

Der Softwareentwurf im Entwicklungsprozess mechatronischer Systeme

**Prof. Dr. Wilhelm Schäfer¹, Tobias Eckardt¹, Christian Henke⁴, Lydia Kaiser³,
Timo Kerstan², Jan Rieke¹, Dr. Matthias Tichy¹**

¹Fachgebiet Softwaretechnik, ²Fachgebiet Entwurf paralleler Systeme,
³Fachgebiet Produktentstehung, ⁴Fachgebiet Regelungstechnik und Mechatronik
Heinz Nixdorf Institut, Universität Paderborn
Warburger Straße 100, 33098 Paderborn
Tel. 05251/60-3312, Fax. 05251/60-3530
E-Mail: wilhelm@upb.de

Zusammenfassung

Die Innovation im modernen Maschinenbau ist vor allem getrieben durch die eingebettete Software. Insbesondere die auf Software basierende Kommunikation und Koordination einzelner Systeme, durch die sich spontan vernetzende dynamische Gesamtsysteme ergeben, ermöglicht es, neuartige Produkte zu entwickeln, die sich flexibel an geänderte Situationen anpassen können. Dies resultiert allerdings in einer deutlich komplexeren Struktur der Software. Es müssen daher geeignete Methoden und Techniken genutzt werden, die Software zu entwickeln, um die Qualität der Produkte sicherzustellen. Ein wichtiger Fokus liegt hierbei auf der Sicherheit der Systeme. Im Rahmen dieses Beitrags erläutern wir die an der Universität Paderborn entstandene Methode für den Softwareentwurf vernetzter, mechatronischer Systeme insbesondere unter dem Aspekt der Sicherheit der entwickelten mechatronischen Systeme.

Schlüsselwörter

Softwareentwurf, Entwicklungsprozess, MechatronicUML, Prinziplösung, RTOS

1 Einleitung

Mit eingebetteter Software bezeichnet man typischerweise die Software eines mechatronischen Systems. Mit Hilfe der Software wird das Bewegungsverhalten bestimmt, d.h. es werden Reaktionen auf die Eingaben von Sensoren berechnet und die notwendigen Aktorbefehle generiert, um das Bewegungsverhalten zu erzeugen bzw. zu verbessern. Dies erlaubt eine wesentlich höhere Flexibilität und Anpassbarkeit des Systems. Die Entwicklung mechatronischer Systeme ist daher zunehmend auch durch die Entwicklung der Softwarekomponenten geprägt.

Dieser prägende Einfluss wird noch dominanter, wenn mechatronische Systeme miteinander vernetzt sind und Software wesentlich die Kommunikation und Koordination bestimmt. Hierbei ist die Regelung der Aktorik massiv von der Koordination mit anderen Systemen abhängig, die typischerweise auf dem Austausch von Nachrichten basiert. Dies führt zu äußerst komplexer hybrider (kontinuierlicher / diskreter) Software [SW07]. Des Weiteren werden mechatronische Systeme häufig für zeitkritische Aufgaben eingesetzt, in denen die Sicherheit des Gesamtsystems von der Einhaltung harter Echtzeitkriterien abhängt.

Die dadurch weiter steigende Komplexität von Software in (vernetzten) mechatronischen Systemen kombiniert mit ihrem Einsatz in sicherheitskritischen Bereichen erfordert eine hohe Präzision in der Entwicklung, um ein qualitativ hochwertiges Produkt zu erhalten. Es müssen daher geeignete Techniken zur Analyse der Software genutzt werden. Verstärkt werden neben testbasierten Techniken auch formale Verifikationstechniken eingesetzt, die die Einhaltung von Anforderungen mathematisch beweisen. Durch den oben angesprochenen prägenden Einfluss der Software stellt der Softwareentwurf einen äußerst wichtigen Bestandteil des Entwicklungsprozesses mechatronischer Systeme dar. In diesem Beitrag wird die in den letzten Jahren im Rahmen des Sonderforschungsbereichs 614 „Selbstoptimierende Systeme des Maschinenbaus“ entwickelte MechatronicUML [BGT05, GHH+08] zum modellgetriebenen Entwurf der Software mechatronischer Systeme sowie deren Einbettung in den Entwicklungsprozess mechatronischer Systeme vorgestellt. Für eine detaillierte Darstellung zu verwandten Arbeiten wird auf [GH06, SW07, GHH+08] verwiesen.

Ein Beispiel für ein mechatronisches System sind die RailCabs der Neuen Bahntechnik Paderborn¹. Die Forschung im Rahmen der Neuen Bahntechnik Paderborn wurde 1998 mit dem Ziel begonnen, die Vorteile von Individualverkehr und öffentlichem Verkehr zu verbinden. Grundlegende Idee ist, den herkömmlichen Schienenverkehr durch autonome selbstfahrende Fahrzeuge – die so genannten RailCabs – zu ersetzen. RailCabs können einzeln fahren oder sich auf längeren Strecken mit einer höheren Verkehrsdichte zu autono-

¹ www.railcab.de

men Konvois berührungslos zusammenschließen. Die im Vergleich zu herkömmlichen Zügen kleineren autonom fahrenden Einheiten und die ohne mechanische Kupplungsvorgänge auskommende Bildung von Konvois ermöglicht eine flexiblere Nutzung der RailCabs, da auf Anforderung des Nutzers gefahren werden kann und keine starren Fahrpläne existieren.

Software ist ein integraler Bestandteil dieses Systems, da viele der innovativen Funktionen wie das autonome Fahren und die Bildung von Konvois nur mit Hilfe von Software realisiert werden können. Im Rahmen dieses Beitrags erläutern wir daher die entwickelte Methode an Hand ausgewählter Teile der Software des RailCabs.

Der folgende Abschnitt beinhaltet eine Darstellung des Entwicklungsprozess mechatronischer Systeme auf Basis der VDI-Richtlinie 2206 [VDI2006] mit dem Fokus auf der Prinziplösung als Ergebnis der ersten, disziplinübergreifenden Entwurfsaktivität. Der Softwareentwurf auf Grundlage der Prinziplösung wird in Abschnitt 3 behandelt. Nach einer kurzen Darstellung der Ausführung der modellierten Software auf einem Echtzeitbetriebssystem schließen wir in Abschnitt 5 mit einem Resümee und einem Ausblick auf zukünftige Arbeiten.

2 Prinziplösung

Die Entwicklung mechatronischer Systeme erfolgt in drei Phasen: Systementwurf, disziplinspezifischer Entwurf und Systemintegration (Abbildung 1). Das V-Modell in der VDI Richtlinie 2206 beschreibt den disziplinübergreifenden Entwicklungsprozess mechatronischer Produkte [VDI2206].

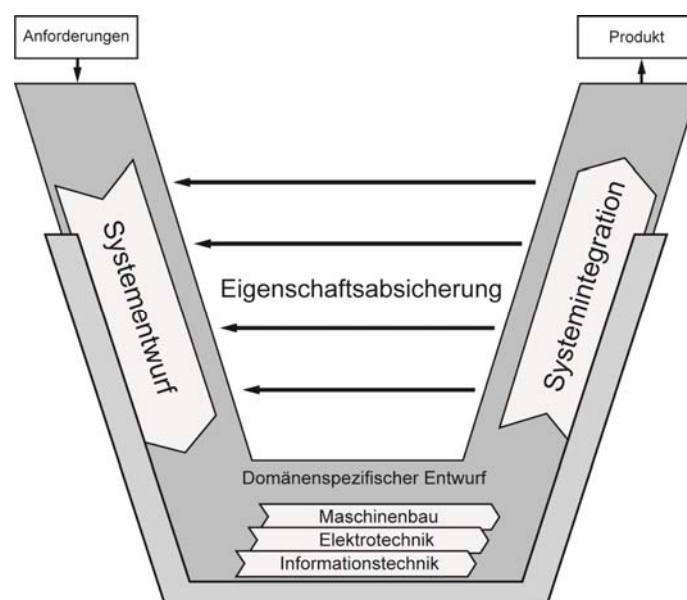


Abbildung 1: V-Modell der VDI Richtlinie 2206 [VDI2206]

Im Systementwurf wird ausgehend von den Anforderungen das Produktkonzept erstellt und in Form der so genannten Prinziplösung beschrieben. Da hier erste Festlegungen getroffen werden, ist es notwendig, dass alle Disziplinen gleichermaßen beteiligt sind. Die Prinziplösung beschreibt das Produkt ganzheitlich und muss für alle Beteiligten verständlich abgebildet werden. Als Basis für diese Zusammenarbeit wurde im Sonderforschungsbereich 614 die Spezifikationstechnik zur disziplinübergreifenden Beschreibung der Prinziplösung entwickelt [GFD+08].

