Regel Darüber hinaus ist noch eine enorme Vielfalt von Varianten einer Fahrzeugbaureihe zu beherrschen. In Deutschland hat sich die Zahl der PKW-Modelle in den letzten 20 Jahren von 140 auf 260 fast verdoppelt; die Zahl der Varianten nahm dabei um 80% zu [34]. Jede Variante eines Fahrzeugs trägt zur exponentiellen Explosion der Anzahl von Kombinationen in der Fahrzeugelektronik und somit der möglichen Probleme bei. Konzeption und Durchführung der Integrationstests werden dadurch zu einem immer größeren Problem. Der Anteil der durch Softwarefehler bedingten Garantie- und Gewährleistungskosten ist ständig wachsend. Während bislang die Komplexität und Leistungsfähigkeit eingebetteter Systeme im Kraftfahrzeug hauptsächlich durch technische Randbedingungen (wie zum Beispiel die Leistungsfähigkeit der Mikrocontroller) eingeschränkt wurde, wird heute der Entwicklungs- und Qualitätssicherungsprozess der Systeme und der darin eingebetteten Software zum entscheidenden Faktor.

Ein neues Paradigma bei der Entwicklung automobiler Steuergeräte ist die *modellbasierte Entwicklung*. Dieses Paradigma soll dazu beitragen, den Größen- und Komplexitätszuwachs der Softwareanteile heutiger Kraftfahrzeuge beherrschbar zu machen und die Entwicklungszeiten durch eine höhere Effizienz zu reduzieren. Zudem soll die modellbasierte Entwicklung dabei helfen, die Sicherheit und Zuverlässigkeit der erstellten Software zu erhöhen und die Variantenvielfalt der Software-Produktfamilien in den Griff zu bekommen. Die Grundidee ist dabei, dass bereits frühzeitig im Entwicklungsprozess ein ausführbares Funktionsmodell des Steuerungs- und Regelungssystems erstellt wird, das zunächst zusammen mit Umgebungsmodellen simuliert und später direkt auf dem Steuergerät implementiert werden kann [18]. Das ausführbare Modell der zu realisierenden Software dient dann als Basis der weiteren konstruktiven und analytischen Entwicklungsschritte [25,3].

In diesem Beitrag geben wir einen Überblick über die modellbasierte Entwicklung, so wie sie momentan in zahlreichen Projekten der Automobilbranche erprobt wird und berichten über dort gewonnenen Erfahrungen. Wir beschreiben einige Probleme und Forschungsfragen, die in diesem Zusammenhang auftreten, und skizzieren die Lösungsansätze, die dabei zurzeit in der Automobilbranche und in Forschungsprojekten diskutiert werden.

2. Modellbasierte Entwicklung

Um den oben geschilderten stark gewachsenen Anforderungen an die Software eingebetteter Steuergeräte im Automobil gerecht zu werden, erfolgt derzeit ein Paradigmenwechsel in der Entwicklung: die klassischen softwaretechnischen Vorgehensmodelle werden immer mehr durch ein modellbasiertes Vorgehen ersetzt beziehungsweise ergänzt. Hauptcharakteristikum der modellbasierten Entwicklung ist der *durchgängige Einsatz ausführbarer Modelle* in allen Entwicklungsphasen bei Beibehaltung der verwendeten Modellierungsnotationen. Ein weiteres wesentliches Merkmal ist es, dass wesentliche der in herkömmlichen Softwareprozessen manuell entstehenden Artefakte automatisch generiert werden. Traditionell liegen die Ergebnisse der Analysephase (der Systemgrobentwurf) und der Designphase (der Systemfeinentwurf) lediglich als informelle Dokumente vor, die keine maschinelle Weiterverarbeitung ihrer Semantik erlauben. Der entsprechende Quellcode wird manuell aus den Modulbeschreibungen erstellt und für die Zielplattform übersetzt. Bei einer späteren Weiterentwicklung oder Portierung des Systems wird oftmals nur der Code angepasst, nicht aber die zugehörigen Entwurfsdokumente. Dadurch divergieren diese

nach einer gewissen Zeit stark von den tatsächlich implementierten Lösungen, was zu einem hohen Aufwand bei der Wartung und Weiterentwicklung führt.
In der modellbasierten Entwicklung werden diese Probleme dadurch vermieden, dass statt informeller Diagramme und natürlichsprachlicher Textdokumente als Antwort auf die Systemanalyse zunächst plattformunabhängige Modelle der späteren Systemfunktion (*Systemverhaltensmodell* und *physikalisches Modell*) erstellt werden. Das Systemmodell beschreibt dabei bestimmte Systemeigenschaften auf einer sehr hohen Abstraktionsstufe, z.B. als Betriebsmodi. Im physikalische Modell wird dann der Algorithmus auf der Ebene physikalischer Größen beschrieben. In weiteren Schritten wird das physikalische Modell so verfeinert, dass es auf die intendierte Zielplattform zugeschnitten ist. Aus dem resultierenden plattformspezifischen Modell (*Implementierungsmodell*) wird dann weitgehend automatisch der Code in einer höheren Programmiersprache erzeugt.

Diese zusätzlichen Verfeinerungen können separat verwaltet werden, so dass Verfeinerungsschritte in hohem Maße wieder verwendet werden können. Die eigentliche Weiterentwicklung der Funktionalität findet somit auf dem physikalischen Modell statt. In der traditionellen Entwicklung sind die Ergebnisse einer Entwicklungsphase in der darauffolgenden nicht unmittelbar verwendbar. Dadurch entstehen „Mauern“, die durch die Aufbereitung der Dokumente übersprungen werden müssen. In der modellbasierten Entwicklung bestehen die einzelnen Phasen in Wesentlichen aus einer inkrementellen Weiterentwicklung des selben Artefakts, so dass keine solchen Mauern entstehen. Der Qualitätssicherungs- und Testprozess begleitet die gesamte Entwicklung, so dass Fehler bereits frühzeitig erkannt und beseitigt werden können. Die Unterschiede zwischen den beiden Paradigmen sind in Abbildung 1 veranschaulicht.

