Integration hybrider Modellierungstechniken in CAMeL-View[1]

Dr. Matthias Tichy, Dr. Martin Hirsch, Dipl.-Inform. (FH) Christopher Brink, Prof. Dr. Wilhelm Schäfer, Christopher Gerking
Fachgebiet Softwaretechnik, Heinz Nixdorf Institut, Universität Paderborn
Warburger Straße 100, 33098 Paderborn
Tel. 05251/60-5008, Fax. 05251/60-3530
E-Mail: [mtt|mahirsch|Christopher.Brink|wilhelm]@upb.de, cgerking@mail.upb.de

Dr.-Ing. Martin Hahn
iXtronics GmbH
Technologiepark 9, 33100 Paderborn
Tel. 05251/41785-10, Fax. 05251/41785-29
E-Mail: Martin.Hahn@iXtronics.com

Zusammenfassung

Bei der Entwicklung mechatronischer Systeme stellt die eingebettete Software heutzutage einen großen Teil der Wertschöpfung dar. Neben der herkömmlichen Regelung kontinuierlicher Werte werden in einem immer größeren Maß zustandsbasierte Echtzeitprotokolle zur Koordination vernetzter, mechatronischer Systeme eingesetzt. Als Reaktion auf die zunehmende Komplexität werden modellbasierte Ansätze zum Entwurf und Testen dieser hybriden (kontinuierlichen / diskreten) Software eingesetzt. Im Rahmen des von der DFG geförderten Transferprojekts „Hybride Modellierung“ wird das Modellierungswerkzeug CAMeL-View um die Modellierungssprache Mechatronic UML erweitert. Die Mechatronic UML ist eine Modellierungssprache für die Kommunikation und Koordination in vernetzten mechatronischen Systemen. Sie unterstützt die Modellierung von zustandsbasiertem Echtzeitverhalten. Ziel des Transferprojekts ist eine Integration der Modellierungsmöglichkeiten in das Werkzeug CAMeL-View. In diesem Papier werden anhand eines Direktvergleichs mit dem Werkzeug Matlab und den Toolboxen Simulink und Stateflow die Neuerungen im Werkzeug CAMeL-View diskutiert.

Schlüsselwörter

Modellbasierte Entwicklung, mechatronische Systeme, Zustandsautomaten

---

[1] Diese Arbeit ist im „Transferprojekt T2 - Hybride Modellierung“, Universität Paderborn, entstanden und wurde auf seine Veranlassung unter Verwendung der ihm von der Deutschen Forschungsgemeinschaft zur Verfügung gestellten Mittel veröffentlicht.

1 Einleitung

Der Anteil von Software in mechatronischen Systemen steigt in den letzten Jahren stark an. Durch die immer stärkere Vernetzung mechatronischer Systeme wird Software neben der Regelung immer mehr auch zur nachrichtenbasierten Kommunikation und Koordination zwischen den einzelnen verteilten mechatronischen Systemen eingesetzt. Diese Kommunikation geht über die Aufnahme von System- und Umweltdaten durch Sensorik hinaus. Hier werden ggf. komplexe Zustandsinformationen über entsprechende Protokolle und zugrunde liegende Kommunikationskanäle ausgetauscht, die dann wieder das Verhalten bzw. die zugrunde liegenden Berechnungen der einzelnen Komponenten massiv beeinflussen können. Diese Entwicklung führt zu äußerst komplexer hybrider (kontinuierlicher / diskreter) Software.

Mechatronische Systeme werden oftmals in Bereichen eingesetzt, in denen die Einhaltung harter Echtzeitbedingungen unverzichtbar für die Sicherheit und Funktionsfähigkeit der Systeme ist. Um die Software dieser Systeme zu entwerfen, wird dementsprechend eine Entwicklungsumgebung benötigt, die es erlaubt sowohl die Regelung des physikalischen Systems als auch die diskrete Kommunikation und Koordination dieser Systeme zu entwickeln. Ein besonderes Augenmerk muss dabei auf der Integration der beiden Aspekte liegen. Hier bieten sich modellbasierte Sprachen an, die es erlauben auf einer abstrakteren Ebene als Programmquelltext die Struktur und das Verhalten dieser Systeme zu beschreiben.

Im Rahmen dieses Papiers stellen wir einen Ansatz vor, der das Werkzeug CAMeL-View der Firma iXtronics GmbH [Lüc90, Hah95, NH96, Hah99, HJ06] mit den Konzepten der Mechatronic UML [BGT05, SW07] kombiniert. Beide Ansätze unterstützen die modellbasierte Entwicklung der Regelung mechatronischer Systeme. CAMeL-View fokussiert bisher auf die Modellierung mechanischer, hydraulischer und regelungstechnischer Aspekte; wobei ein besonderer Fokus auf der physikalischen Modellbildung mechatronischer Systeme liegt, d. h., dass zum Beispiel für die Mechanik eigene Elemente, wie Starrkörper und Gelenke zur Verfügung stehen.

Die Mechatronic UML fokussiert dagegen auf geeignete Techniken zur Modellierung der nachrichtenbasierten (diskreten) Echtzeitkoordination zwischen mechatronischen Systemen unter harten Echtzeiteigenschaften. Des Weiteren wird die Integration der (kontinuierlichen) Modelle zur Regelung und Steuerung mit den diskreten Modellen unterstützt. Die Mechatronic UML ermöglicht des Weiteren den Einsatz ausgefeilter formaler Verifikationstechniken [GTB+03, GBS+04, BBD+02] zur Prüfung der Korrektheit der Modelle.

Im Rahmen eines von der DFG geförderten Transferprojekts werden die Konzepte der Mechatronic UML in das Werkzeug CAMeL-View integriert. Der aktuelle Stand der Arbeiten in Form der Sprachdefinition wird in diesem Papier vorgestellt. Insbesondere wird ein Vergleich mit dem Stand der Technik am Beispiel von Matlab und den Toolboxen Simu-

link und Stateflow gezogen. Hierbei wird vor allem der Fokus auf die Modellierung der diskreten Anteile unter besonderer Berücksichtigung von Echtzeit gelegt.

Als durchgängiges Beispiel für die Darstellung der Konzepte wird im Folgenden das RailCab Forschungsprojekt verwendet. In diesem Projekt wurde ein Transportsystem entwickelt, welches auf kleinen autonomen und führerlosen RailCabs beruht. Diese eignen sich sowohl für den Güter-, als auch für den Personenverkehr und reduzieren durch die Bildung von Konvois auf der Strecke den Luftwiderstand und somit ihren Energieverbrauch.