Bei der Spezifikation des Systems steht neben der Beschreibung des prinzipiellen Aufbaus und der Wirkungsweise die Beschreibung dessen Verhaltens im Vordergrund. Um die umfassende Beschreibung der Prinziplösung handhabbar zu machen, ist diese in acht Aspekte unterteilt: Anforderungen, Funktionen, Anwendungsszenarien, Wirkstruktur, Umfeld, Gestalt, Verhalten und Zielsystem. Rechnerintern werden die Aspekte durch Partialmodelle repräsentiert. Da die Partialmodelle der Prinziplösung verschiedene Sichten auf das gleiche Produkt darstellen, wird sowohl die Konsistenz innerhalb der Partialmodelle als auch die partialmodellübergreifende Konsistenz sichergestellt (s. [Geh05, GKP+10]).

In der Disziplin Softwaretechnik bilden die Aspekte Anforderungen, Wirkstruktur und Verhalten die Grundlage für die Konkretisierung. Eine ausführliche Beschreibung aller weiteren Aspekte ist in [GFD+08] zu finden.

Die Anforderungen (z.B. Baugröße, Leistungsdaten) an das zu entwickelnde Produkt richten sich an alle Disziplinen gleichermaßen. Diese werden in dem Partialmodell Anforderungen strukturiert in einer Liste gesammelt. In den Entwicklungsphasen Konzipierung und Konkretisierung sind diese zu berücksichtigen und umzusetzen. Die Erfüllung der Anforderungen wird in der Phase Systemintegration überprüft.

Die Wirkstruktur beschreibt die prinzipielle Wirkungsweise des Systems mittels Systemelementen und ihren Wechselwirkungen. In Abbildung 2 ist ein Ausschnitt aus der Wirkstruktur des RailCabs zu sehen. Die Systemelemente repräsentieren das System (*RailCab*), Subsysteme (*Regelung der Längsdynamik*) sowie Hardware oder Software (*Abstandsregler*). Die Wechselwirkung zwischen den Elementen wird durch Flüsse dargestellt. Die Soll-Vorgabe der RailCab-Geschwindigkeit wird zum Beispiel von dem *Fahrprofilgenerator* dem *Geschwindigkeitsregler* übermittelt. Dieses ist durch die gestrichelte Linie (Informationsfluss) abgebildet. Die durchgezogene Linie spezifiziert einen Energiefluss (z.B. auf die Schienen wirkende Kraft). Durch die disziplinübergreifende Beschreibung der prinzipiellen Wirkungsweise des Systems in der Prinziplösung wird die Umgebung der Software definiert. Dieses Wissen benötigt der Softwareentwickler für den disziplinspezifischen Entwurf der Software.

Abbildung 3: Zusammenwirken der Partialmodelle Wirkstruktur und Verhalten

3 Softwarespezifikation

Die Prinziplösung stellt den Ausgangspunkt für die Verfeinerung im Rahmen des disziplinspezifischen Entwurfs dar. Im Folgenden wird der Entwurf der Software näher betrachtet. Hierbei verfolgen wir einen integrierten Ansatz, der sowohl die Software im Regler als auch die Software zur nachrichtenbasierten Kommunikation und Koordination betrachtet.

Dieser integrierte Ansatz unterstützt nicht nur die Modellierung der Software, sondern auch die Analyse, ob die Software die Anforderungen, und im Besonderen die Sicherheitsanforderungen erfüllt. Wichtig hierbei ist, dass die Analyse auch für komplexe vernetzte Systeme eingesetzt werden kann. Dies bedeutet, dass kompositionale Analyseverfahren eingesetzt werden, die nur einzelne Aspekte bzw. Teile des Systems analysieren, aber deren Ergebnisse für das ganze System gelten.

In unserem Ansatz nutzen wir das vor allem aus, um die nachrichtenbasierte Koordination zwischen den einzelnen Teilen des Systems auf die Einhaltung von Sicherheitsanforderungen zu überprüfen. Im Folgenden beschreiben wir als erstes den Übergang von der Prinziplösung in den Softwareentwurf. Dann erläutern wir in Abschnitt 3.2 die Spezifikation der Softwarearchitektur. Die Verhaltensmodelle für die Komponenten der Softwarearchitektur werden in Abschnitt 3.3 erläutert.

3.1 Übergang von der Prinziplösung in den Softwareentwurf

Die Prinziplösung des Systems wird durch die Entwickler der verschiedenen Disziplinen mit ihren eigenen Tools und Modellen konkretisiert. Für den modellbasierten Softwareentwurf für verteilte, sicherheitskritische Echtzeitsysteme wird die MechatronicUML [BGT05] eingesetzt, um die Softwarearchitektur zu modellieren und das Verhalten des Systems und seiner Komponenten zu spezifizieren.

Diese disziplinspezifischen Modelle sollten dabei zu Beginn der Konkretisierung sinnvollerweise automatisiert aus der Prinziplösung abgeleitet werden. So wird sichergestellt, dass in der Konkretisierung in den verschiedenen Disziplinen tatsächlich auch das System entwickelt wird, das in der Prinziplösung spezifiziert wurde.

Für Softwaretechnikmodelle müssen dabei unter anderem Systemelemente aus der Wirkstruktur der Prinziplösung in MechatronicUML-Komponenten übersetzt werden. Informationsflüsse zwischen den Systemelementen werden in Ports und entsprechende Verbindungen und Interfaces übersetzt. Zustandsmodelle mit den zugehörigen Regelkonfigurationen werden in hybride Rekonfigurationscharts überführt (siehe Abschnitt 3.3.3). Dabei sind jedoch nicht alle Teile der Prinziplösung relevant für die Softwaretechnik: Reine Mechanik-Elemente (wie beispielsweise das Fahrwerk) oder elektronische Bauteile (wie das Kommunikationsmodul) sollten sinnvollerweise nicht in die Softwaretechnikmodelle über-

nommen werden, um diese nicht unnötig zu verkomplizieren. Dazu werden die Elemente der Prinziplösung mit Relevanzmarkierungen annotiert (siehe Abbildung 4). Übersetzt werden dann nur die Elemente, die für die entsprechenden Disziplinen relevant sind. Dies ist jedoch nicht immer trivial: Während das Kommunikationsmodul nicht relevant für die Softwaretechnik ist, sind es die Informationsflüsse, die durch das Modul laufen, aber sehr wohl. Sie müssen also in Verbindungen im UML-Modell übersetzt werden, die die kommunizierenden Komponenten direkt verbinden und die (nicht vorhandene) Kommunikationskomponente überspringen.

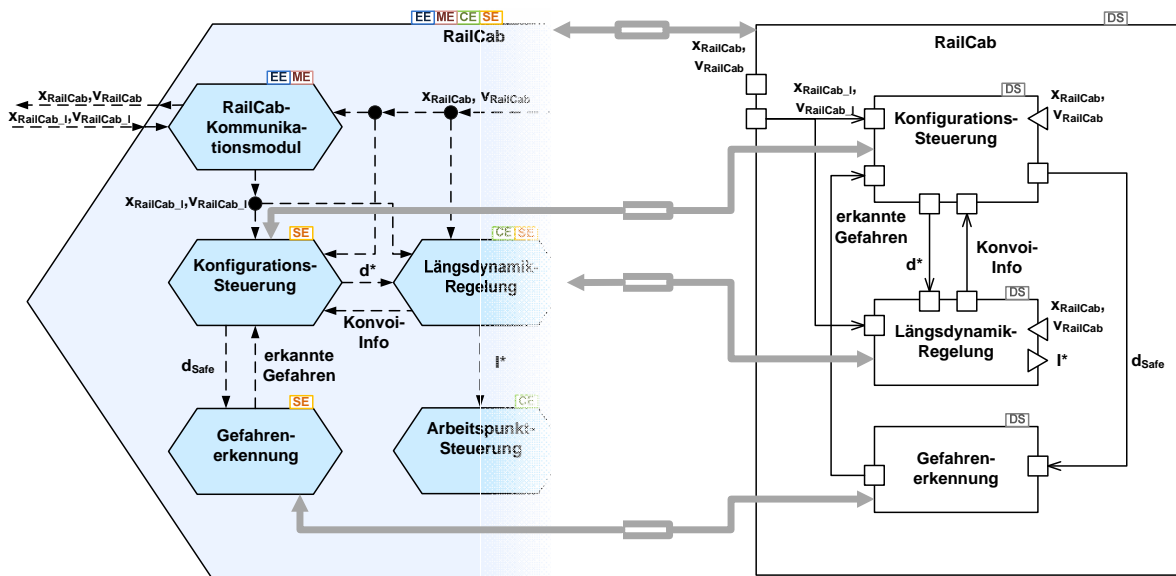


Abbildung 4: Übergang von Wirkstruktur zu UML-Komponentendiagramm

Um solche disziplinspezifischen Modelle zu generieren, werden Tripel-Graph-Grammatiken (TGG) zur automatischen Modelltransformation verwendet. TGG [Sch94] sind ein regelbasierter Formalismus, mit dem man genau solche Relationen definieren kann, wie sie oben beschrieben sind. Zusätzlich erzeugen TGG sogenannte Korrespondenzknoten zwischen Quell- und Zielmodell, so dass nachvollziehbar ist, welche Teile des erzeugten UML-Modells zu welchen Teilen der Systemspezifikation gehören. Außerdem können so nachträgliche Änderungen der beteiligten Modelle synchronisiert werden, so dass auch Änderungen der Prinziplösung während der Konkretisierung möglich bleiben. Details zur Transformation finden sich in [GSS+09].