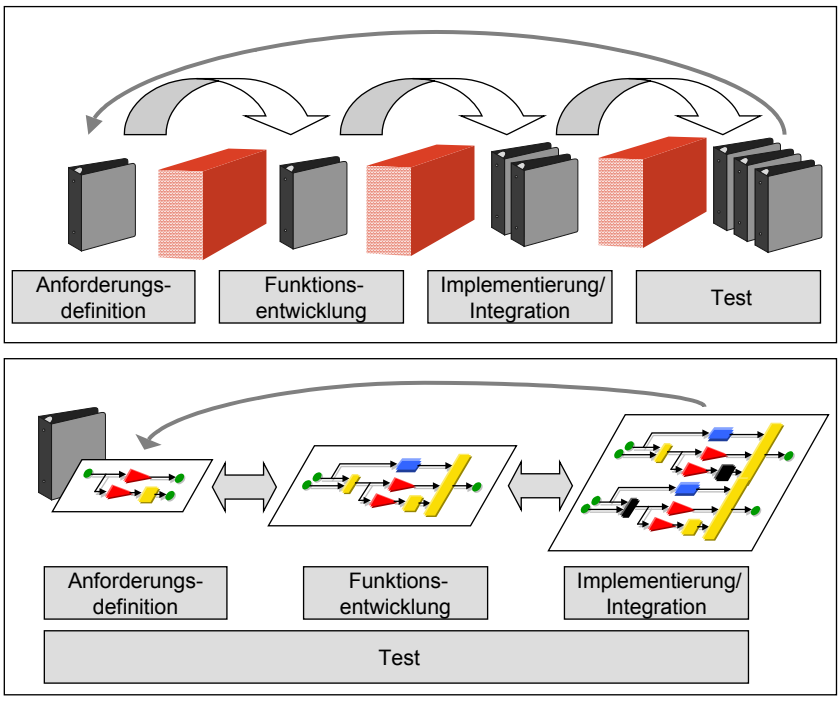

Abbildung 1: Traditioneller und inkrementeller Entwicklungsprozess (vgl. [33])

Bei der modellbasierten Entwicklung wird zunächst aus dem Lasten-/Pflichtenheft ein die textuellen Anforderungen ergänzendes Systemverhaltensmodell entwickelt. 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 Übergangsaktio-

nen, 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.Im Anschluss daran wird unter Anwendung methodischer Handlungsanleitungen das physikalisches Modell entwickelt. In diesem Modell werden die Algorithmen implementiert, die die gewünschte Steuerungs-, Regelungs- oder Überwachungsfunktionalität realisieren. Das physikalische Modell beschreibt die Ausgangsgrößen der Steuerung für die Aktuatoren in Abhängigkeit von den Sollwerten des Fahrers und Messgrößen der Sensoren. Zur Erprobung des physikalischen Modells wird zusätzlich ein Umgebungsmodell erstellt, das die (physikalische oder logische) Umgebung des zu implementierenden Systems beschreibt.

Ein Beispiel für eine Vorgehensweise zur Modellierung ist die im BMBF-Projekt Quasar für die Entwicklung automotiver Steuergeräte definierte Struktur von Anforderungsdokumenten [17], der das von Parnas und Madley vorgeschlagene Vier-Variablen-Modell [23] zugrunde liegt. Das physikalische Modell dient als zentraler Angelpunkt für alle nachfolgenden Entwicklungsaktivitäten, z.B.:

- die Erstellung von virtuellen Prototypen zu Demonstrationszwecken,
- Model-in-the-loop Simulationen,
- die automatisierte Erzeugung und Ausführung von Tests [3,4,5,14],
- Modellprüfung und Verifikation gegenüber formal spezifizierten Eigenschaften [8,24],
- die Durchführung simulationsbasierter Sicherheitsanalysen [13,22], sowie
- die Anreicherung um Implementierungsaspekte zur Generierung von compilierbarem (C-) Programmcode [16,19].

Die modellbasierte Vorgehensweise ermöglicht den Anwendern, verschiedene etablierte Verfahren in einen einheitlichen Rahmen zu integrieren. Das modellbasierte Entwicklungsparadigma schlägt dabei die Brücke zwischen der Erstellung von ausführbaren Modellen, der Generierung von übersetzbarem Programmtext aus diesen sowie der (teil-) automatischen Erzeugung von Tests. Wesentliche Vorteile sind dabei

- die Qualitätssteigerung durch Simulation- und Test auf Modellebene,
- Übertragung der Qualität auf den Code durch automatische Generierung, und
- kurze Rückkopplungsschleifen zwischen der Erprobung im Steuergerät und Funktionsanpassung im physikalischen Modell durch automatische Codegenerierung.
- der Effizienzgewinn durch den Einsatz modellbasierter Implementierungstechniken (Codegenerierung).

Durch Simulation und Test auf Modellebene kann eine erhebliche Steigerung der Qualität erzielt werden. Durch den Einsatz automatischer Codegenerierungsverfahren kann diese hohe Qualität bis zur Systemintegration bzw. Übertragung auf die Zielplattform erhalten werden. Darüber hinaus birgt dieses Paradigma ein erhebliches Potential zur Beschleunigung der Entwicklungsprozesse durch die verkürzten Rückkopplungszeiten zwischen den einzelnen Phasen. Ein weiterer Vorteil ist es, dass für die Modelle Notationen und Methoden verwendet werden können, die den in den Ingenieurdisziplinen üblicherweise verwendeten sehr ähnlich sind und daher schnell akzeptiert werden. Schließlich verbessert der

modellbasierte Ansatz auch die Effizienz der Zusammenarbeit zwischen Automobilherstellern und ihren Zulieferern, da die Verständigung über die Anforderungen nicht mehr auf der Basis informeller Pflichtenhefte oder fertiger Codesegmente, sondern auf der Ebene der Modelle geschieht.

In der Automobilbranche werden dabei häufig Modellierungstechniken verwendet, die auf Matlab/Simulink/Stateflow [32] aufbauen oder mit diesen verwandt sind. Diese verbinden dabei etliche Vorteile:

- Sie sind in Anwenderkreisen gut eingeführt und bekannt und können in allen konstruktiven Entwicklungsphasen verwendet werden.
- Die Ausdrucksmächtigkeit ist hinreichend groß, um alle relevanten Sachverhalte modellieren zu können, jedoch übersichtlich genug, um noch vollständig beherrschbar zu sein.
- Mittels Blockdiagrammen und erweiterten Zustandsautomaten lassen sich Daten- und Kontrollflüsse modellieren; darüber hinaus ist es möglich, hybride Modelle (mit diskreten und kontinuierlichen Anteilen) zu spezifizieren. Mathematische Modellbildungen (Differentialgleichungen) können direkt in das Modell integriert werden.
- Durch die unterstützten Strukturierungskonzepte ist eine hierarchische Dekomposition zur Beherrschung der Komplexität möglich. Mit spezialisierten Bibliotheken sind die Modellklassen an spezifische Bedürfnisse anpassbar.
- Mittels Verfeinerungen kann aus einem ersten Systemverhaltensmodell, welches die Anforderungen reflektiert, ein physikalisches Modell und daraus wiederum ein Implementierungsmodell erstellt werden. Das Implementierungsmodell dient direkt als Eingabe für den Codegenerator.

Als Beispiel soll nachfolgend ein Tempomat mit Abstandsregelung (TmA) dienen, der die Fahrzeuggeschwindigkeit abhängig von einer eingestellten Sollgeschwindigkeit und evtl. vorausfahrenden Fahrzeugen regelt.