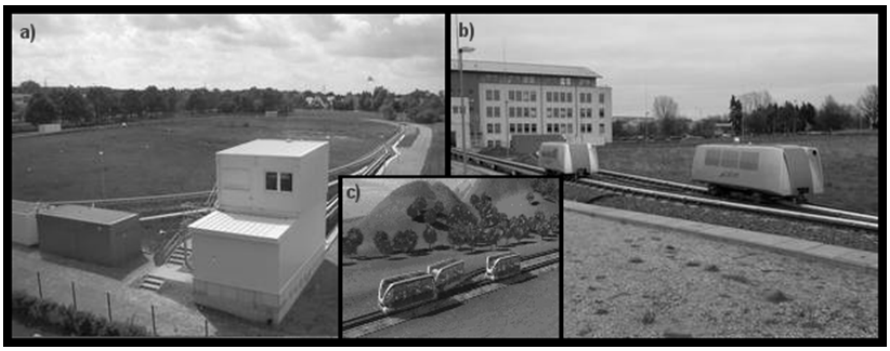

Bild 1: „RailCab – Neue Bahntechnik Paderborn“ (a) Versuchsstrecke des RailCab in Paderborn, (b) RailCabs auf der Versuchsstrecke, (c) Simulierte Konvoi-Bildung

Der Betrieb der entwickelten Bahntechnik wird auf einer Versuchsstrecke an der Universität Paderborn (siehe Bild 1) getestet. Die Herausforderung beim Bilden eines Konvois besteht dabei darin, einen gewissen Sicherheitsabstand einzuhalten und für den Fall, dass das vordere RailCab abbremst, nicht aufzufahren. Um dies zu gewährleisten, führen die RailCabs umfangreiche Kommunikations- und Koordinationsaktivitäten aus.

Das Papier ist in die folgenden Abschnitte unterteilt: Abschnitt 2 enthält eine Darstellung des Stands der Technik in Bezug auf die Modellierung der Software mechatronischer Systeme. In Abschnitt 3 wird die neue Modellierungssprache vorgestellt und vom Stand der Technik abgegrenzt. Das Papier schließt mit einem Fazit und einem Ausblick auf zukünftige Arbeiten in Abschnitt 4.

2 Stand der Technik

Die Kommunikation und Koordination zwischen verteilten mechatronischen Systemen basiert auf dem Austausch von komplexen Nachrichten. Die Reihenfolge des Nachrichtenaustauschs bestimmt den Zustand der Systeme. Zur Modellierung dieses diskreten Verhaltens existieren verschiedene Modellierungssprachen.

2.1 Statecharts

Modellierungssprachen für diskretes Verhalten basieren überwiegend auf Automaten. Automaten bestehen aus Zuständen und Transitionen zwischen diesen. Transitionen werden dafür verwendet, um Zustandsübergänge zu modellieren. Hierfür werden diese mit einem Quell- und einem Zielzustand verbunden. An eine Transition können zudem weitere Elemente (Nachrichten, boolesche Bedingungen) modelliert werden, um anzugeben, wann die Transition ausgeführt wird bzw. wann der Zustandswechsel stattfindet und welche Nachrichten ggf. beim Ausführen ausgelöst werden.

Statecharts [Har87] erweitern diese Automaten um Konzepte zur Bewältigung der Komplexität, wie Parallelität und Hierarchie. Hierarchie ermöglicht die Verfeinerung von Zuständen durch Unterzustände. Des Weiteren werden parallele Hierarchien unterstützt. Hierdurch kann der Entwickler nebenläufiges Verhalten spezifizieren.

Es gibt verschiedene Arten von Statecharts, wie in [Bee94] dargestellt, die sich sowohl in der Syntax als auch in der Semantik unterscheiden. Syntaktische Varianten ergeben sich zum Beispiel in Bezug darauf, ob Transitionen Zustände aus verschiedenen Hierarchieebenen verbinden dürfen oder ob das Nicht-Eintreffen von Nachrichten als Bedingung zum Schalten einer Transition spezifiziert werden kann.

Semantische Varianten ergeben sich vor allem aus der Behandlung des Schaltens von Transitionen und der Gültigkeit von versendeten bzw. empfangenen Nachrichten. Zum Beispiel ist es in einzelnen Statechartdialekten erlaubt, dass mehrere Transitionen in einem Ausführungsschritt schalten können (Run-To-Completion Semantik) und dadurch mehrere Zustände betreten und verlassen werden können. Dies kann zu beliebig vielen und unter Umständen sogar unendlich vielen Transitionsausführungen in einem Zeitschritt führen. Für diese Zeit kann das System nicht auf Einflüsse aus der Umgebung reagieren. Solch ein Verhalten ist für die Domäne der mechatronischen Systeme, die harten Echtzeitbedingungen genügen müssen, nicht adäquat.

In Bezug auf die Gültigkeit von Nachrichten ist in diesem Zusammenhang die im Statechartdialekt gewählte Semantik wesentlich, wann verschickte Nachrichten für den Empfänger sichtbar sind. Falls diese im gleichen Zeitschritt sichtbar sind, kann es zu oben beschriebenen mehrfachen Transitionsausführungen kommen. In Verbindung mit Bedingungen für das Nicht-Eintreffen von Nachrichten kann es sogar zu Widersprüchen kommen, wenn bei der Ausführung einer Transition eine Nachricht versendet wird, die die Bedingungen für das Ausführen dieser Transition ungültig macht.

Varianten von Statecharts existieren auch hinsichtlich der Betrachtung von zeitlichen Aspekten. Dies ist vor allem für die Spezifikation des Verhaltens von Echtzeitsystemen sehr wichtig. Zeitaspekte betreffen zum einen die Spezifikation des Verhaltens in Bezug auf zeitliche Bedingungen, ob Transitionen schalten können. Zum anderen wird definiert, ob bei der Ausführung von Transitionen Zeit vergeht. Wenn in diesem Fall keine Zeit vergeht,

wird die sogenannte Nullzeitannahme getroffen. Das so spezifizierte Verhalten kann nur durch Abweichung vom spezifizierten Zeitverhalten implementiert werden.

Insgesamt betrachtet bedeutet dies, dass weit verbreitete Eigenschaften der Semantik von Statecharts wie die Null-Zeit-Annahme (s. auch die Diskussion in [Lee09] zum Thema Zeit) und Run-To-Completion Semantik ([LQV01, BLM02, DMY02, HMP92]) beim Schalten von Transitionen nicht passend für die hier betrachteten Systeme sind.

2.2 Matlab mit Simulink/Stateflow

Den Stand der Technik im industriellen Kontext zur Modellierung hybrider Systeme stellen Werkzeuge wie Matlab mit den Toolboxen Simulink, Stateflow und Stateflow-Coder dar. Dieses Werkzeug ermöglicht die Entwicklung der Software hybrider Systeme. Mit der Toolbox Simulink können Blockdiagramme modelliert werden, die das kontinuierliche Verhalten der Software, z. B. für eine Regelung, enthalten. Insbesondere werden hierarchische Blockdiagramme zur Strukturierung komplexer Modelle unterstützt.