3.2 Spezifikation der Softwarearchitektur

Grundlage der Softwarearchitektur und damit der Struktur der MechatronicUML-Komponenten ist das Operator-Controller-Modul (OCM) [OHG04]. Das OCM besteht aus mehreren Ebenen, die unterschiedliche Aufgaben erfüllen und unterschiedlichen Anforderungen hinsichtlich der Sicherheit unterliegen.

Der Controller ist die unterste Ebene bezogen auf den Durchgriff auf das technische System (Regelstrecke). Dieser Regelkreis verarbeitet in direkter Wirkkette die Messsignale, ermittelt Stellsignale und gibt diese aus. Dabei kann sich der Controller aus mehreren Reglern zusammensetzen, zwischen denen umgeschaltet werden kann.

Der reflektorische Operator überwacht und steuert den Controller. Er greift dabei nicht direkt auf die Aktorik des Systems zu, sondern modifiziert den Controller, indem er Parameter- oder Strukturänderungen initiiert. Bei Strukturänderungen – wie beispielsweise Rekonfigurationen – werden nicht nur die Regler ausgetauscht, sondern es werden auch entsprechende Kontroll- bzw. Signalflüsse im Controller umgeschaltet. Der reflektorische Operator ist weiterhin für die Echtzeitkommunikation zwischen mehreren OCMs verantwortlich.

Der kognitive Operator stellt die oberste Ebene des OCMs dar. Auf dieser Ebene kann das System durch Anwendung vielfältiger Methoden (etwa Lernverfahren, modellbasierte Optimierungsverfahren oder den Einsatz wissensbasierter Systeme) Wissen über sich und die Umgebung zur Verbesserung des eigenen Verhaltens nutzen. Er wirkt dabei durch Ereignisse auf den reflektorischen Operator ein. Im Folgenden konzentrieren wir uns auf den Entwurf des Controllers und des reflektorischen Operators, da diese im Gegensatz zum kognitiven Operator für die Sicherheit des Systems relevant sind.

Die OCMs des zu entwickelnden Systems werden mittels MechatronicUML-Komponentendiagrammen spezifiziert (siehe Abbildung 5), welche eine Erweiterung von UML-Komponentendiagrammen sind. Die mittels dieser Diagramme spezifizierte Architektur beschreibt die hierarchische Einbettung von Komponenten, wie sie aus der Prinziplösung abgeleitet wird.

An die spezifizierten Komponenten werden Ports angefügt, welche den in der Prinziplösung beschriebenen Informationsfluss umsetzen. An dieser Stelle wird zwischen diskreten und kontinuierlichen Ports unterschieden. Während ein diskreter Port die Kommunikation und Koordination mit anderen Komponenten durch Austausch von Nachrichten umsetzt, liegt an einem kontinuierlichen Port durchgängig der Wert einer bestimmten Variable an. So können sowohl diskret agierende Softwarekomponenten als auch kontinuierlich agierende Regler modelliert werden.

Im Beispiel der RailCab-Komponente (Abbildung 5) wird die RailCab-zu-RailCab-Kommunikation anhand der diskreten Ports hinten und vorne realisiert. Die eingebetteten Regler Fahrprofilgenerator, Abstandsregler und Geschwindigkeitsregler sind hingegen mittels kontinuierlicher Ports verbunden, da hier kontinuierlich Werte zwischen den Reglern ausgetauscht werden. Das dargestellte Beispiel abstrahiert von einer umfassenden Darstellung aller eingebetteten Komponenten und zeigt nur solche Komponenten, die im weiteren Verlauf Relevanz haben.

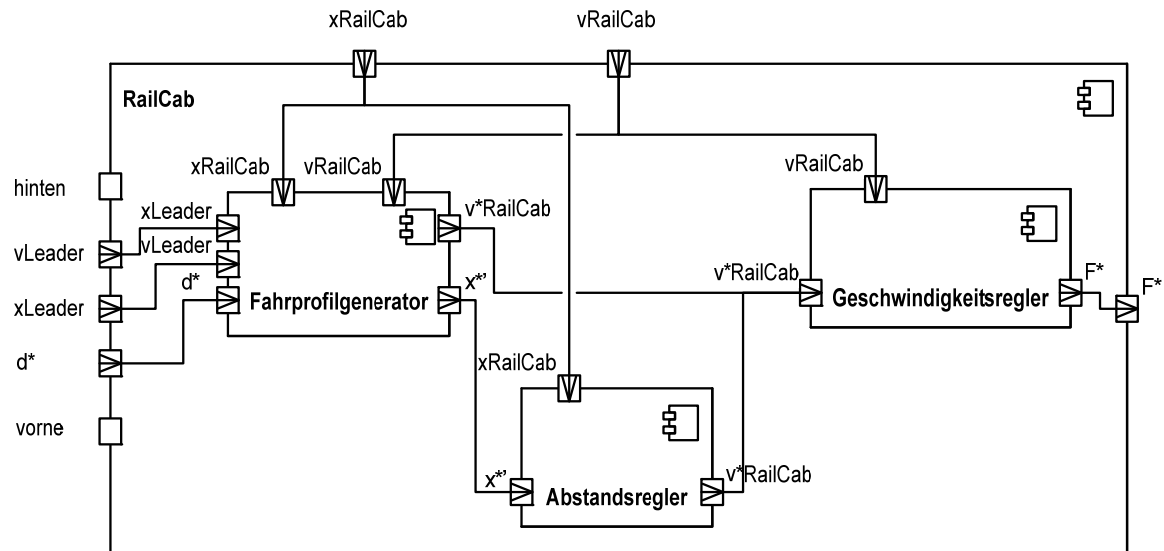


Abbildung 5: Hybride RailCab-Komponente mit eingebetten Komponenten für die Geschwindigkeits- und Abstandsregelung

Die Verbindungen zwischen Ports in einem MechatronicUML-Komponentendiagramm beschreiben alle möglichen Konfigurationen die zur Laufzeit existieren können. Der Geschwindigkeitsregler kann beispielsweise die Soll-Geschwindigkeit des RailCabs $v^*_{RailCab}$ entweder vom *Fahrprofilgenerator* oder vom *Abstandsregler* als Eingabe bekommen. Welche Konfiguration tatsächlich aktiv ist, hängt von dem diskreten Zustand des Systems ab und wird somit im diskreten Verhaltensmodell der Komponente spezifiziert (siehe Abschnitt 3.3.3). Dies ermöglicht die Anpassung des Systems an die Umwelt zur Laufzeit. Das Verhalten von rein kontinuierlichen Komponenten hingegen wird mittels Blockdiagrammen durch die Regelungstechnik festgelegt (siehe Abschnitt 3.3.2).

3.3 Spezifikation des Verhaltens

Die mittels Komponentendiagrammen spezifizierte Architektur beschreibt ein gemischt kontinuierlich-diskretes System. Das Kommunikationsverhalten zur Koordination verschiedener MechatronicUML-Komponenten basiert auf einem diskreten Zustandsmodell, welches in der MechatronicUML mittels Realtime Statecharts modelliert wird. Das Verhalten von kontinuierlich agierenden eingebetteten Reglern hingegen wird mittels Blockdiagrammen spezifiziert. In die diskreten Zustände des Koordinationsverhaltens werden danach die unterschiedlichen Strukturkonfigurationen der kontinuierlichen Regelung eingebettet. Die dabei entstehenden Modelle werden hybride Rekonfigurationscharts genannt.