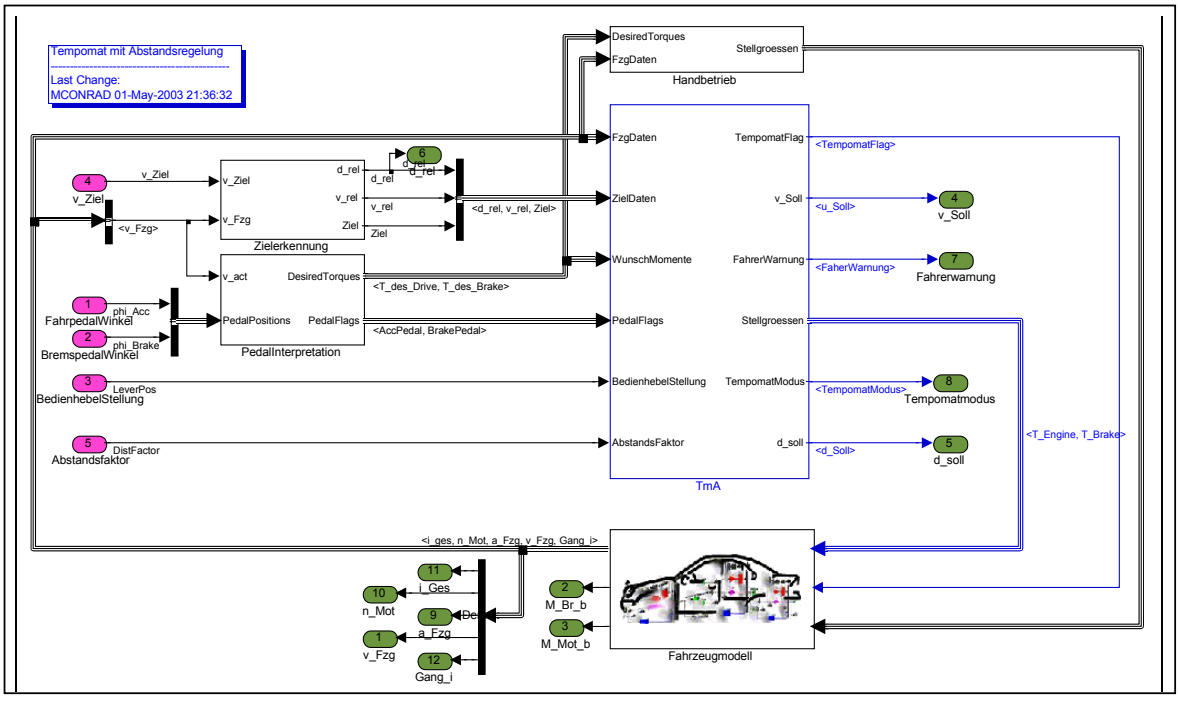

Abbildung 2: Tempomat mit Abstandsregelung

Wenn der Fahrer das System aktiviert, überwacht es den vor dem Fahrzeug liegenden Straßenabschnitt. Je nachdem, ob sich dort ein Fahrzeug befindet oder nicht, regelt das System die Geschwindigkeit so, dass entweder ein vorgegebener sicherer Abstand zum Vorderfahrzeug oder eine vorgegebene Geschwindigkeit eingehalten wird. In Abbildung 2 ist ein Simulink-Modell angegeben, welches die Struktur des Gesamtsystems darstellt [3]. Neben dem eigentlichen TmA System finden sich dort zwei vorverarbeitende Systeme für die Erkennung vorausfahrender Fahrzeuge (Zielerkennung) und für die Auswertung der aktuellen Pedalwerte (Pedalinterpretation) sowie Teile des Umgebungsmodells (Fahrzeugmodell, Handbetrieb).

Die Pfeile in dem Diagramm stellen Signalflüsse dar, wobei mehrere Signale zu einem Bus (Tupel) zusammengefasst werden können. Die Funktionsblöcke repräsentieren konzeptionelle Subsysteme. Das Modulkonzept von Simulink erlaubt eine beliebig tiefe Verschachtelung (schrittweise Verfeinerung) dieser Subsysteme, und stellt ein umfangreiches Repertoire an elementaren Blöcken (Funktionsbibliotheken) zur Verfügung.

Exemplarisch soll diese Verfeinerung der Funktionsblöcke anhand der Vorverarbeitungskomponente Pedalinterpretation illustriert werden.

ID	Beschreibung
SR-PI-01	**Erkennung der Pedalbetätigung** Werden Fahr- oder Bremspedal über einen bestimmten Schwellwert hinaus betätigt, ist dies durch ein pedalspezifisches Binärsignal anzuzeigen.
SR-PI-01.1	**Erkennung der Bremspedalbetätigung** Wird das Bremspedal über einen Schwellwert ped_min hinaus betätigt, ist das Binärsignal BrakePedal auf den Wert 1 zu setzen, andernfalls auf 0.
SR-PI-01.2	**Hysterese Bremspedalbetätigung** Bei der Erkennung der Bremspedalbetätigung ist keine Hysterese vorzusehen.
SR-PI-01.3	**Erkennung der Fahrpedalbetätigung** Wird das Fahrpedal über einen Schwellwert ped_min hinaus betätigt, ist das Binärsignal AccPedal auf den Wert 1 zu setzen, andernfalls auf 0.
SR-PI-01.4	**Hysterese Fahrpedalbetätigung** Bei der Erkennung der Fahrpedalbetätigung ist keine Hysterese vorzusehen.
SR-PI-02	**Interpretation der Pedalwege** Die normierten Pedalwege von Fahr- und Bremspedal sind als Wunschmomente zu interpretieren. Dabei sind sowohl Komfort- als auch Verbrauchsaspekte zu berücksichtigen.
SR-PI-02.1	**Interpretation des Bremspedalweges** Der normierte Bremspedalweg ist als Wunschbremsmoment T_des_Brake [Nm] zu interpretieren. Das Wunschbremsmoment ergibt sich, indem der aktuelle Pedalweg zum maximalen Bremsmoment T_max_Brake ins Verhältnis gesetzt wird.
SR-PI-02.2	**Interpretation des Fahrpedalweges** Der normierte Fahrpedalweg ist als Wunschabtriebsmoment T_des_Drive [Nm] zu interpretieren. Das Wunschabtriebsmoment ist dabei im nichtnegativen Bereich geschwindigkeitsabhängig so zu skalieren, dass bei höheren Geschwindigkeiten der Abtriebsmomentenwunsch kleiner wird.

