Simulation von hybridem Verhalten in CAMeL-View[1]

Dipl.-Inform (FH) Christopher Brink, Dipl.-Inform. Joel Greenyer,
Prof. Dr. Wilhelm Schäfer
Fachgebiet Softwaretechnik, Heinz Nixdorf Institut, Universität Paderborn
Warburger Straße 100, 33098 Paderborn
Tel. +49 (0)5251/60-3307, Fax. +49 (0)5251/60-3530
E-Mail: [Christopher.Brink|jgreen|wilhelm]@upb.de

Dr. Martin Hahn
iXtronics GmbH
Technologiepark 9, 33100 Paderborn
Tel. +49 (0)5251 417 85-0, Fax. +49 (0)5251 417 85-29
martin.hahn@ixtronics.com

Prof. Dr. Matthias Tichy
Lehrstuhl für Organic Computing, Institut für Informatik, Universität Augsburg,
Universitätsstr. 6a, 86159 Augsburg
Tel. +49 (0)821/598-2229, Fax. +49 (0)821/598-2175
E-Mail: tichy@informatik.uni-augsburg.de

Zusammenfassung

Die Innovation in modernen technischen Systemen ist vor allem getrieben durch die eingebettete Software. Insbesondere die auf Software basierende Kommunikation und Koordination zwischen mehreren Systemen ermöglicht es neuartige technische Produkte zu entwickeln, die sich flexibel an geänderte Situationen anpassen können. Diskretes Verhalten und kontinuierliche Regelungsfunktionen beeinflussen sich hierbei massiv und müssen daher geeignet integriert modelliert werden. Das Werkzeug CAMeL-View ermöglicht die modellbasierte Entwicklung dieser hybriden Systeme. In diesem Beitrag wird erläutert, wie aus hybriden Modellen automatisch Quelltext erzeugt wird und wie die komplexen Zusammenhänge zwischen diskreten und kontinuierlichen Verhaltensaspekten bei der Simulationen dieser Modelle geeignet visualisiert werden können.

Schlüsselwörter

Modellbasierte Entwicklung, Visualisierung, mechatronische Systeme, hybrides Verhalten, Simulation

---

[1] Diese Arbeit ist im „Transferprojekt T2 - Hybride Modellierung“ des Sonderforschungsbereichs 614 „Selbstoptimierender Systeme des Maschinenbaus“, Universität Paderborn, entstanden und wurde auf seine Veranlassung unter Verwendung der ihm von der Deutschen Forschungsgemeinschaft zur Verfügung gestellten Mittel veröffentlicht.

1 Einführung

Die fortschrittlichen Funktionen in heutigen mechatronischen Systemen werden häufig durch Softwarekomponenten und deren komplexer Interaktion realisiert. Zusätzlich zu den regelungstechnischen Bestandteilen der Software in einem mechatronischen System steigt deshalb der Anteil der diskreten Software, welche komplexes zustandsbasiertes Verhalten sowie Kommunikation und Koordination der einzelnen Systemkomponenten realisiert. Diskretes Zustandsverhalten und kontinuierliche Regelungsfunktionen beeinflussen sich massiv, wodurch die Komplexität der Software stark zunimmt. Um diese komplexen Vorgänge beschreiben zu können, ist es daher notwendig, dieses hybride (diskret / kontinuierliches) Verhalten geeignet zu modellieren. Des Weiteren ist für die Validierung eine Simulation mit geeigneter Visualisierung notwendig.

Im Rahmen des von der DFG geförderten Transferprojekts „Hybride Modellierung“ wurde das Modellierungswerkzeug CAMeL-View der Firma iXtronics um die Real-Time Statecharts der Modellierungssprache Mechatronic UML [BGT05] erweitert. Durch diese Erweiterung ist es möglich, auch hybrides (kontinuierliches und diskretes) Verhalten in dem Werkzeug zu modellieren. Hierzu wurden die Real-Time Statecharts in Form eines neuen Blocktyps in das Werkzeug integriert [THB+10]. In dem vorliegenden Beitrag wird nun zum einen erläutert, wie der Codegenerator von CAMeL-View um die Unterstützung von Real-Time Statecharts erweitert wurde. Zum anderen wird beschrieben, welche neuen Konzepte zur geeigneten Visualisierung des diskreten Verhaltens entwickelt wurden.