Nach der folgenden Erläuterung der Modelle für das diskrete Komponentenverhalten werden nachfolgend die Regelungsstruktur sowie die Einbettung der Regler in das diskrete Komponentenverhalten vorgestellt.

3.3.1 Diskretes Komponentenverhalten

Das diskrete Verhalten von Softwarekomponenten wird mittels Realtime Statecharts spezifiziert. Grundsätzlich muss das gesamte Verhalten des Systems im Abschluss verifiziert werden, um die Sicherheit des Systems zu gewährleisten. Da eine Verifikation des gesamten Systems jedoch meist aufgrund der Komplexität zu einer Zustandsraumexplosion führt, wird das Gesamtverhalten des Systems kompositional spezifiziert, um es anschließend auch kompositional verifizieren zu können. Für die Verhaltensspezifikation bedeutet dies, dass unterschieden wird zwischen komponenteninternem und komponentenexternem Verhalten und dass diese Verhalten zunächst unabhängig voneinander spezifiziert werden.

Komponentenexternes Verhalten wird mittels *Echtzeitkoordinationsmustern* [GTB+03] spezifiziert. So wird die Kommunikation zur Koordination eines Konvois zwischen zwei RailCabs beispielsweise mit dem Muster *KonvoiKoordination* spezifiziert (siehe Abbildung 6). Dieses besteht strukturell aus den beteiligten Rollen *vorne* und *hinten*, dem Konnektor sowie Sicherheitseigenschaften. Das Verhalten der Rollen sowie des Konnektor werden mit Realtime Statecharts spezifiziert. Die Sicherheitseigenschaften mit einer Temporallogik.

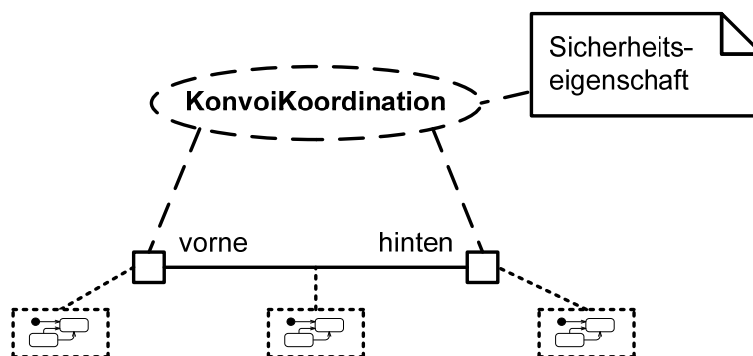


Abbildung 6: Echtzeitkoordinationsmuster KonvoiKoordination

Realtime Statecharts [GB03] sind eine Erweiterung von UML-Zustandsautomaten um Echtzeitannotationen, welche die Modellierung von periodischen Ausführungen, echtzeitabhängigem Verhalten und Worst-Case Execution Times (WCETs) erlauben. Sie werden eingesetzt, um das Echtzeitkoordinationsverhalten zwischen zwei Rollen eines Echtzeitkoordinationsmusters zu beschreiben (siehe Abbildung 7). Neben einer nachrichtenbasierten Kommunikation sind die wesentlichen Erweiterungen Uhren und Deadline-Intervalle an Transitionen.

Uhren sind vergleichbar mit Stoppuhren; sie können zurückgesetzt und in Transitionsbedingungen und Zustandsinvarianten genutzt werden. So wird die Ausführung eines Realtime Statecharts zeitabhängig gemacht. In dem Beispiel zur Konvoikoordination (Abbildung 7) schickt die Rolle *hinten* eine Nachricht *konvoiAnfrage* an die Rolle *vorne*. Diese setzt bei Empfang dieser Nachricht die Uhr t_0 zurück und wechselt in den Zustand *warten*. Nach frühestens 1, aber spätestens 1000 Zeiteinheiten wechselt sie in den Zustand *antworten*,

worauf sie der Rolle *hinten* entweder mit einer *konvoiAnfrageAbgelehnt*- oder einer *starteKonvoi*-Nachricht antwortet.

In den Realtime Statecharts zur Konvoikoordination werden weiterhin *Deadline-Intervalle* d_c für die Transitionen eines Realtime Statecharts spezifiziert. Diese definieren die minimale und die maximale Ausführungszeit einer Transition. In Zusammenhang mit Worst-Case Execution Times für die Seiteneffekte, die bei der Ausführung von Transitionen oder in Zuständen ausgeführt werden, sind somit alle relevanten Zeitinformatoren im Modell enthalten, um es korrekt auf einer Hardwareplattform auszuführen.

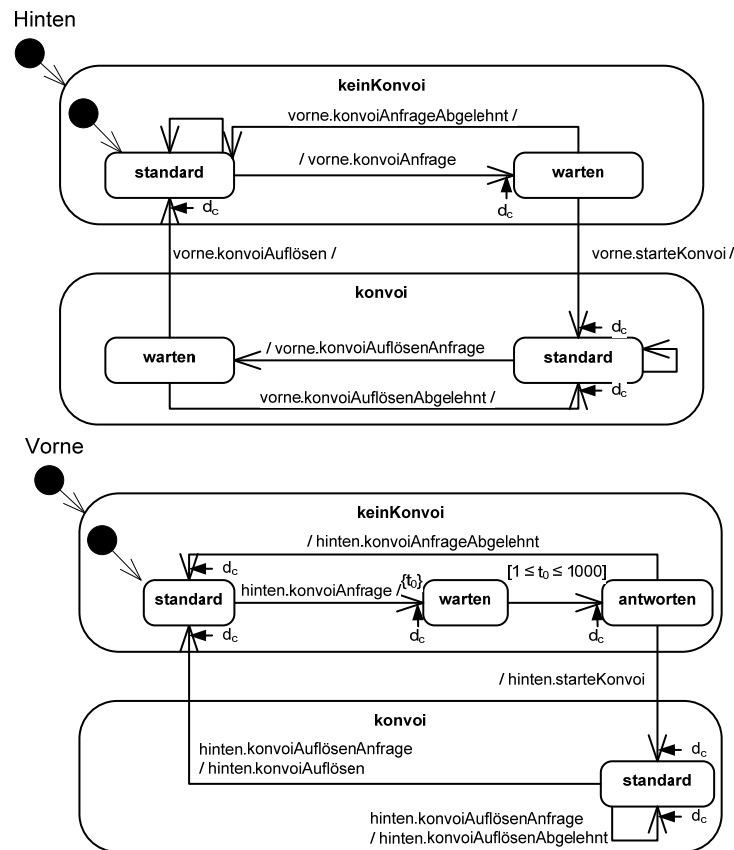


Abbildung 7: Realtime Statecharts für die Rollen vorne und hinten des Musters KonvoiKoordination

Nachdem ein Muster spezifiziert wurde, können Sicherheitseigenschaften mittels Model Checking verifiziert werden. Für das Muster *KonvoiKoordination* kann so beispielsweise bewiesen werden, dass nie die hintere Rolle im Zustand *Konvoi* ist, während sich die vordere Rolle im Zustand *KeinKonvoi* befindet.

Nachdem Koordinationsmuster spezifiziert und verifiziert wurden, werden diese in einer Komponente angewendet. Dabei können gegebenenfalls die Realtime Statecharts der Rollen verfeinert werden, um die tatsächlich bestehenden Abhängigkeiten zwischen dem komponenteninternen und dem komponentenexternen Verhalten herzustellen. Um sicher-

zustellen, dass die verifizierten Eigenschaften bei der Verfeinerung nicht verletzt werden, wurde eine Verfeinerungsbeziehung formal definiert [GTB+03], die bei der Verfeinerung zu berücksichtigen ist. Im verfeinerten Verhalten der RailCab-Komponente (siehe Abbildung 8) wurde mittels eines Synchronisationsstatecharts umgesetzt, dass ein RailCab sowohl die Rolle *vorne* als auch die Rolle *hinten* in einem Konvoi einnehmen kann, aber nie beide gleichzeitig. Dies stellt sicher, dass ein voranfahrendes Fahrzeug nie Teil eines anderen Konvois ist. Hierzu mussten die Realtime Statecharts der Rollen verfeinert werden, indem die zusätzlichen Nachrichten *bildeKonvoi*, *istKonvoiOk*, *keinKonvoi* und *konvoiAuflösen* eingefügt wurden, die das Verhalten der Rollen-Statecharts mit dem des Synchronisations-Statecharts synchronisieren. Um die Korrektheit des Komponentenverhaltens zu garantieren, wird dann die einzelne Komponente verifiziert. Die Einhaltung der Sicherheitseigenschaften im Gesamtsystem ergibt sich aus der Einhaltung der Sicherheitseigenschaften der Muster und der Komponenten, die jeweils nur einzeln verifiziert werden müssen, im Zusammenhang mit der Verfeinerungsbeziehung zwischen Musterrollen und Komponenten.