Stateflow wurde als zusätzliches Tool zu Simulink entwickelt. Um hybride Systeme zu modellieren werden die Stateflowmodelle als Blöcke in Simulink abgebildet. Mit Stateflow lassen sich hierarchische, parallele Zustandsmodelle auf Basis von Automaten erstellen.

Stateflow ist ein event getriggertes Modellierungswerkzeug, es enthält keine eigenen Timer. Um Zeitverhalten in Stateflow zu modellieren, muss der Anwender Hilfskonstrukte wie `after` und `before` benutzen, um Zeittakte, die durch Timersignale aus Simulink kommen, in Stateflow zu zählen. Neuere Versionen unterstützen die Modellierung von Zeitbedingungen mit Hilfe von Zeitangaben, wie Sekunden und Millisekunden. Allerdings kann bei diesen Zeitangaben als relativer Zeitpunkt nur auf den Eintritt in den jeweils aktiven Zustand verwiesen werden und nicht auf beliebige vorherige Aktivitäten.

Stateflow geht bei der Modellierung auch von einer Nullzeit-Anahme aus. Weiterhin ist das Schaltverhalten von Transitionen in Stateflow abhängig von der graphischen Darstellung des Modells, da ohne Benutzervorgabe die Prioritäten der Transitionen von der Darstellung der Transitionen abhängen.

Problematisch ist weiterhin die gewählte Semantik für Nachrichten (bzw. Ereignisse). Wenn Nachrichten versandt werden und keine Transition auf Grund dieser Nachricht schaltet, werden die Nachrichten ignoriert und gelöscht. Dies führt zu schwierig zu findenden Fehlern in den Modellen und ist insbesondere für den betrachteten Fall der verteilten mechatronischen Systeme unpassend, da durch die Verteilung oftmals Nachrichten zwischen den Systemen versandt und damit in Puffern zwischengespeichert werden müssen. Dies führt dazu, dass eine passende Semantik für Nachrichten manuell implementiert werden muss und daher die Stateflow-Modelle umfangreicher und unübersichtlicher werden.

Bild 2: Ausschnitt aus dem Stateflow-Modell mit mehreren Nachrichten

Bild 2 zeigt einen kleinen Ausschnitt aus dem Stateflow-Modell des RailCabs, welches auf den realen Fahrzeugen auf der Teststrecke ausgeführt wird. Insgesamt besteht das Modell aus insgesamt sechs Hierarchieebenen mit über 50 Zuständen, 25 verschiedenen Nachrichten und 24 kontinuierlichen Werten, die zwischen Fahrzeugen ausgetauscht werden.

An der Transition in Bild 1 werden komplexe Operationen aufgerufen, die mit Hilfe von Embedded Matlab Funktionen überprüfen, ob eine bestimmte Menge an Nachrichten eingetroffen sind (Aufrufe von `checkQueue`), und wenn ja, diese dann verarbeitet und dabei die in den Nachrichten enthaltenen Daten ausliest (Aufrufe von `dequeue`). Problematisch ist hierbei unter anderem die sehr unübersichtliche Modellierung durch mehrere Funktionsaufrufe, die in komplexen und schlecht wartbaren Modellen resultiert.

Matlab unterstützt die automatische Generierung des Quelltextes aus den spezifizierten Simulink und Stateflow Modellen. Für den Einsatz in eingebetteten Systemen wird mit Hilfe von speziellen Erweiterungen (Real-Time Workshop, Embedded Coder) oder Fremdwerkzeugen Quelltext generiert, der auf die jeweilige Ausführungsplattform hinsichtlich Größe und Ressourcenverbrauch optimiert ist. Um eine Anbindung an eine Echtzeit-Hardware zu erhalten, sind weitere Tools wie z. B. die dSPACE Toolkette notwendig.

2.3 Mechatronic UML

Die im Rahmen des Sonderforschungsbereichs 614 entwickelte Mechatronic UML [BGT05, SW07] ist ein modellbasierter Ansatz zum Entwurf der Software mechatronischer Systeme. Sie fokussiert auf geeignete Techniken zur Modellierung der nachrichtenbasierten (diskreten) Echtzeitkoordination zwischen mechatronischen Systemen. Des Weiteren ist es möglich (kontinuierliche) Modelle zur Regelung und Steuerung in das diskrete Verhalten zu integrieren. Auf Basis der Modelle wird eine Unterstützung für die Überprüfung der entworfenen Modelle auf die Einhaltung sicherheitskritischer Eigenschaften geboten. Es existiert eine Quelltextsynthese für die automatische Generierung von Quelltext aus den Modellen. Der erzeugte Quelltext setzt die spezifizierten Echtzeiteigenschaften um.

Die Mechatronic UML unterstützt die Modellierung des diskreten Echtzeitverhaltens mit Real-Time Statecharts. Diese entstanden aus den UML State Machines. Ein wesentliches Problem, mit dem man bei der Verwendung von UML State Machines für Echtzeitsysteme konfrontiert ist, stellen die für Echtzeitverhalten ungenügenden Ausdrucksmöglichkeiten

dar. Vor diesem Hintergrund wurde eine Erweiterung um geeignete Konstrukte zur Beschreibung von Echtzeitaspekten untersucht, die zur Entwicklung der Real-Time Statecharts [BGT05] führte.

Eine wesentliche Erweiterung bei Real-Time Statecharts ist die Loslösung von der bei Statecharts verbreiteten Nullzeit-Annahme und Run-To-Completion Semantik. Daher werden für Transitionen eine minimale und eine maximale Zeitdauer in Form von Deadline Intervallen angegeben, die zwischen Aktivierung der Transition und Beendigung des Schaltens vergehen darf.

Darüber hinaus werden Uhren und Bedingungen über die Uhrenwerte zur Modellierung von zeitlichen Bedingungen und Zustandsinvarianten verwendet. Als Basis dienen hierbei Timed Automata [AD94]. Damit können die UML eigenen Konstrukte wie `after` und `when` nachgebildet werden, aber darüber hinaus auch komplexere Zeitbedingungen, die über mehrere Zustände reichen, modelliert werden. Die Auswahl, welche von mehreren aktivierten Transitionen schaltet, richtet sich nach dem Zeitpunkt der Aktivierung. Bei mehreren gleichzeitig aktivierten Transitionen entscheiden ggf. die Prioritäten der Transitionen. Des Weiteren unterstützen Real-Time Statecharts sowohl synchrone als auch asynchrone Kommunikation.