Abbildung 3: Anforderungen an die Pedalinterpretation

Bei rein mechanischen Systemen entsprechen die vom Fahrer eingestellten Winkel des Fahr- und Bremspedals direkt den Kraftmomenten am Gas- bzw. Bremszug. Mit einer elektronischen Steuerung ist es möglich, aus den aktuellen, normierten Pedalwegen von Fahr- und Bremspedal unter Verwendung von Kennlinien und der aktuellen Geschwindigkeit Wunschmomente für die Beschleunigung oder Abbremsung zu berechnen, die sowohl Komfort- als auch Verbrauchsaspekte berücksichtigen. Diese Interpretation soll von der genannten Komponente vorgenommen werden. Darüber hinaus soll für jedes Pedal ein Flag generiert werden, das anzeigt, ob es vom Fahrer gedrückt wird oder nicht.

In Abbildung 3 sind einige der Anforderungen für die Pedalinterpretation aufgelistet.Die letzte dieser Anforderungen (SR-PI-02.2) wird dabei über eine tabellengestützte mathematische Berechnungsvorschrift detailliert. Ein Modell, welches diese und weitere Anforderungen umsetzt, ist in der nachfolgenden Abbildung 4 angegeben. Im oberen Teil dieses Diagramms sieht man die Berechnung der Pedalerkennung aus den gegebenen Pedalwinkeln. In der Mitte ist die Normierung des Bremspedalwinkels mit dem maximalen Bremsmoment dargestellt. Im unteren Teil ist die Interpretation des Fahrpedalwinkels abhängig von der aktuellen Fahrzeuggeschwindigkeit zu sehen.

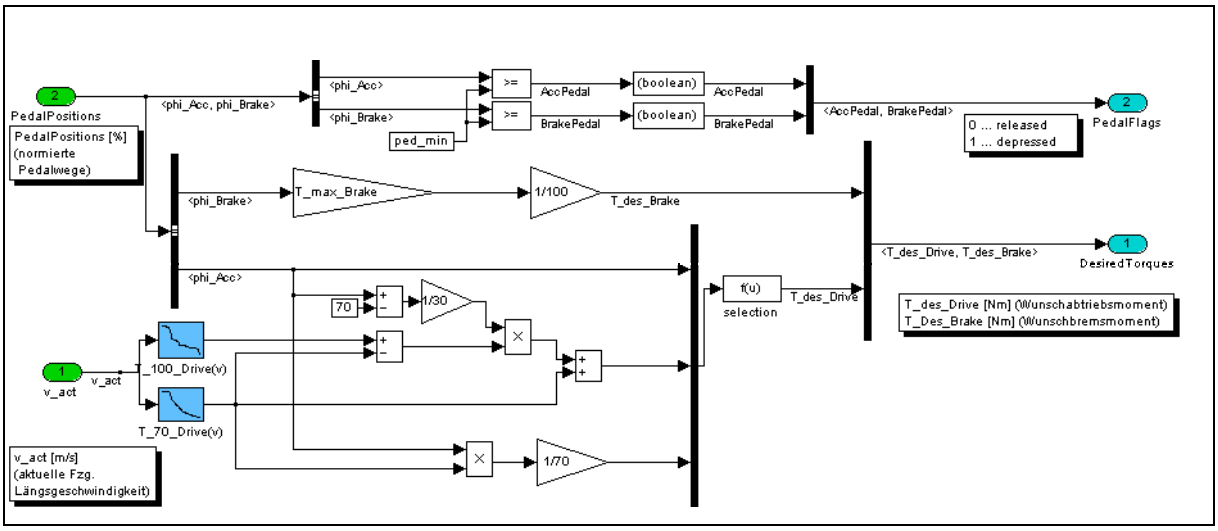

Abbildung 4: Physikalisches Modell der Pedalinterpretation

Reichert man ein solches Modell um die notwendigen Implementierungsinformationen an, kann daraus mit einem Codegenerator wie TargetLink [9] automatisch Floating- (Abbildung 5) oder Fix-Point Code erzeugt werden. Der generierte Code ist dabei ANSI-C konform und kann daher für jeden Microcontroller verwendet werden, für den ein entsprechender Compiler vorliegt. TargetLink unterstützt auch OSEK/VDX konforme Betriebssysteme und kann somit auf Multitasking- und Realzeitunterstützung zurückgreifen[19].

In diesem Codefragment entspricht Sb1_InPort1 und Sb1_InPort2 den Pedalpositionen von Fahr- und Bremspedal, Sb1_InPort der aktuellen Geschwindigkeit des Fahrzeugs, Sb1_OutPort1 bzw. Sb1_OutPort3 den Pedal-Flags (Fahr- bzw. Bremspedal betätigt) sowie Sb1_OutPort und Sb1_OutPort2 dem Wunschabtriebs- bzw. Wunschbremsmoment. Die Berechnung der Ausgabewerte ergibt sich in diesem Fall als einfache Zuweisung durch arithmetische Ausdrücke über den Eingangsgrößen und den im Modell verwendeten Konstanten. Im Allgemeinen kann die Auflösung des Modellgraphen jedoch zu beliebig komplizierten Programmen mit Bedingungen und Schleifen führen. Der Codegenerator geht dabei hochoptimierend vor und berücksichtigt auch Optimierungskriterien des Zielcompilers und der Zielplattform.

```
/*****************************************************************\
   Model step function of subsystem(s): PedalInterp
\*****************************************************************/
Void PedalInterp(
   Float32 Sb1_InPort              /* 32 bit floating point variable */,
   Float32 Sb1_InPort1             /* 32 bit floating point variable */,
   Float32 Sb1_InPort2             /* 32 bit floating point variable */,
   Float32 * Sb1_OutPort  /* pointer to 32 bit floating point variable */,
   Bool * Sb1_OutPort1             /* pointer to boolean type */,
   Float32 * Sb1_OutPort2 /* pointer to 32 bit floating point variable */,
   Bool * Sb1_OutPort3             /* pointer to boolean type */)
{
   /* 1D-TableLookup: PedalInterp/T_70_Drive(v) */
   Sb1_T_70_Drive_v_ = (Int16)(Tab1DS16I80T30237_b(
       (const MAP_Tab1DS16I80T30237_b *)(&(Sb1_T_70_Drive_v__map)),
       (Int32)Sb1_InPort));

  /* TargetLink outport: PedalInterp/OutPort2
   *Sb1_OutPort2 = Sb1_InPort2 * 4000.F * 0.01F;

   /* TargetLink outport: PedalInterp/OutPort
    *Sb1_OutPort = ((Sb1_InPort1 - 70.F) * 0.03333333333F * (((Float32)(Tab1DS16I80T30237_b(
       (const MAP_Tab1DS16I80T30237_b *)(&(Sb1_T_100_Drive_v__map)),
       (Int32)Sb1_InPort)))-((Float32)Sb1_T_70_Drive_v_)))+((Float32)Sb1_T_70_Drive_v_)+
    * Sb1_InPort1 * 0.01428571429F * ((Float32)(Sb1_InPort1 <= 70.F)));     (((Float32)Sb1_T_70_Drive_v_)

   /* TargetLink outport: PedalInterp/OutPort1
   *Sb1_OutPort1 = Sb1_InPort1 >= 5.F;

   /* TargetLink outport: PedalInterp/OutPort3
   *Sb1_OutPort3 = Sb1_InPort2 >= 5.F;
}
```