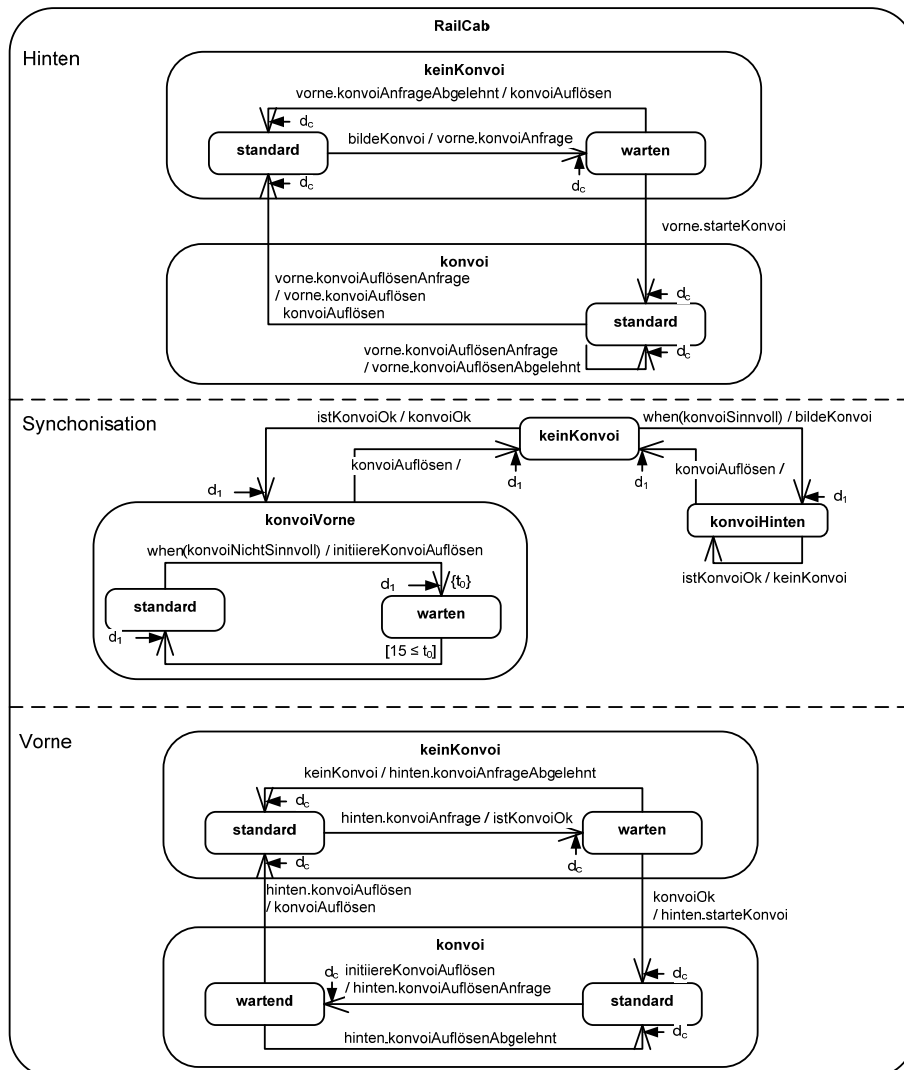


Abbildung 8: Verfeinertes Komponentenverhalten einer RailCab-Komponente

3.3.2 Regelung

Zur Regelung der Fahrzeuglängsbewegung wird zwischen zwei Betriebsmodi differenziert. Wird ein Fahrzeug einzeln betrieben, so befindet es sich im Modus Geschwindigkeitsregelung. Fährt ein RailCab als Folgefahrzeug in einem Konvoi, so muss die Abstandsregelung aktiviert werden, die einer Positionsregelung mit bewegtem Zielpunkt entspricht. Die Fahrzeugregelung ist als Kaskadenregelung ausgelegt mit einer unterlagerten Geschwindigkeits- und einer überlagerten Abstandsregelung. Die Betriebsart wird mittels eines Schalters gewählt, wobei der Schaltbefehl innerhalb des Fahrprofilgenerators erzeugt wird sowie von der Konfigurationssteuerung im reflektorischen Operator beeinflusst wird. Der Ausgang des Geschwindigkeitsreglers liefert die Sollschubkraft. Aus dieser werden mittels einer Arbeitspunktsteuerung die Sollströme des Linearmotors bestimmt (siehe Abbildung 9).

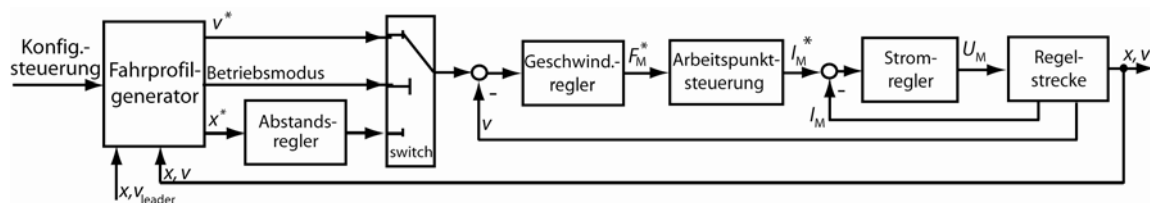


Abbildung 9: Regelkreisstruktur

Als Bindeglied zwischen Regler und Steuerung ist ein Fahrprofilgenerator angeordnet. Dieser generiert anhand von Messgrößen (z. B. Geschwindigkeit, Position etc.) und Streckendaten die Sollwertverläufe für die Fahrzeugregler sowie die Befehle für die Reglerumschaltung. Einzelheiten zur Fahrzeugregelung sowie zur Fahrprofilgenerierung können [HTS+08a] entnommen werden.

Die Reglerauslegung erfolgt mit Werkzeugen für den modellbasierten Entwurf (Matlab/Simulink). Die Verwendung einer Rapid Prototyping Plattform (Fa. dSPACE) macht es zudem möglich, mittels Hardware-in-the-Loop-Simulation (HiL) das Verhalten von einzelnen Systemteilen präzise zu analysieren. Folglich ist möglich, sowohl den Reglerentwurf zu validieren, als auch das Zusammenspiel zwischen Konvoikoordination und Regelung unter Einbeziehung realer Systemkomponenten (z. B. Fahrzeug-Fahrzeug-Kommunikation) zu testen. Die Umschaltung der Regler selbst ist flachheitsbasiert implementiert. Die Methode ermöglicht strukturbedingt einfache Nachweise der notwendigen Stabilität und stellt die Stoßfreiheit der Stellgrößen sicher. Außerdem kann durch Sie auf weitere Sicherheits- und Komfortaspekte der Reglerumschaltung eingegangen werden, wie z. B. Trajektorienfolge und Vermeiden von Überschwingern [OT+08].

In Abbildung 10 sind Simulationsergebnisse für eine Konvoibildung von zwei Fahrzeugen dargestellt. Die Koordinationsstruktur, die in der realen Funkverbindung abgebildet ist, initialisiert die Konvoibildung. Dies bedeutet jedoch nicht, dass direkt zur Abstandsregelung übergegangen werden darf. Es wird dem Folgefahrzeug dadurch lediglich erlaubt, in den

Zustand *KonvoiBilden* zu wechseln, um einen definierten Ablauf zur Fahrzeugannäherung zu starten. Das Folgefahrzeug durchläuft nun die Phasen der Konvoibildung bis es in den Modus der *Folgefahrt* gelangt, in dem erst die Abstandsregelung aktiviert wird. Dies stellt eine Verfeinerung des diskreten Zustandsmodells aus Abbildung 8 dar und wird im nächsten Abschnitt detaillierter erläutert.