Im Vergleich zu den im Folgenden dargestellten Konzepten unterstützen aktuelle Werkzeuge zur modellbasierten Entwicklung hybrider Systeme, wie z.B. Matlab[2] mit den Toolboxen Simulink und Stateflow, die Entwickler bei der Untersuchung von Simulationen und Tests komplexer hybrider Systeme derzeit nur eingeschränkt. So ist es zwar in Matlab möglich, sich einen Plot der kontinuierlichen Ausgaben anzuzeigen und dort auch diskrete Informationen, z.B. ob ein Zustand aktiv ist, darzustellen. Die Darstellung ist allerdings darauf beschränkt, diese diskreten Informationen als 0/1-Werte im Plot darzustellen, sodass die Übersichtlichkeit bei größeren Modellen mit einer Vielzahl an Zuständen und komplexem Nachrichtenaustausch nicht mehr gegeben ist. Auch LabView[3] oder Dymola[4] verwenden für eine Visualisierung Plot, das Tool Rhapsody[5] beschränkt sich bei der Simulation von Statecharts auf die Darstellung von Sequenzdiagrammen. Zusammenhänge zwischen kontinuierlichem und diskretem Verhalten bei der Simulation stellt allerdings kein Tool derzeit dar.

[2] www.mathworks.de

[3] www.ni.com/labview/d/

[4] www.3ds.com/de/products/catia/portfolio/dymola/overview/

[5] www-01.ibm.com/software/rational/products/rhapsody/swarchitect/

Nach einer Erläuterung eines durchlaufenden Beispiels und dessen Modellierung werden die Erweiterung der Codegenerierung sowie die Konzepte für die Visualisierung vorgestellt. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick.

2 Beispiel

Als Beispiel dient das Modell einer Mondlandefähre. In dieser Landefähre kann ein Pilot zunächst den Schub der Bremsraketen steuern um damit die Höhe der Landefähre zu beeinflussen. Das physikalische Verhalten der Landefähre, welches aus dem Schub des Triebwerks, der Anziehungskraft des Mondes und dem Gewicht der Landefähre ergibt, wird dabei in einem kontinuierlichen, mechanischen Mehrkörpersystemmodell beschrieben [Hah10]. Das Modell enthält zudem einen einfachen Regler, welcher den Schub der Landefähre regelt, um eine bestimmte Sollhöhe zu erreichen. Dieses Modell wird nun um zwei diskrete/kontinuierliche Aspekte erweitert. Zunächst soll es dem Piloten möglich sein, von der manuellen Schubkontrolle auf einen Automatik umzuschalten, bei der die Landefähre die aktuelle Höhe beibehält. Die zweite Erweiterung des Modells soll es dem Piloten erlauben, die Verbindung zu einer Bodenstation herzustellen. Soweit diese verfügbar ist, leitet sie eine dreistufige Landesequenz ein (s. Bild 1).

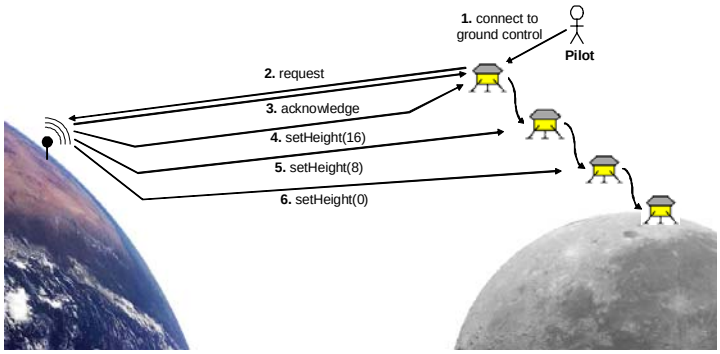

Bild 1: Kommunikation bei der ferngesteuerten Landung der Mondlandefähre

3 Modellierung

Bild 2 zeigt ein Blockdiagramm aus CAMeL-View, welches das physikalische Verhalten der Landefähre und die Regelung für das Erreichen einer Sollhöhe beschreibt. Es enthält zudem Real-Time-Statechart-Blöcke für die Realisierung des o.g. Kommunikationsverhaltens. Die Regelung der Sollhöhe wird durch die vier Blöcke oben rechts in Bild 2 beschrieben. Der Block **height** extrahiert die Höhe aus den Koordinaten der Landefähre im physikalischen Modell, **sub** berechnet die Differenz zwischen der aktuellen Höhe der Fähre und einer Zielhöhe („target height“). Auf Basis dieser Differenzwerte stellt der **controller** (PD-Regler) den Schub ein, welcher nachfolgend durch den **limiter** begrenzt wird. Am Block **thrust** wird der Schub eingestellt. Dieser Wert wird dann in den Teil des Modells weitergegeben, welcher das physikalische Verhalten der Landefähre beschreibt. Auf diesen Teil wird hier aber nicht weiter eingegangen.

Die Blockdiagramme in CAMeL-View wurden um Real-Time-Statechart-Blöcke erweitert. Ein Real-Time-Statechart-Block enthält ein Real-Time Statechart und kann über Eingänge und Ausgänge mit anderen Böcken kontinuierliche Werte austauschen. Des Weiteren können Real-Time-Statechart-Blöcke über Konnektoren und Ports mit anderen Real-Time-Statechart-Blöcken verbunden werden um Nachrichten auszutauschen. Bild 2 zeigt die Real-Time-Statechart-Blöcke **landerCtrl** und **groundCtrl**, welche durch eine Nachrichtenverbindung verbunden sind.