Die Mechatronic UML wurde im Softwarewerkzeug Fujaba Real-Time Tool Suite umgesetzt. Die Integration mit kontinuierlichen Modellen wurde bisher durch eine prototypische Toolkopplung mit dem Werkzeug CAMeL-View realisiert [BGH+07]. Im Rahmen des Transferprojekts wird nun durch eine nahtlose Integration der diskreten Modellierung in CAMeL-View ein interdisziplinäres Werkzeug zur Modellierung hybrider Systeme entwickelt.

3 Hybride Modellierung in CAMeL-View

3.1 CAMeL-View

Das Werkzeug CAMeL-View [Hah95, NH96, Hah99] unterstützt den modellbasierten Entwurf mechatronischer Systeme. Es unterstützt die Modellierung sowie den Test am Prüfstand und am Prototypen.

Bei der Modellierung wird ein physikalisch/topologisches Modell des Systems aufgebaut, bestehend aus mechanischen, hydraulischen und regelungstechnischen Bauteilen. Essentiell dabei ist, dass ein Modellbildungswerkzeug zur Verfügung steht, das die physikalische Modellbildung mechatronischer Systeme unterstützt, d. h. das z. B. für die Mechanik Massen und Gelenke zur Verfügung stehen und diese Modelle 1:1 für die Prüfstands- und Prototypenphase unter harten Echtzeitbedingungen verwendet werden können. Alle Systembestandteile können zudem in der Modellphase optimiert und verschiedene Varianten auf ihre

Einsatztauglichkeit mit Hilfe von Simulationsrechnungen, Frequenzgangsbetrachtungen und Modalanalysen untersucht werden.

Für die Modellierung kontinuierlichen Verhaltens werden in CAMeL-View Blockdiagramme genutzt. Es existieren u. a. Blöcke zur Modellierung linearer und nichtlinearer Differentialgleichungen, lineare Transferfunktionen sowie Blöcke auf Basis von Kennfeldern, falls dynamisches Verhalten nicht in Gleichungen darstellbar ist. Des Weiteren existieren Blöcke für die Modellierung von Mehrkörpersystemen (u. a. Starre Körper, Gelenke, Gelenke mit Steifigkeiten und Dämpfungen). Hydraulische Systeme werden durch Blöcke für hydraulische Kammern und hydropneumatische Speicher, Leitungen, Ventile, Drosseln und Blenden sowie Motoren, Pumpen und Zylinder unterstützt. Bild 3 zeigt die Modellierung von Blockdiagrammen in CAMeL-View.

Nach der Modellierung werden für die Modelle Leistungsdaten am Prüfstand überprüft. Dabei kann das System insbesondere auf Robustheit gegenüber Parameterschwankungen, Leistungsreserven und Dauerlaufeigenschaften hin überprüft werden. Dabei können entweder Messungen vorgenommen werden oder CAMeL-View TestRig für HiL-Applikationen verwendet werden. Bei HiL-Untersuchungen wird dazu der Teil auf dem Prüfstand aufgebaut, dessen physikalisches Verhalten besser untersucht und abgesichert werden soll. Die Identifikation der auf dem Prüfstand untersuchten Komponenten erlaubt es, diese im Modell zu identifizieren und die Auslegung des Gesamtsystems auf der Basis von Simulationsrechnungen abzusichern.

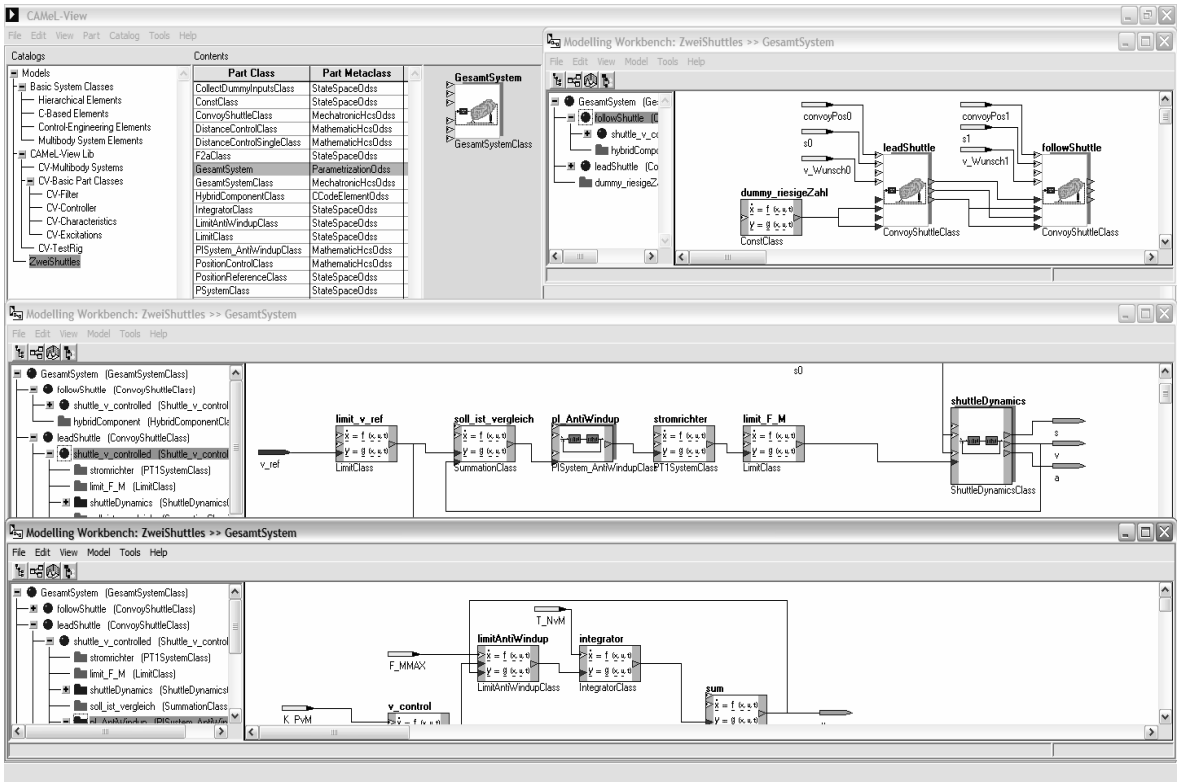

Bild 3: Modelle des physikalischen Systems und der Regler in CAMeL-View

Nach bestandenen Prüfstandstests wird ein Prototyp aufgebaut. Dabei liegt ein besonderes Augenmerk auf Eigenschaften, die in der Simulation nur mit unverhältnismäßig hohem Aufwand ermittelt werden können, wie z. B. der Bauteilverschleiß. Außerdem dienen die dabei ermittelten Messwerte zur Absicherung der modellbasierten Untersuchungen und stellen eine Wissensbasis für die weitere Entwicklung dar.