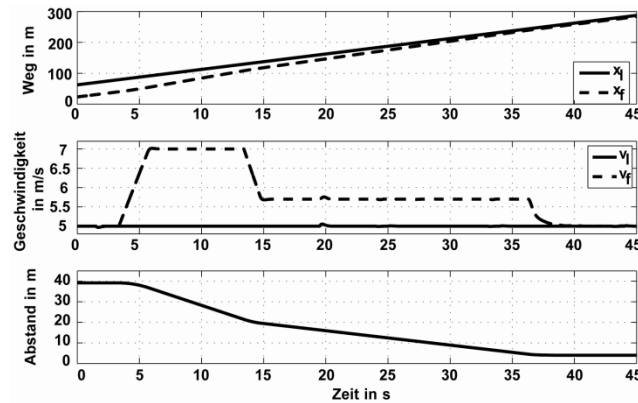


Abbildung 10: Simulationsergebnis einer HiL-Simulation zur Konvoibildung

3.3.3 Rekonfiguration

Nachdem sowohl das diskrete Verhalten als auch die Regelung spezifiziert wurde, muss abschließend noch das kontinuierliche Verhalten der Komponente integriert werden. Die Rekonfiguration der Regler ist im Beispiel der RailCab-Konvoifahrt nicht nur abhängig von der diskreten Echtzeitkommunikation sondern bei der Bildung und dem Auflösen des Konvois auch vom Abstand der Fahrzeuge. So wurde das Synchronisationsstatechart weiter verfeinert, indem die bestehenden Zustände durch weitere Zustände und Transitionen mit Bezug auf den Abstand verfeinert wurden [HTS+08b].

Für jeden der modellierten Zustände wurde die jeweilige Reglerkonfiguration spezifiziert sowie Parameter für die Regelung, zum Beispiel der Sollabstand d^* , die maximale Differenz der Sollgeschwindigkeiten der RailCabs Δv_{max} sowie die Sollposition des hinteren Railcabs x_f^* , gesetzt. Das so erweiterte Synchronisationsstatechart der RailCab-Komponenten ist in Abbildung 11 dargestellt. Hier ist spezifiziert, dass im Zustand *Kein-Konvoi* lediglich der Fahrprofilgenerator die Sollgeschwindigkeit des RailCabs vorgibt, während im Zustand *FolgeFahrt* zusätzlich der Abstandsregler genutzt wird. So wird in der Konvoifahrt der entsprechende Sicherheitsabstand zum vorherfahrenden Fahrzeug gewährleistet.

Die MechatronicUML ermöglicht zu überprüfen, ob die Einbettung der Regler konform zur Spezifikation des diskreten Verhaltens der Realtime Statecharts ist. Hier wird vor allem überprüft, ob die Dauern der Reglerumschaltung zu den spezifizierten Deadlines an den Transitionen des Statecharts passen [GBS+04].

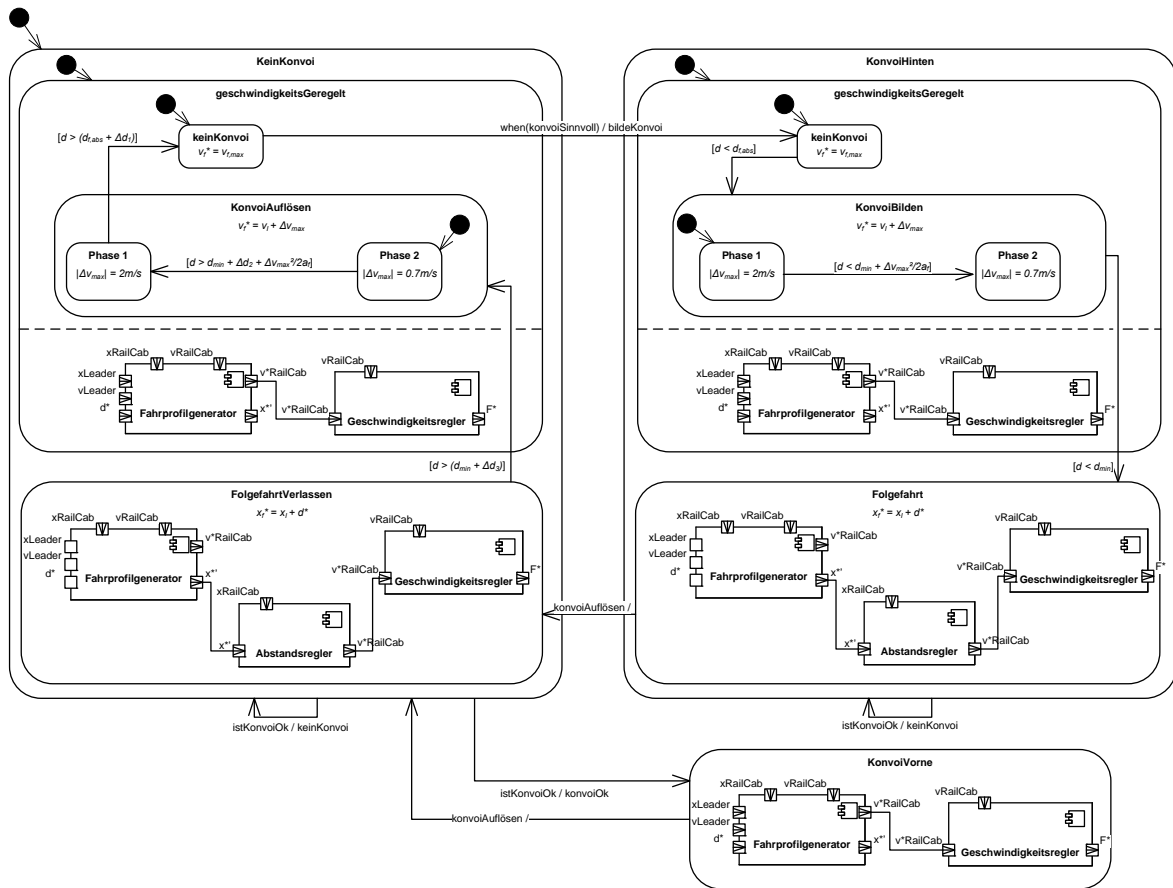


Abbildung 11: Hybrides Synchronisationsstatechart der RailCab-Komponente

4 Codegenerierung und Ausführung

Das Softwaremodell enthält alle relevanten Daten, um Quelltext zu generieren, der die spezifizierten Zeitbedingungen einhält. Dies erlaubt vor allem eine Periode für die periodische Ausführung zu berechnen. Im Zusammenhang mit einem Echtzeitscheduling in einem Echtzeitbetriebssystem garantiert dies, dass bei der Ausführung alle spezifizierten Zeitbedingungen eingehalten werden.

Durch die sich zur Laufzeit verändernden Anforderungen von selbstoptimierenden Anwendungen sind neue Techniken im Bereich der Systemsoftware nötig, da beispielsweise ein statisches Echtzeitbetriebssystem (RTOS) zu ineffizient wäre. Aus diesem Grund wurde ein selbstoptimierendes RCOS/RTOS-System entwickelt [OB04], welches sich zur Laufzeit optimal an die sich dynamisch ändernden Anforderungen anpasst. Dazu werden unterschiedliche Profile bestimmt, die verschiedene Ressourcenanforderungen definieren. So können z.B. für Regler mit unterschiedlicher Regelgüte Profile für jeden dieser Regler bestimmt werden. Die hybriden Rekonfigurationscharts werden mit den dafür relevanten Informationen annotiert [BGG+04]. Zur Laufzeit entscheidet das RTOS anhand der Qualität des Systems und den aktuell vorhandenen Ressourcen, welches dieser Profile aktiviert wird, um die Qualität des Systems zu erhöhen. Ressourcen von temporär nicht benötigten

Betriebssystemdiensten können anderen Anwendungen zur Verfügung gestellt werden, bis die Dienste wieder benötigt werden. Ebenfalls ist diese Umverteilung möglich mit Ressourcen, die von Anwendungen für Notfallszenarien reserviert wurden, aber ungenutzt sind. Das System genügt dabei strengen Echtzeit- und Sicherheitsbedingungen.

Neben der Ausführung auf einer Echtzeithardware kann der generierte Quelltext auch simuliert werden. Um dies zu ermöglichen, wurde die MechatronicUML in der Fujaba Real-Time Tool Suite [BGH+07, PTH+10] prototypisch umgesetzt. Mit der Fujaba Real-Time Tool Suite wurde die Architektur sowie die Statecharts spezifiziert. Die Werkzeuge CAMEL-View [HJ06] sowie Matlab/Simulink wurden zur Spezifikation der Regelung genutzt. Abbildung 12 zeigt Simulationen eines RailCab-Konvois. Die Abbildung rechts unten zeigt eine Simulation im Werkzeug CAMEL-View. Bei der Simulation kann sich der Entwickler Plots für verschiedene kontinuierliche Ausgänge sowie eine 3D-Darstellung anzeigen lassen. Die diskreten Anteile der Software sind nur indirekt, z.B. an Änderungen der Werte in den Plots, zu sehen. In einer weiteren Simulation, links oben in der Abbildung gezeigt, wurde daher in die 3D-Darstellung zusätzlich eine abstrakte Sicht auf das Statechartmodell inklusive des aktiven Zustands eingebettet.