Der Real-Time-Statechart-Block **landerCtrl** besitzt einen Eingang „currentHeight“, über den kontinuierlich die aktuelle Höhe der Fähre vom Block **height** mitgeteilt wird. Des Weiteren besitzt der Block zwei Ausgänge, „isOnManual“ und „targetHeight“, welche mit Eingängen „userControl“ des Blocks **thrust**, bzw. „targetHeight“ des Blocks **sub** verbunden ist. Über diese Ausgänge stellt das Real-Time-Statechart ein, ob die Steuerung manuell erfolgen oder ob die Höhe automatisch geregelt werden soll, bzw. welche Zielhöhe die Fähre zu erreichen hat. Über den Parameter „manual“ und „connectToGC“ kann das Verhalten des Real-Time Statecharts beeinflusst werden: Ist der Wert von „manual“ null, bedeutet dies, dass der Pilot auf die Höhenautomatik geschaltet hat. Ist der Wert von „connectToGC“ größer null, bedeutet dies, dass der Pilot die durch die Bodenstation ferngesteuerte Landesequenz initiieren möchte. Der Real-Time-Statechart-Block **groundCtrl** besitzt den Parameter „available“. Ist der Wert für diesen Parameter größer Null, ist die Bodenstation erreichbar.

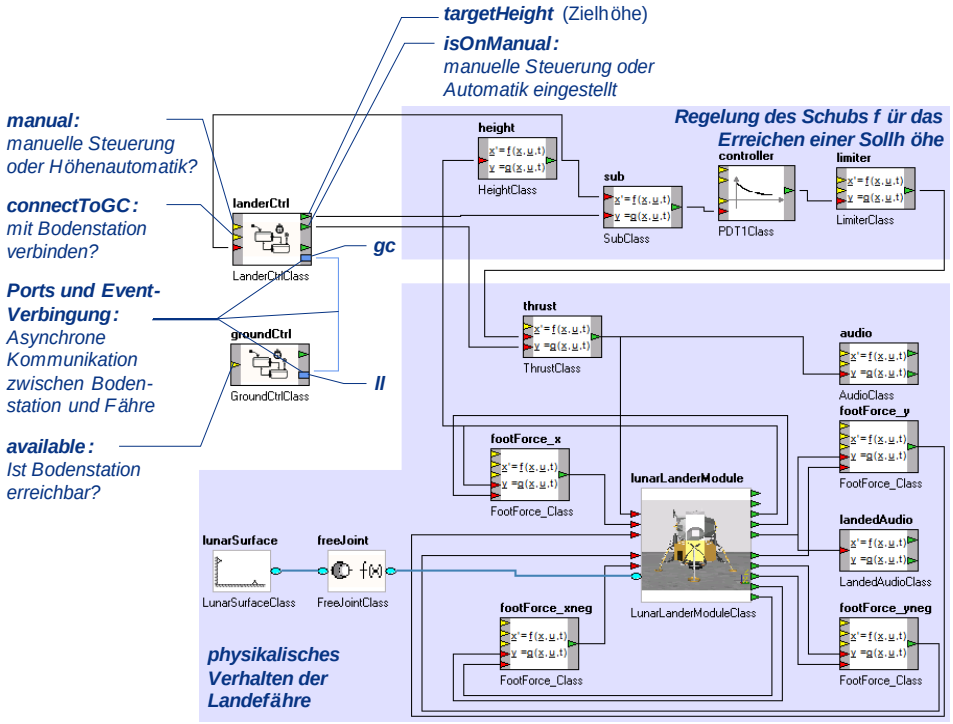

Bild 2: Blockdiagramm der Mondlandefähre

Real-Time Statecharts wurden in CAMeL-View mit allen in Mechatronic UML definierten Sprachkonstrukten integriert. Es ist also möglich Zustandsmaschinen mit hierar-

chischen Zuständen und parallelen Regionen zu modellieren. Transitionen in verschiedenen parallelen Regionen eines Zustands können über synchrone Nachrichten synchronisiert werden, und verschiedene Real-Time Statecharts können über asynchrone, parametrisierte Nachrichten kommunizieren. Zeit wird in den Real-Time Statecharts mit Hilfe von Uhren repräsentiert. Uhren sind kontinuierliche Variablen, welche in Invarianten von Zuständen oder Bedingungen von Transitionen verwendet werden können. Werte von Uhren steigen kontinuierlich, linear und synchron mit der Zeit; eine Uhr kann durch ein *Reset* an einer Transition wieder auf null gesetzt werden. Nichtdeterminismus, welcher in anderen Statecharts auftreten kann, wird in Real-Time Statecharts durch Prioritäten an Transitionen und parallelen Regionen verhindert. Real-Time Statecharts sind derart in CAMeL-View integriert, dass die kontinuierlichen Werte von Eingängen und Parametern in Bedingungen und Anweisungen gelesen werden können. In Anweisungen kann der Wert eines kontinuierlichen Ausgangs geschrieben werden.