Abbildung 5: Automatisch erzeugter C Code

Erste modellbasierte Entwicklungsprojekte zeigen massive Verbesserungen gegenüber der klassischen Softwareentwicklung für Steuergeräte [18]. Da es sich bei der modellbasierten Softwareentwicklung jedoch um eine relativ junge Technologie handelt, ist diese bisher in wichtigen Bereichen nur unzureichend unterstützt. Für den zukünftigen flächendeckenden Einsatz sind die systematische Verfeinerung der vorhandenen Methoden und Werkzeuge und deren Integration notwendig. Der Einsatz in sicherheitsrelevanten Innovationen erfordert insbesondere die Verbindung von konstruktiven und analytischen Techniken zur Gewährleistung der Qualität (das heißt in erster Linie der Zuverlässigkeit) der Steuergeräte. Im folgenden Abschnitt beschreiben wir daher einige Fragen und Lösungsansätze, die dazu beitragen können, die genannten Probleme in den Griff zu bekommen.

3. Aktuelle Problemstellungen und Forschungsfragen

Im vorigen Abschnitt wurde der Stand der Technik für die modellbasierte Entwicklung von Software in sicherheitsrelevanten automobilen Steuergeräten beschrieben, so wie sie derzeit in Pilotprojekten großer Hersteller und Zulieferer eingesetzt wird. Die dabei gemachten Erfahrungen sprechen für die Praktikabilität des Ansatzes, führen jedoch auch zu einer Reihe weiterer Forschungsfragen und Problemstellungen. Wesentliche Fragen betreffen die Definition und Verwaltung der Anforderungen in Relation zum Modell, die Absicherung der Codegenerierung bei hochoptimierenden Codegeneratoren, geeignete Strategien für den entwicklungsbegleitenden modellbasierten Test, sowie eine integrierte Methodik, welche auf domänenspezifischen Informationsmodellen basiert. Nachfolgend werden diese Punkte genauer erörtert sowie einige aktuelle Lösungsansätze vorgestellt.

2.1. Integration der Verwaltung von Anforderungen in den modellbasierten Entwicklungsprozess

Anforderungen an sicherheitsrelevante Steuergeräte und ihre Software werden üblicherweise textuell in Requirements-Management-Tools wie beispielsweise DOORS [31] verwaltet. Diese erlauben es, Anforderungen miteinander und mit anderen Artefakten (z.B.

Testbeschreibungen) zu vernetzen und ihre Umsetzung im Modell und im Code zu verfolgen. Die herkömmlichen Vorgehensmodelle gehen dabei davon aus, dass die Anforderungsdefinitionen während der Anforderungsanalyse erstellt und dann kontinuierlich verfeinert werden. In der Praxis werden Anforderungsbeschreibungen an eine Steuergerätefunktionalität jedoch oftmals nicht nur einmalig erstellt, sondern wachsen kontinuierlich während des Entwicklungsprozesses. In jeder Entwicklungsphase kommen neue Anforderungen hinzu, die sich mit den bislang existierenden überschneiden oder ihnen sogar widersprechen [15]. Daraus ergeben sich eine Vielzahl von Problemen hinsichtlich des Aufwandes zur Verwaltung von Anforderungen, zur Sicherung der Konsistenz etc. Hinzu kommt, dass die bestehenden Lösungen zur Verwaltung von Anforderungen domänenunabhängig ausgelegt sind und kaum Möglichkeiten für eine semantische Integration in die verschiedenen Entwicklungsphasen mitbringen.

Eine Lösungsmöglichkeit für die genannten Probleme besteht darin, einen semantischen Kern zu definieren, der einerseits textuell formulierte Anforderungen zulässt, der aber zugleich eine inhaltsbezogene Verfeinerung von sowohl funktionalen wie nichtfunktionalen Anforderungen in die modellbasierte Entwicklung unterstützt. Als geeignete formale Grundlage sind hier verschiedene formale Spezifikationssprachen, etwa OCL [21], CSP-Casl [11] oder logische Ansätze [2] denkbar. Mit einer derartigen Fundierung kann die Verwaltung und Überprüfung von Anforderungen den gewünschten zentralen Stellenwert erhalten und direkt zur Qualitätssicherung beitragen. Zugleich kann ein derartiger Ansatz das traditionell aufgabenteilige Zusammenspiel von Herstellern und Zulieferern verbessern.

Um textuelle Anforderungsbeschreibungen in den modellbasierten Entwicklungsprozess einbeziehen zu können, müssen sie in modellartige Beschreibungen transformiert werden. Dazu ist es notwendig, eine Strukturierung vorzugeben, die dies ermöglicht. Anforderungen, die sich auf diese zu definierende Strukturierung beschränken, können dann auf kanonische Art in modellbasierte Beschreibungen umgesetzt werden. Eine interessante Frage dabei ist, ob die Anforderung in einer bestimmten Struktur angeordnet werden muss, oder die Formulierung einer Anforderung eine bestimmte Struktur aufweisen muss. Eine Möglichkeit ist es, dass die Dokumentgliederung der textuellen Anforderung analog zu einem Modell aufgebaut wird. Die Ableitung von Modellen aus textuellen Anforderungen setzt voraus, dass Modellkonzepte bereits in den textuellen Anforderungen vorweggenommen werden. In diesem Fall scheint es jedoch effizienter, gleich mit der Erstellung des Modells zu beginnen. Zur Formalisierung sowohl der funktionalen wie auch der nichtfunktionalen Anteile von Anforderungsbeschreibungen gibt es Industriestandards bzw. Standardisierungsbemühungen, zum Beispiel die oben erwähnte OCL. Das allgemeine Problem der Abbildung beliebiger sprachlicher Ausdrücke in formale Modelle ist aus theoretischen Gründen unlösbar. Für geeignete Teilsprachen lassen sich jedoch gute Erfolge erzielen [10]. Die Abbildung von Anforderungsbeschreibungen auf ein Ausführungsmodell ist nur möglich, wenn eine entsprechende formale Semantik zu Grunde gelegt wird; nur dann kann man von einer „korrekten“ Umsetzung der Anforderungsbeschreibungen sprechen. Eine solche formale Semantik für Modelle und Anforderungen ermöglicht dann auch weitergehende Qualitätssicherungsmaßnahmen wie Korrektheitsbeweise, Simulationen und Modellprüfungen [2,12].