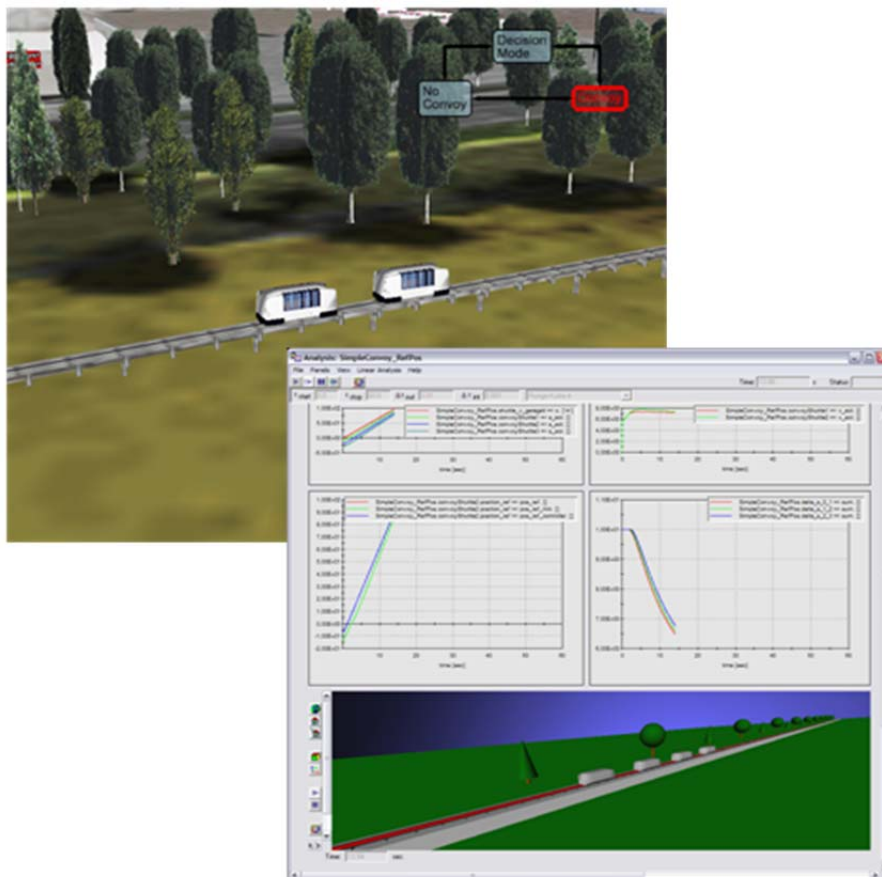


Abbildung 12: Simulation der spezifizierten Modelle

5 Resümee und Ausblick

Die hier vorgestellte MechatronicUML zum Entwurf der softwarespezifischen Anteile in mechatronischen Systemen wurde an Hand mehrerer Forschungsprojekte evaluiert. Zum Beispiel fahren die RailCabs auf der Teststrecke mit der so spezifizierten Software.

Bei der Entwicklung mechatronischer Systeme wird derzeit das Wissen der Entwickler um vorhandene mechanische oder softwarebasierte Lösungen zur Realisierung von Anforderungen kaum erfasst. Daher wurden erste Forschungen begonnen, um auf Basis der Technologie Semantic Web eine effektive Speicherung und Austausch von Lösungswissen in den Branchenwertschöpfungsketten zu ermöglichen. In Bezug auf die Software und deren Entwurf ist hier vor allem zu untersuchen, inwieweit wiederkehrende Softwarelösungen für die Anforderungen mechatronischer Systeme in einem Semantic Web abgelegt, aufgefunden und an das konkrete zu entwickelnde Produkt angepasst werden können.

Die Integration mehrerer sicherheitskritischer Anwendungen auf eine Ausführungsplattform stellt ein großes Problem bei der Entwicklung selbstoptimierender mechatronischer Systeme dar. Es muss zu jedem Zeitpunkt gewährleistet sein, dass diese Anwendungen sich nicht negativ gegenseitig beeinflussen können. Um dies zu garantieren, wurde ein Virtual Machine Monitor [BK09, KBS09] entwickelt, der mehrere Echtzeitsysteme mit ihren Anwendungen auf eine Ausführungsplattform abbildet. Die Virtualisierung der Echtzeitsysteme ermöglicht eine strenge temporale als auch eine räumliche Trennung auf Basis der Instruktionssatzarchitektur. Derzeit wird daran gearbeitet, die Konzepte zur Codegenerierung mit diesem Ansatz zu integrieren.

Danksagungen

Diese Arbeit ist im Sonderforschungsbereich (SFB) 614 „Selbstoptimierende Systeme des Maschinenbaus“ der Universität Paderborn entstanden. Die Autoren danken der Deutschen Forschungsgemeinschaft für die Förderung des Vorhabens.

Diese Arbeit ist weiterhin im Rahmen des Verbundprojekts „ENTIME: Entwurfstechnik Intelligente Mechatronik“ entstanden. Das Projekt ENTIME wird vom Land NRW sowie der EUROPÄISCHEN UNION, Europäischer Fonds für regionale Entwicklung, „Investition in unsere Zukunft“ gefördert.

Literatur

- [AD94] ALUR, R.; DILL, D.: A Theory of Timed Automata. In: Theoretical Computer Science, Band 126, S. 183–235, 1994
- [BGG+04] BURMESTER, S.; GEHRKE, M.; GIESE, H.; OBERTHÜR, S.: Making Mechatronic Agents Resource-aware in order to Enable Safe Dynamic Resource Allocation. In Proc. of Fourth ACM International Conference on Embedded Software 2004 (EMSOFT 2004), Pisa, Italien (B. Georgio, ed.), S. 175-183, ACM Press, September 2004

- [BGH+07] BURMESTER, S.; GIESE, H.; HENKLER, S.; HIRSCH, M.; TICHY, M.; GAMBUTTA, A.; MÜNCH, E.; VÖCKING, H.: Tool Support for Developing Advanced Mechatronic Systems: Integrating the Fujaba Real-Time Tool Suite with CAMEL-View. In: Proc. of the 29th International Conference on Software Engineering (ICSE), Minneapolis, Minnesota, USA, S. 801-804, IEEE Computer Society Press, Mai 2007
- [BGT05] BURMESTER, S.; GIESE, H.; TICHY, M.: Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML. In: Model Driven Architecture: Foundations and Applications, Lecture Notes in Computer Science (LNCS), Band 3599, S. 47–61, Springer Verlag, August 2005
- [BK09] BALDIN, D.; KERSTAN, T.: Proteus, a hybrid Virtualization Platform for Embedded Systems. In: Rettberg, Achim; Rammig, Franz Josef (Hrsg.) Analysis, Architectures and Modelling of Embedded Systems, 14. - 16. Sep. 2009 IFIP WG 10.5, Springer-Verlag, 2009
- [GB03] GIESE, H.; BURMESTER, S.: Real-Time Statechart Semantics. Technical Report tr-ri-03-239, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, 2003
- [GBS+04] GIESE, H.; BURMESTER, S.; SCHÄFER, W.; OBERSCHELP, O.: Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In: Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA, S. 179-188, ACM Press, November 2004
- [Geh05] GEHRKE, M.: Entwurf mechatronischer Systeme auf Basis von Funktionshierarchien und Systemstrukturen. Dissertation, Universität Paderborn, 2005
- [GFD+08] GAUSEMEIER, J.; FRANK, U.; DONOTH, J.; KAHL, S.: Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme des Maschinenbaus. Konstruktion Teil 1 7/8-2008, Teil 2 9/2008
- [GH06] GIESE, H.; HENKLER, S.: A Survey of Approaches for the Visual Model-Driven Development of Next Generation Software-Intensive Systems. In: Journal of Visual Languages and Computing, Band. 17, S. 528-550, Dezember 2006
- [GHH+08] GIESE, H.; HENKLER, S.; HIRSCH, M.; ROUBIN, V.; TICHY, M.: Modeling Techniques for Software-Intensive Systems. In Designing Software-Intensive Systems: Methods and Principles (Dr. Pierre-F. Tiako, ed.), S. 21-58, Langston University, OK, 2008
- [GKP+10] GAUSEMEIER, J.; KAISER, L.; POOK, S.; NYBEN, A.; TERFLOTH, A.: Rechnerunterstützte Modellierung der Prinziplösung mechatronischer Systeme. In: 7. Paderborner Workshop "Entwurf mechatronischer Systeme", 18.-19. März, 2010, HNI Verlag, Paderborn, 2010
- [GSG+09] GAUSEMEIER, J.; SCHÄFER, W.; GREENYER, J.; KAHL, S.; POOK, S.; RIEKE, J.: Management of Cross-Domain Model Consistency During the Development of Advanced Mechatronic Systems. In: Proceedings of the 17th International Conference on Engineering Design (ICED'09), Band. 6, S. 1–12, Design Society, August 2009
- [GTB+03] GIESE, H.; TICHY, M.; BURMESTER, S.; SCHÄFER, W.; FLAKE, S.: Towards the Compositional Verification of Real-Time UML Designs. In: Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-11), S. 38–47. ACM Press, September 2003
- [HJ06] HAHN, M.; JÄKER, K.-P.: Domänenübergreifende Modellbildung eines aktiv gefederten Nutzfahrzeugs (CAMEL-View TestRig). In Isermann, R. (Hrsg.): Fahrdynamik-Regelung. Modellbildung, Fahrerassistenzsysteme, Mechatronik, S. 117-136, Vieweg, Wiesbaden, 2006