Als Beispiel zeigen Bild 3 und Bild 4 die Real-Time Statecharts der Landefähre und der Bodenstation. Sie bestehen jeweils auf oberster Ebene nur aus einem Zustand mit zwei parallelen Regionen. In den oberen Regionen ist Kommunikationsverhalten zwischen der Landefähre und der Bodenstation modelliert. Ist der Parameter „connectToGC“ größer null, wechselt die Landefähre in den Zustand *connectionRequested*. Beim Zustandswechsel wird die Nachricht „request“ über den Port „gc“ an das Real-Time-Statechart der Bodenstation gesendet. Das Real-Time-Statechart der Bodenstation (s. Bild 4) wechselt beim Empfangen dieser Nachricht ebenfalls in den Zustand *connectionRequested*. Ist die Bodenstation erreichbar („available“ größer null), wechselt das Real-Time-Statechart der Bodenstation in den Zustand *connected* und sendet dabei die Nachricht „acknowledge“ über den Port „ll“ an die Landefähre. Als Resultat sind dann beide Statecharts im Zustand *connected*.

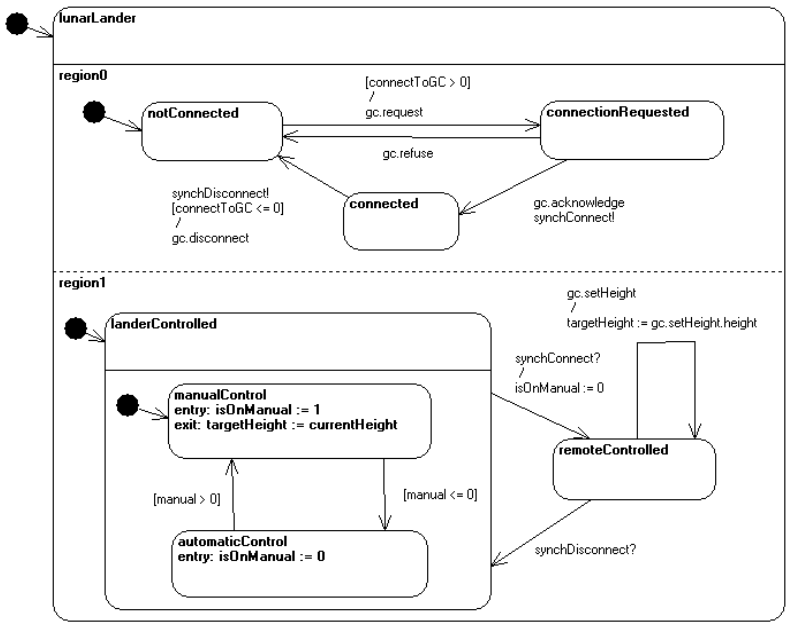

Bild 3: Das Real-Time-Statechart der Mondlandefähre

Die Transitionen in parallelen Regionen eines Statecharts können über synchrone Nachrichten synchronisiert werden. Im Statechart der Bodenstation gibt es zum Beispiel die synchronen Nachrichten „synchConnect“ und „synchDisconnect“. Die sendende Transition, z.B. von *connectionRequested* nach *connected* kann nur dann schalten, wenn gleichzeitig eine empfangende Transition aktiv ist. In diesem Fall schalten dann beide Transitionen gleichzeitig. Beim Wechsel von *connectionRequested* in *connected* wechselt das Statechart der Bodenstation also gleichzeitig vom Zustand *idle* in den hierarchischen Zustand *controlLander*. Ähnlich wechselt die Landefähre gleichzeitig in den Zustand *remoteControlled*. Dabei wird der Wert des Ausgangs „isOnManual“ auf null gesetzt. Dies führt dazu, dass der Schub (am Block **thrust**, s. Bild 2) nun durch die Regler eingestellt wird. Die Sollhöhe für den Regler wird nun über den Ausgang „targetHeight“ der **landerCtrl** vorgegeben. Im Statechart der Mondlandefähre wird der Wert für diesen Ausgang gesetzt, wenn im Zustand *remoteControlled* parametrisierte Nachrichten „setHeight“ empfangen werden. Diese Nachrichten werden im Zustand *controlLander* der Bodenstation in einem Abstand von 20 Zeiteinheiten gesendet. Bei der Transition von *initLanding* nach *heightA* wird die Uhr „c“ auf null gesetzt. Für die nachfolgende Transitionen ist als Bedingung eine untere zeitliche Schranke definiert.