In allen drei Phasen des modellbasierten Entwurfs mechatronischer Systeme wird eine Entwicklungsumgebung benötigt, mit der man die Systemmodelle erstellen, simulieren und analysieren kann. Dabei ist die Simulation der Modelle in Echtzeit von besonderer Bedeutung. Genau für diesen integrierten Entwurfsprozess wurde CAMeL-View TestRig entwickelt, damit man im Modell oder am Prüfstand Einblick in das dynamische Verhalten der Systeme bekommt, bevor ein Prototyp gebaut wird. Für die Simulation wird neben der Unterstützung für die 3D-Simulation eine automatische Quelltextsynthese [NH96, Hah99] für die Verhaltensmodelle zur Verfügung gestellt.

3.2 Zustandsbasiertes Verhalten

Basierend auf der Mechatronic UML wird derzeit nun eine neue Modellierungssprache für zustandsbasiertes Echtzeitverhalten in CAMeL-View integriert. Die Sprache enthält wie andere statechartbasierte Sprachen Zustände und Transitionen. An den Transitionen können boolesche Bedingungen annotiert werden, die auf Basis der Werte von lokalen Variablen oder Ein-/Ausgängen, das Schalten einer Transition ermöglichen. Die Statecharts sind hierarchisch und erlauben auch die Spezifikation von parallelen Regionen.

Im Unterschied zu den übrigen Sprachen werden allerdings zum einen verschiedene Arten von Nachrichten unterstützt sowie besondere syntaktische Konstrukte für die Spezifikation zeitlicher Aspekte. Dies resultiert in typischerweise kompakteren und verständlicheren, und damit wartbareren, Modellen sowie einer erhöhten Ausdrucksmächtigkeit der Modellierungssprache im Vergleich zu Standardwerkzeugen. Des Weiteren ist im Vergleich zu Stateflow keine zusätzliche manuelle, fehlerträchtige Implementierung der für verteilte Systeme sinnvollen Nachrichtensemantik notwendig.

3.2.1 Nachrichtenarten

Es werden verschiedene Nachrichtenarten unterstützt. Diese unterscheiden sich vor allem dahingehend, ob sie zur Kommunikation zwischen Statecharts oder innerhalb von Statecharts genutzt werden. Die Kommunikation zwischen Statecharts wird mittels asynchroner Nachrichten modelliert. Asynchrone Nachrichten werden nach dem Versand durch die Ausführung einer Transition asynchron weiterverarbeitet. Dies beinhaltet sowohl den Transport zum Empfängerstatechart als auch die dortige Weiterverarbeitung. Insbesonders werden eingehende asynchrone Nachrichten in Puffern zwischengespeichert. Dies bedeutet, dass eingehende Nachrichten nicht sofort verarbeitet werden müssen. Dieser Unter-

schied in Bezug auf andere Statechartdialekte und Stateflow ermöglicht eine einfachere Modellierung der Kommunikation und Koordination in verteilten Systemen.

Die asynchronen Nachrichten zur Kommunikation zwischen Statecharts in CAMeL-View werden zu sogenannten Ports gruppiert, zwischen denen die Kommunikationskanäle spezifiziert werden (vgl. Bild 4). Dies ermöglicht zum einen die Verbesserung der Lesbarkeit der Modelle und zum anderen eine Wiederverwendung von Nachrichten zur Kommunikation mit mehreren Partnern.

Bild 4: Statecharts kommunizieren über Ports

Asynchrone Nachrichten können auch zur Kommunikation innerhalb paralleler Regionen in einem Statechart genutzt werden. Nachrichten können zusätzlich Parameter enthalten. Diese Parameter erlauben den Austausch von Daten. Dies bedeutet, dass für den diskreten Austausch von Daten keine Signalflüsse modelliert werden müssen, sondern stattdessen nur Parameter von Nachrichten gesetzt werden müssen.

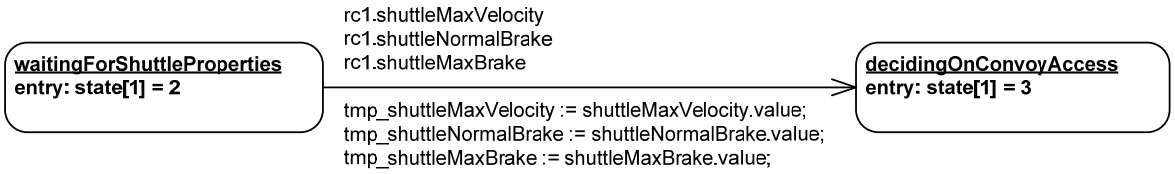

Bild 5: Beispiel für die Behandlung von eingehenden Nachrichten

Bild 5 zeigt eine Umsetzung des Stateflow-Beispiels aus Bild 3 im entwickelten Statechartdialekt in Bezug auf die Nachrichten. Anstelle des Aufrufs von mehreren (manuell implementierten) Funktionen für die Behandlung von Nachrichten wie in Bild 3 kann hier nun direkt formuliert werden, dass nach Eintreffen der verschiedenen Nachrichten über den Port `rc1` die Transition schaltet und die an die Nachricht angehängten Daten in lokalen Variablen des Statecharts gespeichert werden.

Für die Kommunikation innerhalb eines Statecharts werden zusätzlich synchrone Nachrichten unterstützt. Diese Nachrichten ermöglichen parallelen Regionen eines Statecharts sich synchron zu verhalten, indem eine Transition in einer Region genau dann schaltet, wenn auch die Transition einer anderen Region schaltet. Dies ist derzeit in Stateflow nicht spezifizierbar sondern nur über zwei nacheinander schaltende Transitionen unter Benutzung von `in`-Bedingungen, die prüfen, ob ein Statechart in einem bestimmten Zustand ist.