- [HTS+08a] HENKE, C.; TICHY, M.; SCHNEIDER, T.; BÖCKER, J.; SCHÄFER, W.: Organization and Control of Autonomous Railway Convoys. 9th International Symposium on Advanced Vehicle Control (AVEC 08), 6. - 9. Oktober 2008, Kobe, Japan, 2008
- [HTS+08b] HENKE, C.; TICHY, M.; SCHNEIDER, T.; BÖCKER, J.; SCHÄFER, W.: System Architecture and Risk Management for Autonomous Railway Convoys. 2nd Annual IEEE Systems Conference, 7. - 10. April 2008, Montreal, Kanada, 2008
- [KBS09] KERSTAN, T.; BALDIN, D.; SCHOMAKER, G.: Formale Bestimmung von Systemparametern zum transparenten Scheduling virtueller Maschinen unter Echtzeitbedingungen. In: Informatik aktuell (Tagungsband Echtzeit 2009), 19. - 20. November 2009 Fachauschuß Echtzeitsysteme der Gesellschaft für Informatik und der VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA), Springer-Verlag, 2009
- [OB04] OBERTHÜR, S.; BÖKE, C.: Flexible Resource Management - A framework for self-optimizing real-time systems. In: Kleinjohann, Bernd; Gao, Guang R.; Kopetz, Hermann; Kleinjohann, Lisa; Rettberg, Achim (Hrsg.) Proceedings of IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES'04), 23. - 26. August 2004, Kluwer Academic Publishers, 2004
- [OHG04] OBERSCHELP, O.; HESTERMEYER, T.; GIESE, H.: Strukturierte Informationsverarbeitung für selbstoptimierende mechatronische Systeme. In Proc. of the Second Paderborner Workshop Intelligente Mechatronische Systeme, HNI-Verlagsschriftenreihe, Band 145, S. 43-56, Paderborn, 2004
- [OT+08] OSMIĆ, S.; TRÄCHTLER, A.: Flatness-based online controller reconfiguration. 34th Annual Conference of IEEE IECON, November 2008, USA, Florida, 2008
- [PTH+10] PRIESTERJAHN, C.; TICHY, M.; HENKLER, S.; HIRSCH, M.; SCHÄFER, W.: Fujaba4Eclipse Real-Time Tool Suite. In Model-Based Engineering of Embedded Real-Time Systems (MBEERTS), LNCS, Springer, 2010 (akzeptiert)
- [Sch94] SCHÜRR, A.: Specification of Graph Translators with Triple Graph Grammars. In: Graph-Theoretic Concepts in Computer Science, 20th International Workshop, LNCS, Band. 903, S. 151–163, 1994
- [SW07] SCHÄFER, W.; WEHRHEIM, H.: The Challenges of Building Advanced Mechatronic Systems. In FOSE '07: 2007 Future of Software Engineering, S. 72-84, IEEE Computer Society, 2007
- [VDI2206] VEREIN DEUTSCHER INGENIEURE (VDI): VDI-Richtlinie 2206 – Entwicklungsmethodik für mechatronische Systeme. Beuth-Verlag, Berlin, 2004

Autoren

Prof. Dr. Wilhelm Schäfer, Jahrgang 1954, ist Professor für Praktische Informatik (Softwaretechnik) an der Universität Paderborn am Institut für Informatik, zuvor war er von 1991 bis 1994 Professor für Praktische Informatik (Softwaretechnik) an der Universität Dortmund im Fachbereich Informatik. In den Jahren 1987 bis 1990 war er Leiter der Forschungs- und Entwicklungsabteilung der STZ Gesellschaft für Softwaretechnologie mbH, nachdem er von 1986 bis 1987 eine Assistenzprofessur an der McGill Universität in Montreal/Kanada innehatte. Er promovierte 1988 an der Universität Osnabrück im Bereich Softwaretechnik/Softwarewerkzeuge. Wilhelm Schäfer ist derzeit Vizepräsident für Forschung der Universität Paderborn, Chair der International Graduate School of Dynamic Intelligent Systems der Universität Paderborn und Teilprojektleiter im SFB 614. In For-

schung und Lehre beschäftigt er sich mit lernenden Verfahren zum Re-Engineering, der Spezifikation und Verifikation verteilter Echtzeitsysteme sowie der zugehörigen Entwicklungsprozesse.

Tobias Eckardt, Jahrgang 1984, ist seit 2009 wissenschaftlicher Mitarbeiter im Software Quality Lab (s-lab) der Universität Paderborn. Während seines Studiums der Informatik forschte er sowohl an den Softwareentwicklungsmethoden für eingebettete Echtzeitsysteme im Bereich der Mechatronik als auch im Bereich des modellbasierten und automatisierten Testens von Softwaresystemen. Nach Abschluss seines Studiums arbeitet er nun im Projekt ENTIME an einer Entwurfstechnik für intelligente mechatronische Systeme.

Christian Henke, Jahrgang 1975, hat Elektrotechnik an der Universität Paderborn studiert. Nach seinem Hochschulabschluss im Jahre 2004 war er wissenschaftlicher Mitarbeiter in der Fachgruppe Leistungselektronik und Elektrische Antriebstechnik (LEA) und arbeitete im Forschungsprojekt „Neue Bahntechnik Paderborn – RailCab“. Dort befasste er sich mit der Entwicklung von autonomen, Linearmotor getriebenen Bahnfahrzeugen. Seit 2009 ist er wissenschaftlicher Mitarbeiter in der Fachgruppe Regelungstechnik und Mechatronik (RtM).

Lydia Kaiser, Jahrgang 1982, hat Physik an der Universität Paderborn studiert. Seit 2007 ist sie wissenschaftliche Mitarbeiterin am Lehrstuhl für Produktentstehung bei Prof. Gausemeier am Heinz Nixdorf Institut der Universität Paderborn. Ihr Forschungsschwerpunkt liegt im Bereich der Entwicklungsmethodik mechatronischer Systeme.

Timo Kerstan, Jahrgang 1979, ist seit seinem Informatik Hochschulabschluss im Jahr 2006 wissenschaftlicher Mitarbeiter der Fachgruppe Entwurf paralleler Systeme unter der Leitung von Prof. Dr. Franz Josef Rammig. Er forscht seit 2008 im Sonderforschungsbereich 614 „Selbstoptimierende Systeme des Maschinenbaus“. Sein Arbeitsschwerpunkt liegt im Bereich Echtzeitbetriebssysteme und Virtualisierung von eingebetteten Echtzeitsystemen.

Jan Rieke, Jahrgang 1982, ist seit 2009 Doktorand der International Graduate School Dynamic Intelligent Systems und arbeitet im Fachgebiet Softwaretechnik an der Universität Paderborn. Seit seinem Hochschulabschluss in Informatik im Jahr 2008 forscht er im Sonderforschungsbereich 614 „Selbstoptimierende Systeme des Maschinenbaus“. Seine Schwerpunkte sind dabei die modellbasierte Entwicklung und insbesondere das Konsistenz-Management von Modellen verschiedener Disziplinen im Entwicklungsprozess mechatronischer Systeme.

Dr. Matthias Tichy, Jahrgang 1978, ist seit 2009 Senior Researcher im Software Quality Lab (s-lab) der Universität Paderborn. Nach seinem Hochschulabschluss in Wirtschaftsinformatik im Jahre 2002 forschte er im Sonderforschungsbereich 614 „Selbstoptimierende Systeme des Maschinenbaus“. Er promovierte 2009 an der Universität Paderborn im Institut für Informatik zum Thema „Gefahrenanalyse selbstoptimierender Systeme“. Sein Ar-

beitsschwerpunkt ist die modellbasierte Entwicklung verlässlicher, mechatronischer Systeme.