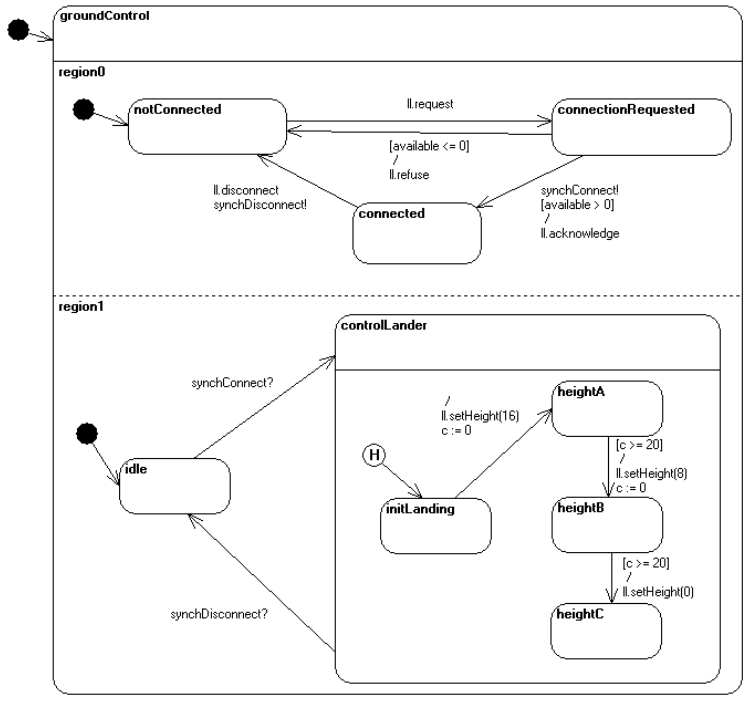

Bild 4: Das Real-Time-Statechart der Bodenstation

4 Codegenerierung

CAMeL-View stellt für die Modellbildung mechatronischer Systeme die objektorientierte Modellbeschreibungssprache Objective-DSS bereit [Hah99]. In Objective-DSS können Modellobjekte textuell beschrieben und in modular-hierarchischer Form zu mechatronischen Baugruppen und Gesamtsystemen verknüpft werden. Objective-DSS de-

finiert zudem spezielle Klassen für Modellobjekte der Domänen Mechanik, Hydraulik, Elektrotechnik, Regelungstechnik und Informationsverarbeitung, wie oben in Bild 5 illustriert. Systemmodelle, welche aus diesen Modellobjekten aufgebaut sind, können in einem mehrstufigen Prozess in C-Code für eine Laufzeitumgebung zur Simulation oder für die Ausführung auf bestimmten Plattformen übersetzt werden (s. Bild 5 unten). In diesem Prozess werden Systemmodelle in Objective-DSS zunächst in ein mathematisches Modell in der Sprache Objective-DSL überführt. Daraus wird dann ein plattformunabhängiges Laufzeitmodell generiert, woraus dann oben genannter C-Code erzeugt werden kann.