2.2. Absicherung der Codegenerierung bei hochoptimierenden Codegeneratoren

Wie bereits oben dargelegt wurde, besteht einer der wesentlichen Vorteile der modellbasierten Entwicklung darin, dass durch die automatische Codegenerierung nicht nur eine Effizienz-, sondern auch eine Qualitätssteigerung erreicht wird, da der generierte Code

semantisch genau dem Modell entspricht. Dies setzt jedoch eine absolute Zuverlässigkeit der entsprechenden Codegeneratoren voraus. Im Gegensatz zu klassischen Compilern für Programmiersprachen gibt es für Codegeneratoren noch keine etablierten Prozesse, die deren Korrektheit sicherstellen können. Da die Betriebsbewährtheit der Codegeneratoren durch den schnellen Technologiewandel in diesem Bereich nicht vorausgesetzt werden kann, muss die jeweils verwendete Codegeneratorkonfiguration einem wohldefinierten Absicherungsprozess unterzogen werden [29]. Eine Möglichkeit besteht darin, den erzeugten Code zusätzlich abzusichern: Beispielsweise kann ein manuelles Review des generierten Codes (vgl. [17]) durch spezielle Richtlinien unterstützt werden. Eine andere Möglichkeit ist es, die im Codegenerator verwendeten Optimierungsverfahren formal zu verifizieren, wie dies für sicherheitsrelevante Objektsprachencompiler getan wird. Im Codegenerator werden die Modelle als Graph repräsentiert, deren Semantik durch den generierten Code gegeben ist. Eine Codeoptimierung entspricht dann einer Graphtransformation [6,28,29]. Wenn es gelingt, zu zeigen, dass diese Transformation semantikerhaltend ist, so ist die Korrektheit des generierten Codes gewährleistet.

Im Allgemeinen ist es nicht möglich, die Korrektheit jeder Optimierung formal zu beweisen. Darüber hinaus gibt es Probleme, die nicht auf Optimierungen in der Codegenerierung, sondern auf das nachgeschaltete Zusammenspiel von Zielcompiler und Zielplattform zurückzuführen sind. Ein weiterer zentraler Bestandteil einer Strategie zur Absicherung der Codegenerierung ist daher eine *generische Testsuite für Codegeneratoren*, die die Absicherung unter Berücksichtigung der verschiedenen Entwicklungsstadien der Steuergeräte und der Host-Target-Problematik unterstützt [29,30]. Diese Testsuite besteht aus einer Anzahl verschiedener Modelle, aus Eingabedaten für diese Modelle, sowie aus den zugehörigen erwarteten Reaktionen (Ausgaben). Das obige Beispiel der Pedalinterpretation mitsamt mehrerer Ein- und Ausgabesequenzen verschiedener Pedalwinkel und Wunschmomente entspricht *einem* Testfall einer solchen generischen Suite. Zur Durchführung der Codegenerator-Tests werden die Testfälle (Modelle) übersetzt, auf der Zielplattform zur Ausführung gebracht, und die erhaltenen Ergebnisse mit den Erwartungsergebnissen verglichen. Die Testüberdeckung der generischen Testsuite ergibt sich aus der Anzahl der Modellkonstrukte, den jeweiligen Eingabedatenräumen der Modelle sowie den verschiedenen Codegenerator-Optimierungen [29,7].

2.3. Strategien für den Modellbasierter Test

Im modellbasierten Entwicklungsparadigma wird der gesamte Entwicklungsprozess parallel von Qualitätssicherungsmaßnahmen und Tests begleitet. Im Gegensatz zur klassischen Softwareentwicklung, bei der für eine Funktionalität i.d.R. nur ein ausführbares Ergebnis, der Programmcode, erstellt wird, ergeben sich in der modellbasierten Entwicklung zu einer Funktionalität routinemäßig mehrere ausführbare Artefakte (z.B. Funktionsmodell, Implementierungsmodell, Autocode). Kombiniert man diese mit den unterschiedlichen Ausführungsplattformen (z.B. Host-PC, Evaluierungsboard, Steuergerät) und Umgebungen (z.B. simuliertes Umgebungsmodell, Prüfstand, Fahrzeug) ergeben sich eine Vielzahl sogenannter *Testmöglichkeiten* und damit die Möglichkeit neuer Herangehensweisen an den Softwaretest. Dabei müssen für einen gründlichen Test der Funktionalität nicht zwangsläufig alle Testmöglichkeiten ausgeschöpft werden. Eine Teststrategie für die modellbasierte Entwicklung sollte daher möglichst komplementäre Testverfahren kombinieren und die Auswahl einer geeigneten Teilmenge von Testmöglichkeiten unterstützen, so dass eine hohe Fehleraufdeckungswahrscheinlichkeit erreicht wird [4,5].

Die Basis einer solchen *Teststrategie* bildet die systematische Ableitung von funktionalen Tests und deren Anwendung auf das physikalische Modell [3,4]. Falls notwendig, sind so-

lange zusätzlich Strukturtests zu ermitteln, bis ein geeignetes Strukturtestkriterium auf Modellebene erfüllt ist [5,7]. 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.

Bei der Umsetzung einer solchen Strategie ergibt sich das Problem, eine kohärente Testumgebung bereit zu stellen, die es erlaubt, das System auf den verschiedenen Entwicklungsstufen durchgängig mit den selben Testfällen zu testen. Beim Model-in-the-loop-Test (MIL) wird das System- oder Implementierungsmodell mit dem Umgebungsmodell zusammengeschaltet und zur Ausführung gebracht. Auf der nächsten Stufe (Software-/Processor-in-the-loop-Test, SIL/PIL) wird der generierte Code auf einer Host- bzw. targetnahen Plattform getestet. Die Umgebung wird dabei üblicherweise durch spezielle Testtreiber und -module realisiert. Beim Hardware-in-the-loop-Test (HIL) wird die in das Steuergerät integrierte Software im Zusammenspiel mit simulierten (z.B. Fahrer, Straße usw.) und realen (z.B: Drosselklappe) Umgebungsbestandteilen getestet. Für die Abbildung abstrakter Ereignisse der Testbeschreibung auf die verschiedenen Realisierungen der Testumgebung wird eine einheitliche Vorgehensweise benötigt [20].