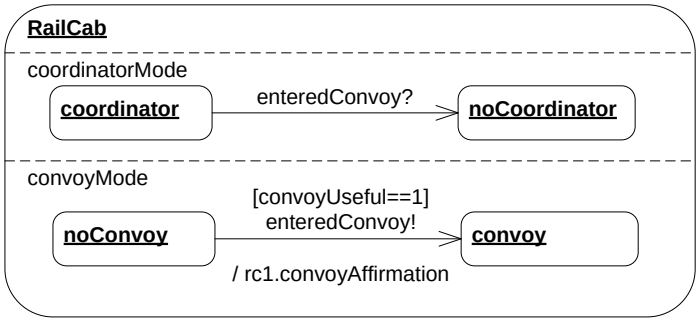

Bild 6: Beispiel für synchrone Nachrichtenaustausch innerhalb eines Statecharts

Bild 6 zeigt ein vereinfachtes Beispiel für die Nutzung von synchronen Nachrichten im RailCab-Beispiel. RailCab Konvois werden durch einen Koordinator organisiert [HTS+08]. Dieser Koordinator entscheidet über den Beitritt von anderen RailCabs zum Konvoi und überträgt für die stabile Regelung relevante Daten an die übrigen RailCabs im Konvoi. Jedes Fahrzeug kann sowohl in einem Konvoi mitfahren, als auch einen Konvoi koordinieren. Das diskrete Verhalten der RailCabs wurde dabei so modelliert, dass ein alleine fahrendes Fahrzeug der Koordinator seines Einzelkonvois ist. Wenn es nun einem anderen Konvoi beitritt, beendet es die Koordination des eigenen Konvois.

Dieses Verhalten ist in Bild 4 zu sehen. Ausgehend von der Situation, dass das Fahrzeug bis jetzt nicht Teil eines größeren Konvois ist (Zustand `noConvoy`) aber die Teilnahme an einem Konvoi sinnvoll ist (Bedingung `convoyUseful == 1`) tritt das RailCab durch das Versenden der Nachricht `convoyAffirmation` einem anderen Konvoi bei, wechselt in den Zustand `convoy` und beendet die Koordination des Einzelkonvois durch den gleichzeitigen Wechsel in den Zustand `noCoordinator`. Dies wird durch die synchrone Nachricht `enteredConvoy`, die zwischen den parallelen Regionen ausgetauscht wird, erreicht. Wenn das RailCab nicht im Zustand `noConvoy` ist, kann die synchrone Nachricht nicht empfangen werden (`enteredConvoy!`), so dass auch die Transition zwischen `coordinator` und `noCoordinator` nicht ausgeführt wird.

3.2.2 Zeit

Um das Verhalten von mechatronischen Systemen korrekt beschreiben zu können, sind verschiedene syntaktische Konstrukte für die Spezifikation von Zeitbedingungen notwendig. Hierbei übernehmen wir den Ansatz der Timed Automata zur Spezifikation der Zeitbedingungen [AD94]. Die Statecharts können hierfür Uhren besitzen. Uhren beschreiben den Verlauf der Zeit während der Ausführung des Statecharts. Sie können beim Schalten einer Transition auf Null gesetzt werden.

Um zeitliches Verhalten zu formulieren, können Zeitbedingungen über die definierten Uhren an Transitionen spezifiziert werden. Eine Transition ist in diesem Fall nur aktiviert, wenn zusätzlich zu den üblichen Bedingungen und erwarteten Nachrichten auch die Bedingungen über die Uhren, sprich Bedingungen über den Zeitverlauf seit dem letzten Null-

setzen der Uhr, wahr sind. Diese Bedingungen können beliebige Uhren des Statecharts referenzieren, so dass sich eine deutlich mächtigere Spezifikation von zeitlichen Bedingungen als mit einfachen `after()`-Konstrukten ergibt.

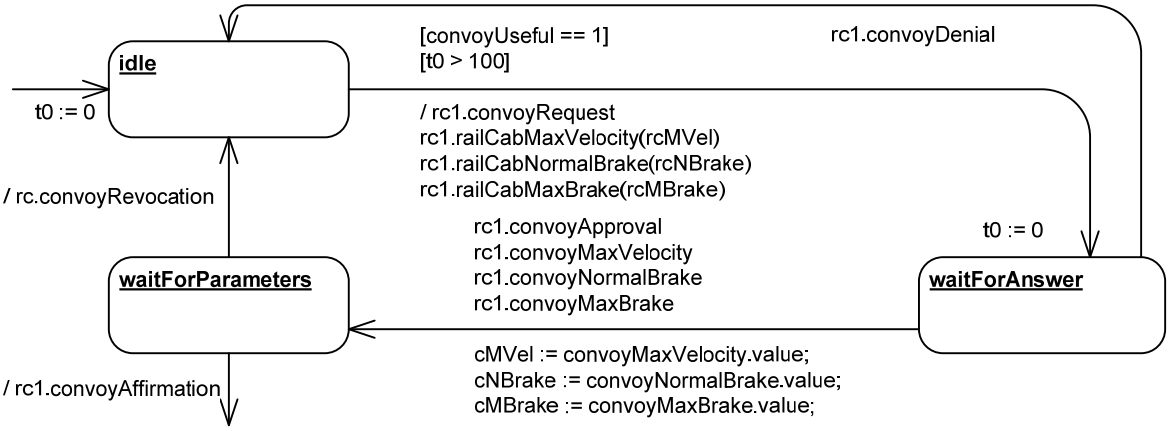

Bild 7: Zustandsbasiertes Verhalten mit Zeitannotationen

Bild 7 zeigt einen Ausschnitt des Verhaltens eines RailCabs in Bezug auf die Teilnahme an einem Konvoi (eine Erweiterung von Bild 5). Das RailCab kommuniziert mit dem Koordinator eines anderen Konvois über den Port `rc1`. Startend im Zustand `idle` fragt das RailCab den Koordinator (unter der Bedingungen, dass der Konvoi für das RailCab sinnvoll ist), ob es am Konvoi teilnehmen darf und sendet zusätzlich seine Fahrtparameter. Der Koordinator antwortet dann mit einer Zusage und den Fahrtparametern des Konvois oder einer Absage. Im ersten Fall entscheidet dann dass RailCab, ob es unter den Bedingungen der Fahrtparametern des Konvois mitfahren möchte oder nicht.

Um zu verhindern, dass nach einer Nichtteilnahme am Konvoi z. B. bei einer Ablehnung des Koordinators das Fahrzeug direkt wieder eine Anfrage stellt, können Uhren und Zeitbedingungen genutzt werden. Beim verschicken der Konvoianfrage wird zusätzlich die Uhr `t0` zurückgesetzt (Annotation `t0:=0`). Dies bedeutet, dass die Uhr zu diesem Zeitpunkt auf den Wert 0 gesetzt wird und die ab diesem Zeitpunkt verstrichene Zeit speichert. An diese Transition wird des Weiteren eine Zeitbedingung `[t0 > 100]` annotiert. Diese beschreibt die zusätzliche Bedingung, dass seit dem letzten Zurücksetzen der Uhr `t0` mindestens 100 Zeiteinheiten verstrichen sein müssen. Insgesamt ist also garantiert, dass maximal alle 100 Zeiteinheiten nach einer Konvoianfrage eine neue Anfrage gestellt wird. Zeiteinheiten können mit den üblichen Zeitmaßen (s, ms, ns, etc.) beschrieben werden.