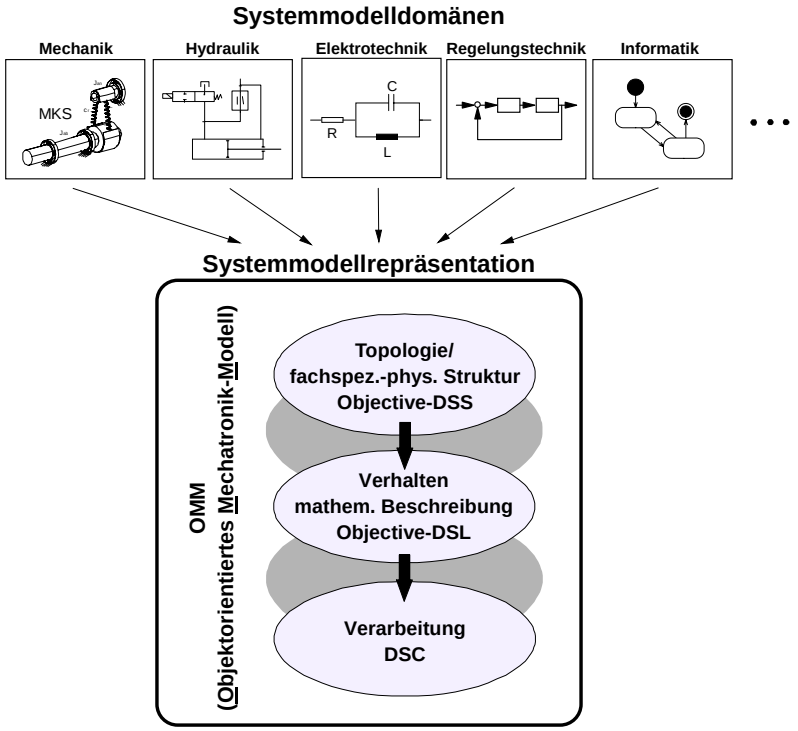

Bild 5: Objektorientiertes Mechatronik-Modell von CAMeL-View

Für die Integration von Real-Time Statecharts in CAMeL-View wurde eine neue Klasse „RealtimeStateChartOdss“ in Objective-DSS definiert. Diese Definition beschreibt, wie Real-Time Statecharts textbasiert beschrieben werden können, es wurde jedoch auch die in Bild 3 und Bild 4 gezeigte grafische Syntax definiert. Um plattformspezifischen C-Code oder C-Code für die Simulationsumgebung aus den Real-Time Statecharts zu generieren, hat es sich als zweckmäßig erwiesen, sich in den bestehenden Codegenerierungsprozess von CAMeL-View einzugliedern. Es wurde ein Generator entwickelt, um Real-Time Statecharts in Modellobjekte der Sprache Objective-DSL, genauer Modellobjekte der Klasse „StateSpaceOdss“ zu übersetzen. Modellobjekte dieser Klasse stellten sich als ein geeignetes Element für die Beschreibung von hybridem, zeitbehafteten Verhalten heraus.

StateSpaceOdss definiert die Beschreibung von Funktionen durch Variablen und Gleichungen. Spezielle Variablen sind Eingänge, Ausgänge, Parameter, Hilfsvariablen, Zeit und Zustandsvariablen. Gleichungen können Differentialgleichungen oder Zuweisungen

für Hilfsvariablen oder Ausgänge sein. Schleifen können nicht in StateSpaceOdss definiert werden, die Gleichungen werden also rein sequentiell ausgeführt.

Die Abbildung von Real-Time Statecharts auf StateSpaceOdss funktioniert wie folgt. Für jedes Real-Time Statechart wird eine Hilfsvariable erzeugt, welche den aktuellen Zustand eines Real-Time Statecharts speichert. Eine Transition wird durch bedingte Ausdrücke beschrieben, welche überprüfen, ob die Voraussetzungen für das Schalten einer Transition erfüllt sind, z.B. ob der aktuelle Zustand der Quellzustand der Transition ist, oder ob die Bedingung einer Transition erfüllt ist. Quelltext 1 zeigt das Beispiel einer Transition, welche in RealtimeStateChartOdss beschrieben ist. (Hier ist es die Transition für den Übergang vom Zustand *manualControl* in den Zustand *automaticControl* aus dem Mondlande-Beispiel, s. Bild 3.)

```
1       source: manualControl target: automaticControl
2           guard: manual <= 0;
```

Quelltext 1: Beispieltransition in RealtimeStateChartOdss

Zunächst werden der Quell- und der Zielzustand angegeben (Zeile 1), bevor Bedingungen für die Transition definiert werden. In diesem Fall wird nur festgelegt, dass *manual* einen Wert kleiner gleich 0 annehmen muss (Zeile 2), damit die Transition schalten kann.

Quelltext 2 zeigt die gleiche Transition nach der automatischen Abbildung auf StateSpaceOdss. Hierbei ist zu erkennen, dass die Transition bei erfüllter Bedingung (Zeile 3) schaltet. Hierzu wird der Zustand *manualControl* verlassen (Zeile 6), bevor der neue Zustand *automaticControl* betreten wird (Zeile 8-9). Texte in Hochkommata sind Kommentare.

```
1       " transition<15> from <manualControl> to <automaticControl> with priorities(region
2       ,transition) <(1,0)>"
3       manual <= 0 & curState_[4] = 2 & fired_[4] = 0 & errorCode_[1] = 0 isTrue:
4           [
5           " exit state <manualControl> "
6           manualControlled_ := 0;
7           " enter state <automaticControl> "
8           fired_[4] := 1;
9           curState_[4] := 1;
10          targetHeight_ := currentHeight
11      ];
```

Quelltext 2: Beispieltransition in StateSpaceOdss

Durch die Abbildung auf StateSpaceOdss ist es nun möglich, die bestehende und bewährte Codegenerierung von CAMeL-View zu nutzen. Zudem ist es auch möglich, die vorhandene Simulationsumgebung zu nutzen. Die Konzepte der Codegenerierung wurden in einer Diplomarbeit ausgearbeitet, implementiert und dokumentiert [Gew10].

5 Simulation

Auf Basis des generierten Codes können die Modelle nun simuliert werden. Hierfür stellt CAMeL-View eine umfangreiche Simulationsumgebung zur Verfügung. Diese nutzt visuelle Elemente, sogenannte *Widgets*, um spezielle Informationen aus den Modellen darzustellen. Bild 6 zeigt die Anordnung verschiedener Widgets für die Visualisierung der Simulation der Mondlandefähre. Oben rechts findet sich ein *Time Plot*, der für die Darstellung der Werte von kontinuierlichen Variablen, wie z.B. Eingängen, Ausgängen oder Parametern, verwendet wird. Ein Time Plot besteht aus einem Koordinatensystem, bei dem die X-Achse die Simulationszeit und die Y-Achse die aktuellen Werte der Simulationsdaten repräsentiert. Neben den Widgets zur Darstellung von Simulationsdaten bietet die Simulationsoberfläche auch Eingabewidgets, die eine aktive Beeinflussung der Simulation ermöglichen, wie z.B. Schieberegler oder Eingabefelder.