2.4. Ein kohärentes Informationsmodell für die modellbasierte Systementwicklung

Ein wesentlicher Aspekt bei der Gestaltung des modellbasierten Entwicklungsprozesses ist die Integration der Testaktivitäten mit den Entwurfs- und Implementierungsaktivitäten, die zeitlich miteinander verzahnt auszuführen sind: Beispielsweise muss die Vorbereitung der Testdurchführung im Projekt so früh wie möglich einsetzen. Die oben genannten Fragestellungen konzentrieren sich auf die Konzeption von spezialisierten Entwicklungsmethodiken für einzelne Entwicklungsdisziplinen, also z.B. für Anforderungsbeschreibung, Modellierung / Implementierung und Qualitätssicherung in der modellbasierten Entwicklung. Das Zusammenwirken dieser Einzelmethoden muss im Rahmen einer *integrierten Methodik für die Entwicklung von Steuergerätesoftware* definiert werden. Eine solche integrierte Methodik sollte das Vorgehen bei der Entwicklung eingebetteter Software im Automobilbereich beschreiben und dem Systementwickler Hilfsmittel zur Anwendung der Methodik zur Verfügung stellen. Ein solches Hilfsmittel ist z.B. das in [12] beschriebene Konzept der Agenden.

Hierzu müssen die Einzelmethoden in geeigneter, einheitlicher Weise verfügbar gemacht werden und ihr Zusammenwirken definiert werden. Dazu werden die jeweiligen Informationen, die im Rahmen der Anforderungsermittlung, der Modellierung und im Test erhoben werden, beschrieben und zueinander in Bezug gesetzt. Durch diese Bezugnahme kann die integrierte Methodik für die modellbasierte Entwicklung verteilter Steuerungs- und Regelungssysteme im Kraftfahrzeug erarbeitet werden. Grundlage hierfür ist die Erstellung eines *Informationsmodells für die modellbasierte Entwicklung*, welches alle wesentlichen in der modellbasierten Entwicklung auftretenden Informationseinheiten und deren Zusammenhänge in abstrakter Form beschreibt. Auf Grundlage solch eines kohärenten Informationsmodells wird beispielsweise der modellbasierte Test methodisch fundiert.

Typische Fragestellungen, die hierbei auftreten, betreffen den Zusammenhang von Anforderungen und Modellen, die Abdeckung von Anforderungen durch Testfälle, die Auswir-

kung von Änderungen einzelner Subsysteme, und die Übertragbarkeit und Wartbarkeit von Systemteilen im Rahmen des Variantenmanagements.

Ein möglicher Lösungsansatz ist auf Basis des generischen Metamodellierungsstandards MOF (Meta Object Facility) denkbar [21]. Mit MOF lassen sich in einer allgemeingültigen Darstellung verteilte Objekte (so genannte Meta-Objekte) verwalten. Zum Austausch der generischen MOF-Daten dient das XMI (XML Metadata Interchange Format), welches durch seine MOF-Integration alle im vorstehend beschriebenen Entwicklungsprozess entstehenden textuellen und binären Daten verarbeiten kann. Auf Basis von XMI bzw. MOF können Datenflussmodelle, Zustandsmodelle aber auch Daten aus dem objektorientierten Softwareentwurf verwaltet werden. Auch eine Erweiterung für andere Beschreibungsnotationen ist ohne große Schwierigkeiten möglich.

Um den Anforderungen unterschiedlicher Entwurfsdaten aus verschiedenen Phasen des Entwurfsprozesses gerecht zu werden, ist es notwendig, in der Architektur ein zentrales Informationsmodells vorzusehen. Für die Aufstellung eines solchen kohärenten Informationsmodells wird ein Verfahren zur domänenspezifische Beschreibung der eingesetzten Methoden und deren Informationsgehalte verwendet. Die Datenhaltung für die entsprechenden Artefakte erfolgt – konsistent zum zentralen Metamodell – weiterhin dezentral in den jeweiligen Werkzeugen. Eine für alle Werkzeuge verfügbare Kommunikationsschicht vermittelt die Zugriffe von Entwurfswerkzeugen, und ein zentrales integriertes Konfigurationsmanagement versioniert die zu verwaltenden Daten [1]. Durch Bildung und Speicherung von Konfigurationen muss dabei eine Durchgängigkeit der Entwurfsdaten im Laufe der verschiedenen Entwurfsphasen sichergestellt werden. Auf diese Weise kann eine strukturierte Identifikation der Zusammenhänge zwischen Requirements, Modellen und Tests zum Aufbau einer geschlossenen Vorgehensweise ermöglicht werden, welche die bereits existierenden modellbasierten Einzeltechniken - vor allem die modellbasierte Anforderungsanalyse, die modellbasierte Codegenerierung und den modellbasierten Test - zu einer einheitlichen Entwicklungsmethodik integriert.

4. Ausblick

In diesem Artikel haben wir einen Überblick über den modellbasierten Entwicklungsprozess gegeben, so wie er derzeit in der Automobilbranche vielfach erprobt wird. Wir haben Probleme mit diesem Prozess aufgezeigt und Lösungsansätze für diese Probleme diskutiert. Einige der Probleme werden derzeit in laufenden und beantragten Forschungsprojekten bearbeitet. Im Rahmen der Forschungsoffensive „Software Engineering 2006“ fördert das BMBF die Projekte IMMOS (Integrierte Methodik zur modellbasierten Steuergeräteentwicklung, [27]), AutoMoDe (Modellbasierte Entwicklung verteilter Steuerungssoftware im Automobilsektor) und SiLEST (Entwicklung und Erprobung von Methoden und Werkzeugen für den Test und die Sicherheitsanalyse eingebetteter Echtzeitsoftware). Diese Projekte beschäftigen sich direkt mit den angegebenen Forschungs- und Entwicklungsaufgaben. Sie sind im Frühjahr 2004 gestartet und laufen bis Sommer 2006. Auf Grund der zugehörigen umfangreichen Verwertungsplanung ist noch während der Laufzeit dieser Projekte mit einem weiteren Aufschwung der modellbasierten Entwicklung zu rechnen.

Literatur

[1] Altheide, F.; Dörr, H. and Schürr, A.: Requirements to a Framework for sustainable Integration of System Development Tools. In H. Stoewer, L. Garnier (eds.), Proc. of 3rd Europ. Systems Engineering Conference (EuSEC'02), pp. 53-57, Toulouse, France, May 2002

[2] Clarke, E.M. and Schlingloff, H.: Model Checking. Chapter 21 in Alan Robinson and Andrei Voronkov (eds.), Handbook of Automated Reasoning, Elsevier Science Publishers B.V., pp. 1367-1522, 2000.

[3] Conrad, M.: Auswahl und Beschreibung von Testszenarien für den Modell-basierten Test eingebetteter Software im Automobil. Dissertation, Technische Universität Berlin, 2004 (in Vorbereitung).

[4] Conrad, M.; Dörr, H.; Fey, I. and Yap, A.: Model Based Generation and Structured Representation of Test Scenarios, Proc. of Workshop on Software-Embedded Systems Testing (WSEST'99), 1999.

[5] Conrad, M.; Fey, I., and Sadeghipour, S.: Systematic Model-Based Testing of Embedded Control Software: The MB^{3}T Approach. ICSE 2004 Workshop on Software Engineering for Automotive Systems (SEAS'04), Edinburgh, May 25th, 2004.