Zeitbedingungen können auch als Invariante in einem Zustand annotiert werden. Diese Zeitbedingung muss dann gelten, so lange das Statechart in diesem Zustand verweilt.

Zusätzlich wird an eine Transition eine Deadline annotiert, die angibt, wann der Schaltvorgang einer Transition beendet sein muss. Dies steht im Gegensatz zu der oben angesprochene fehlerhaften Nullzeitannahme bei verschiedenen Statechartdialekten. Die Spezifikation von Deadlines ermöglicht damit die Synthese von Quelltext aus den Modellen, der bei der Ausführung auf einer Hardware zeitlich dem Verhalten des Modells entspricht. Dies

ermöglicht, das zeitliche Verhalten des Systems auf einer Ausführungsplattform bei modellbasierten Verifikationsverfahren, wie zum Beispiel Simulationen oder Model Checking, zu überprüfen. Des Weiteren können Deadlines genutzt werden, um automatisch eine korrekte Periode für die Ausführung des Statecharts zu bestimmen, in der alle Deadlines eingehalten werden.

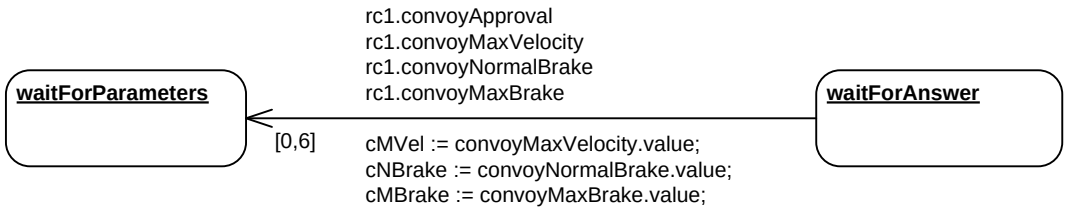

Bild 8: Darstellung einer Deadline [0, 6] für die Ausführung einer Transition

In Bild 8 wird die Transition zwischen den Zuständen `waitForAnswer` und `waitForParameters` aus Bild 6 um die Deadline `[0, 6]` erweitert. Diese Deadline beschreibt, dass nach Prüfung, ob alle Voraussetzungen für das Schalten der Transition (also der Eingang der vier Nachrichten), das Schalten der Transition zwischen 0 und 6 Zeiteinheiten verbrauchen darf.

3.3 Integration mit kontinuierlichem Verhalten

Um Steuerungs- bzw. Regelungsaufgaben in einem hybriden System ausführen zu können, wird ein Statechart in ein solches System eingebettet und muss auf geeignete Art und Weise mit diesem System integriert werden. Hierbei ist einerseits eine Verarbeitung kontinuierlicher Werte des Systems erforderlich, um diskretes Verhalten auf Basis kontinuierlicher Größen wie Sensorwerte zu realisieren. Es besteht hierzu die Möglichkeit, kontinuierliche Größen des übergeordneten Systems über Inputs und Outputs mit dem diskreten Verhalten des Statecharts zu verbinden (vgl. Bild 9).

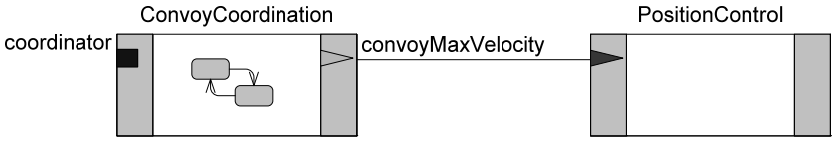

Bild 9: zeigt den Austausch von kontinuierlichen Größen zwischen Statechart und Regler

Die Inputs können über Bedingungen an Transitionen den aktuellen diskreten Systemzustand beeinflussen. Zudem können Outputs innerhalb des Statecharts anhand bestimmter Rechnungsvorschriften mit festen Werten belegt werden, wobei der zuletzt zugewiesene Wert auf dem Output anliegt, bis eine neue Zuweisung erfolgt. Wertberechnungen und -zuweisungen anhand einer Berechnungsvorschrift können einmalig bei Zustandsübergängen erfolgen, jedoch auch kontinuierlich, während das Statechart in einem Zustand verweilt. Ein Statechart ist also in der Lage, in beide Richtungen kontinuierlich mit dem gesteuerten System zu interagieren.

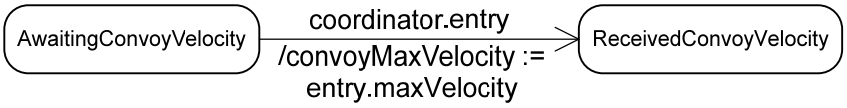

Bild 10: zeigt einen Ausschnitt eines Statecharts, das den Output convoyMaxVelocity in Abhängigkeit des empfangenden Events entry verändert

Ein Beispiel für ein solches Statechart ist in Bild 10 dargestellt. Hierbei wird bei Empfang eines diskreten Events *entry* vom Koordinator zwischen den beiden Zuständen *AwaitingConvoyVelocity* und *ReceivedConvoyVelocity* geschaltet und zusätzlich der Output *convoyMaxVelocity* des Statecharts mit dem Argumentwert des empfangenen Events belegt.

4 Resümee und Ausblick

In diesem Papier haben wir die Modellierungssprache für die Spezifikation hybriden Verhaltens vorgestellt, die im Rahmen eines Transferprojekts entwickelt wird. Die entwickelte Sprache basiert auf den Real-Time Statecharts der Mechatronic UML und erweitert die UML state machines vor allem um Konstrukte zur geeigneten Modellierung von Zeit.

Die Sprache wurde bisher textuell mit einer Grammatik definiert. Derzeit wird an einer geeigneten graphischen Syntax, angelehnt an verwandte Arbeiten, wie in diesem Aufsatz gezeigt gearbeitet. Des Weiteren werden die existierenden Konzepte zur Codegenerierung für Real-Time Statecharts an die neue Sprache angepasst und in die Werkzeugkette zur Codegenerierung in CAMeL-View integriert. Schließlich arbeiten wir an einer Unterstützung der Simulation der Statecharts inklusive einer geeigneten Visualisierung (evtl. auf Basis von [MHS08]) zur Unterstützung von Validierungsaktivitäten.