Um die diskreten Anteile des Verhaltens geeignet zu visualisieren, war es notwendig die Möglichkeiten der Simulationsoberfläche zu erweitern. Diese Erweiterungen wurden in einer Diplomarbeit ausgearbeitet und implementiert [Ric10]. Dabei standen drei Anforderungen im Vordergrund. Erstens sollte es möglich sein, die *Wechselwirkung zwischen kontinuierlichem und diskretem Verhalten* geeignet darzustellen. Zweitens war es wichtig den *Nachrichtenaustausch zwischen Real-Time Statecharts* sowie die stattfindenden *Zustandswechsel bei der Kommunikation* zu visualisieren. Da die Real-Time Statecharts oft komplexe Abläufe beschreiben, war es drittens notwendig, die *Informationen der Visualisierung mit der grafischen Darstellung der Real-Time Statecharts zu koppeln.*

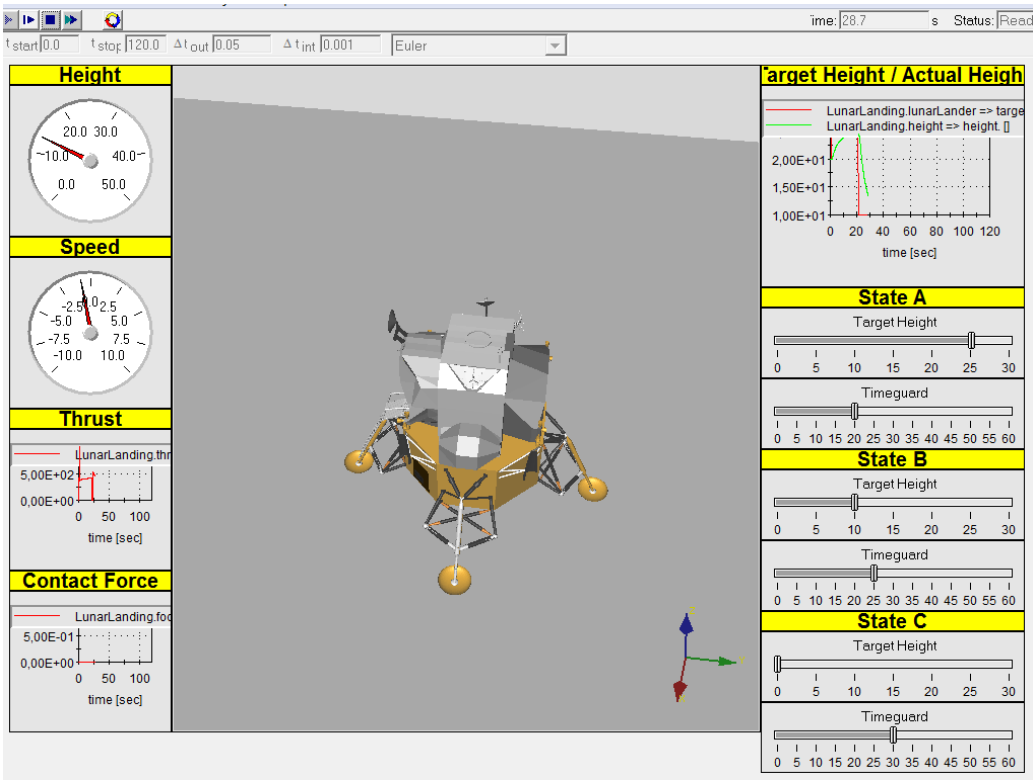

Bild 6: Simulationsoberfläche in CAMeL-View

Visualisierung der Wechselwirkung von kontinuierlichem und diskretem Verhalten
In mechatronischen Systemen bestehen typischerweise starke Zusammenhänge zwischen diskreten und kontinuierlichen Bestandteilen des Verhaltens. Werden diese Zusammenhänge bei der Entwicklung nicht deutlich, kann dies leicht zu Fehlern in der Software führen. Deshalb wurde im Transferprojekt der zuvor beschriebene Time Plot erweitert, sodass sowohl die Zustände, Zustandswechsel, Transitionen und Nachrichten eines Real-Time Statecharts als auch der kontinuierliche Datenfluss visualisiert werden können. Hierdurch kann bei einer Simulation beispielsweise festgestellt werden, ob ein Zustandswechsel oder ein Nachrichtenaustausch für eine Verhaltensänderung in kontinuierlichen Verhalten verantwortlich ist, oder ob dies durch die kontinuierlichen Anteile bedingt ist. Bild 7 zeigt einen erweiterten Time Plot des Mondlande-Beispiels, der die Kurve einer des kontinuierlichen Ausgangs *height* des Blocks **height** (s. Bild 2) anzeigt und zusätzlich die diskreten Anteile darstellt. Unterhalb der Zeitachse sind die einzelnen, zum jeweiligen Zeitpunkt aktiven Zustände aufgeführt, wobei auch die Hierarchie der Zustände visualisiert wird. (Die Zustände *HeightA*, *HeightB* und *HeightC* gehören zum Oberzustand *controlLander*.)