[6] Conrad, M.; Dörr, H.; Schürr, A. and Stürmer, I.: Graph-Transformations for Model-based Testing. Modellierung 2002, GI-Lecture Notes in Informatics, P-12, pp. 39-50, Tutzing, 2002.

[7] Conrad, M. and Sadeghipour, S.: Einsatz von Überdeckungskriterien auf Modellebene – Erfahrungsbericht und experimentelle Ergebnisse. Software-Technik Trends, Softwaretechnik-Trends 22:2, pp. 1-6, 2002.

[8] Damm, W.; Schulte, C.; Wittke, H.; Segelken, M.; Higgen, U., and Eckrich, M.: Formale Verifikation von ASCET Modellen im Rahmen der Entwicklung der Aktivlenkung. Proc. Workshop Automotive SW Engineering & Concepts, Frankfurt/M., 2003.

[9] dSPACE GmbH: TargetLink – Production Code Generator, www.dspace.de, 2004.

[10] Flake, S; Müller, W. and Ruf, J.: Structured English for Model Checking Specification. In K. Waldschmidt, C. Grimm (Hrsg.), Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Frankfurt/M., pp. 99-108, VDE-Verlag, 2000.

[11] Gimblett, A.; Roggenbach, M.; and Schlingloff, H.: Towards a formal specification of an electronic payment systems in CSP-CASL; Proc. Workshop on abstract data types (WADT) at ETAPS, 2004.

[12] Grieskamp, W.; Heisel, M. and Dörr, H.: Specifying Safety-Critical Embedded Systems with Statecharts and Z: An Agenda for Cyclic Software Components. In E. Astesiano (ed.), Proc. ETAPS-FASE'98', LNCS 1382, pp. 88-106, Springer, 1998.

[13] Grönberg, R., Conrad. M.: Werkzeugunterstützung für Sicherheitsanalysen von Hardware- und Software-Systemen. Technical Report FT3/A-2000-007, DaimlerChrysler AG, Forschung und Technologie 3, Berlin, 2000.

[14] Hahn, G.; Philipps, J.; Pretschner, A. and Stauner, T.: Prototype-Based Tests for Hybrid Reactive Systems. 14. Proc. IEEE Int. Workshop on Rapid System Prototyping, pp. 78-85, San Diego, US, June 2003.

[15] Hausmann, J.H.; Heckel, R. and Taentzer, G.: Detection of Conflicting Functional Requirements in a Use Case driven approach. Proc. of 24th Int. Conference on Software Engineering (ICSE) 2002, pp. 105-115, ACM, 2002.

[16] Jungmann, M. and Beine, B.: Automatische Code-Generierung für sicherheitskritische Systeme, Automotive
Electronics, Heft II/2003.

[17] Kamsties, E.; von Knethen, A. and Paech, B.: Structure of QUASAR Requirements Documents. Technischer Bericht Nr. 073.01/E, Fraunhofer IESE, 2001.

[18] Klein, T.; Conrad, M.; Fey, I. and Grochtmann, M.: Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler. In B. Rumpe, W. Hesse (Hrsg.), Modellierung 2004, Lecture Notes in Informatics (LNI), Band P-45, S. 31-41, Köllen Verlag, 2004

[19] Köster, L.; Thomsen, Th. and Stracke, R.: Von Simulink nach OSEK: Automatische Codegenerierung für Echtzeitbetriebssysteme mit TargetLink. Embedded Intelligence, 2001.

[20] Lamberg, K.; Beine, M.; Eschmann, M.; Otterbach, R.; Conrad, M. and Fey, I.: Model-based testing of embedded automotive software using MTest. Proc. of SAE World Congress 2004, Detroit, US, 2004.

[21] OMG (Object Management Group): http://www.omg.org/
UML 2.0 Object Constraint Language (OCL) Specification: http://www.omg.org/docs/ptc/03-10-14.pdf; Meta-Object Facility (MOF™), version 1.4: http://www.omg.org/technology/documents/formal/mof.htm

[22] Papadopoulos, Y., McDermid J., Mavrides, A., Scheidler S., Maruhn, M.: Model-Based Semiautomatic Safety Analysis Of Programmable Systems In Automotive Applications. Proc. of Int. Conference on Advanced Driver Assistance Systems (ADAS'01), Birmingham, UK, pp.53-57, September 2001.

[23] Parnas, D.L. and Madey, J. Functional documents for computer systems. Science of Computer Programming, 25:41–61, May 1995.

[24] Rau, A.: Verwendung von Zusicherungen in einem modellbasierten Entwicklungsprozess. Informationstechnik und Technische Informatik it + ti, 44 (2002) 3, pp. 137-144.

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

[26] Schlich, M. and Denger, Ch.: Optimierung von Inspektionen durch die Nutzung von Agendas für die perspektivenbasierte Lesetechnik. Visek-Forum, Berlin, 17. Oktober 2003, http://www.visek.de

[27] Schlingloff, H.; Sühl, C.; Dörr, H.; Conrad, M.; Stroop, J.; Sadeghipour, S.; Kühl, M.; Rammig, F. and Engels, G.: IMMOS - Eine integrierte Methodik zur modellbasierten Steuergeräteentwicklung. Proceedings, BMBF-Statusseminar „Software Engineering 2006“, Berlin, 1.-3. Juli 2004.

[28] Stürmer, I.: A Contribution of Graph-Grammar Techniques for the Specification, Verification and Certification of Code Generation Tools, Proc. of Int. Workshop on Graph-based Tools (GraBaTs'02), 2002. published in Electronic Notes in Theoretical Computer Science, Volume 72, No.2.

[29] Stürmer, I. and Conrad, M.: Test Suite Design for Code Generation Tools. 18th IEEE Int. Conf. on Automated Software Engineering (ASE '03), Montreal, Canada, October 6-10, 2003.

[30] Stürmer, I. and Conrad, M.: Code Generator Certification: A Testsuite-oriented Approach. Proc. of Automotive - Safety & Security, Stuttgart, 6.-7. Oktober 2004

[31] Telelogic AB: DOORS. www.telelogic.com/products/doorsers, 2003.

[32] The MathWorks: Simulink, Stateflow and Real-Time Workshop. www.mathworks.com/products, 2004.

[33] Tung, J.: From Specification and Design to Implementation and Test: A Platform for Embedded System Development. Proc. of Int. Automotive Conference 2004 (IAC'04), Stuttgart, 15.-16. Juni 2004.

[34] Womack, J.P., Jones, D. and Roos, D.: Die zweite Revolution in der Autoindustrie – Konsequenzen aus der weltweiten Studie des MIT. Heyne Campus (Frankfurt, New York) 1997.