Literatur

[AD94] ALUR, R.; DILL, D. L.: A Theory of Timed Automata. In: Theoretical Computer Science 126, S. 183–235, 1994

[BBD+02] BEHRMANN, G.; BENGTSSON, J.; DAVID, A.; LARSEN, K. G.; PETTERSSON, P.; YI, W.: Uppaal Implementation Secrets. In: Proc. of 7th International Symposiumon Formal Techniques in Real-Time and Fault Tolerant Systems, 2002

[Bee94] von der BEEK, M.: A comparison of statechart variants. In: Formal Techniques in Real-Time and Fault-Tolerant Systems (Hrsg. L. de Roever and J. Vytopil). Lecture Notes in Computer Science, Band 863, S. 128-148, Springer Verlag, Berlin, 1994

[BGH+07] BURMESTER, S.; GIESE, H.; HENKLER, S.; HIRSCH, M.; TICHY, M.; GAMBUZZA, A.; MÜCH, 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, 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 (Uwe Assmann, Arend Rensink, and Mehmet Aksit (Hrsg.).

Band 3599 Lecture Notes in Computer Science (LNCS), S. 47-61, Springer Verlag, 2005

[BLM02] BIANCO, V.D.; LAVAZZA, L.; MAURI, M.: Formalization of UML Statecharts for Real-Time Software Modeling. In: Proceedings of the 6th International Conference on Integrated Design and Process Technology (IDPT2002), 2002

[DMY02] DAVID, A.; MÖLLER, O.; YI, W.: Formal Verification of UML Statecharts with Real-Time Extensions. In: Kutsche, R.-D.; Weber, H. (Hrsg.): Proceedings of 5th International Conference on Fundamental Approaches to Software Engineering (FASE2002). Grenoble, France, Springer Verlag, Lecture Notes in Computer Science, Band 2306, S. 218–232, Berlin, 2002

[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, 2004

[GTB+03] GIESE, H.; TICHY, M.; BURMESTER, S.; SCHÄFER, W.; FLAKE, S.: Towards the Compositional Verification of Real-Time UML Designs. In: Proc. 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, 2003

[Hah95] HAHN, M.: Object-Oriented Physical Modelling of Mechtronic Systems. Mathematical Modelling of Systems, Vol. 1, No. 4, S. 286-303, Wien, 1995

[Hah99] HAHN, M.: OMD - Ein Objektmodell für den Mechatronikentwurf. Dissertation, Universität-Gesamthochschule Paderborn, 1999

[Har87] HAREL, D.: Statecharts: A Visual Formulation for Complex Systems. Sci. Comput. Program. 8(3), S. 231-274, 1987

[HJ06] HAHN, M.; JÄKER, K.-P.: Domänenübergreifende Modellbildung eines aktiv gefederten Nutzfahrzeugs (CAMeL-View TestRig). In: Isermann, R. (Hrsg.):Fahrdynamik-Regelung. Modellbildung, Fahrerassistenzsysteme, Mechtronik. Vieweg, S. 117-136, Wiesbaden, 2006

[HTS+08] HENKE, C.; TICHY, M.; SCHNEIDER, T.; BÖCKER, J.; SCHÄFER, W.: Organization and Control of Autonomous Railway Convoys. In: Proceedings of the 9th International Symposium on Advanced Vehicle Control, Kobe, Japan, 2008

[HMP92] HENZINGER, T.A.; MANNA, Z.; PNUELI, A.: What Good Are Digital Clocks? In: 9th International Colloquium on Automata, Languages, and Programming (ICALP '92). Springer Verlag, Lecture Notes in Computer Science, Band 623, S. 545–558, Heidelberg, 1992

[Lee09] LEE, E. A.: Computing needs time. *Commun. ACM* 52, 5, S.70-79, 2009

[Lüc90] LÜCKEL, J.: Design Tools for Mechatronic Systems. Vortrag zum Forum'90, Wissenschaft und Technik, Trier, 1990

[LQV01] LAVAZZA, L.; QUARONI, G.; VENTURELLI, M.: Combining UML and formal notations for modelling real-time systems. In: Gruhn, V.(Hrsg.): Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundation of Software Engineering (FSE-9). S. 196–206, Wien, Österreich, ACM Press, 2001

[MHS08] MCINTOSH, P.; HAMILTON, M.; van SCHYNDEL, R.: X3D-UML: 3D UML State Machine Diagrams. In: ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems, 2008

[NH96] NAUMANN, R.; HAHN, M.: Graphical Design Environment for CAMeL-Tools. Mechtronics '96. Universidado do Minho, Guimarães, 1996

[SW07] SCHÄFER, W.; WEHRHEIM, H.: The Challenges of Building Advanced Mechatronic Systems. In: Proceedings of the 29th International Conference on Software Engineering 2007 (Future of Software Engineering Track), Minneapolis, Minnesota, USA. S. 72-84, IEEE Computer Society, 2007

Autoren

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 Arbeitsschwerpunkt ist die modellbasierte Entwicklung verlässlicher Systeme.

Dr. Martin Hirsch, Jahrgang 1978, ist seit Oktober 2009 wissenschaftlicher Mitarbeiter und Leiter der Attract-Arbeitsgruppe von Prof. Jürjens am Fraunhofer-Institut für Software und Systemtechnik ISST in Dortmund. Nach seinem Hochschullabschluss zum Diplom Informatiker im Jahre 2004 an der Universität Paderborn forschte er dort im Sonderforschungsbreich 614 "Selbstoptimierende Systeme des Maschinenbaus". Er promovierte 2008 an der Universität Paderborn im Institut für Informatik zum Thema "Modell-basierte Verifikation von vernetzten mechatronischen Systemen". Sein Arbeitsschwerpunkt ist die modellbasierte Entwicklung von Software-Intensiven Systemen.

Dipl.-Inform. (FH) Christopher Brink, Jahrgang 1984, ist Doktorand im Fachbereich Softwaretechnik bei Prof. Dr. Wilhelm Schäfer am Institut für Informatik der Universität Paderborn. Er hat im August 2008 sein Studium der Informatik als Dipl.-Inform. (FH) abgeschlossen, erlangte 2009 seine Promotionsberechtigung und forscht seitdem u.a. im Bereich der Softwareproduktlinien.

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

Christopher Gerking, Jahrgang 1985, ist seit 2005 Student der Informatik an der Universität Paderborn. Hier beschäftigt er sich mit der modellbasierten Softwareentwicklung mechatronischer Systeme.

Dr.-Ing. Martin Hahn Jahrgang 1965, ist seit 1999 Geschäftsführer der iXtronics GmbH. Nach seinem Hochschulabschluss im Maschinenbau im Jahre 1991 forschte er am Mechatronik Laboratorium Paderborn (Prof. Dr.-Ing. J. Lückel) im Bereich der objektorientierten Modellbildung mechatronischer Systeme. Er promovierte 1999 an der Universität Paderborn im Maschinenbau zum Thema "OMD - Ein Objektmodell für den Mechatronikentwurf". Sein Arbeitsschwerpunkt ist die modellgetriebene Entwicklung mechatronischer Systeme in Projekten der Automobil-, Luftfahrt- und Feinwerktechnik.