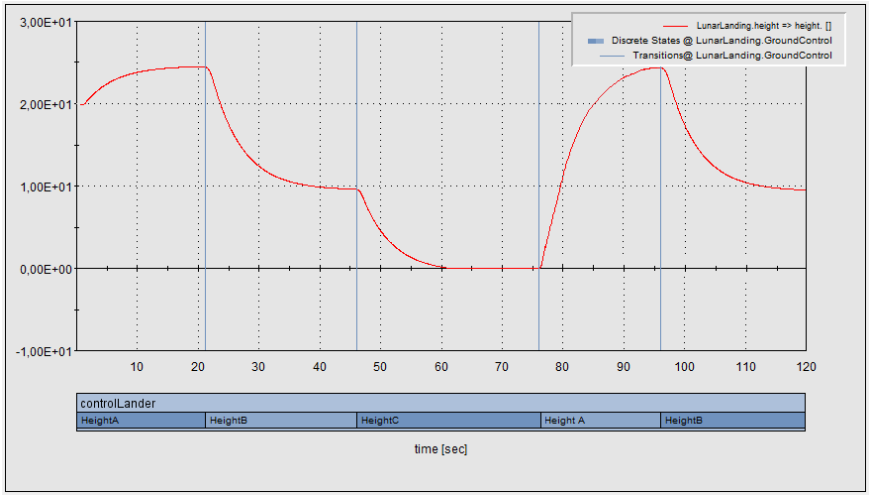

Bild 7: Visualisierung von hybridem Verhalten

Die Platzierung der Zustände wurde unterhalb des Koordinatensystems gewählt, da diese keine eigentlichen Werte annehmen, sondern lediglich zwischen aktiv und inaktiv wechseln. Schaltende Transitionen werden durch eine vertikale Linie innerhalb des Koordinatensystems dargestellt. Dies erleichtert die Analyse des Verhaltens, da Transitionen verschiedene Ereignisse auslösen können, die das Verhalten beeinflussen. Hierzu zählen unter anderem das Versenden von Nachrichten, Wertzuweisungen von kontinuierlichen Variablen und das Zurücksetzen von Uhren. Bei schaltenden Transitionen sind darüber hinaus Sonderfälle zu berücksichtigen, die einer anderen Darstellung bedürfen. Aktive Transitionen mit einer zeitlichen Bedingung schalten ggf. nicht sofort und werden dann als Fläche dargestellt. Wenn zwei synchronisierte Transitionen in parallelen Regionen gleichzeitig schalten, werden diese Transitionen durch eine zweifarbige Linie dargestellt, bzw. durch eine Fläche mit zwei horizontal getrennten Farbbereichen.

Visualisierung von diskretem Verhalten

Für die Validierung der rein diskreten Anteile des Systems ist es besonders wichtig, das Interaktionsverhalten zwischen den Statecharts zu betrachten. Um dies zu ermöglichen, wurde ein Ansatz erweitert, der das Verhalten mithilfe von Sequenzdiagrammen darstellt [TK03]. Dieser wurde angepasst und in ein neues *Widget für Sequenzdiagramme* integriert, welches eine beliebige Anzahl an im System vorhandenen Real-Time Statecharts anzeigen kann. Jedes Statechart wird dabei durch eine Lebenslinie dargestellt, welche die jeweils aktiven Zustände anzeigt. Die zwischen den Statecharts ausgetauschten Nachrichten werden durch Pfeile repräsentiert, an denen der jeweilige Name der Nachricht steht. Anders als beim zuvor vorgestellten Time Plot wird die Zeit nicht linear dargestellt. Es ist jedoch möglich, zeitliche Angaben in dieser Darstellung an ausgewählten Positionen zu erhalten. Diese Einschränkung wurde getroffen, da hierdurch eine kompaktere Darstellung erreicht wird, welche die Analysemöglichkeiten nicht beeinflusst. Bild 8 zeigt das beschriebene Widget für das Mondlande-Beispiel. Dargestellt werden die beiden Statecharts *GroundControl* und *LunarLander* sowie die beiden ausgetauschten Nachrichten *request* und *acknowledge*. Zusätzlich wird die Simulationszeit bei jedem Zustandswechsel in Form einer Linie angezeigt.

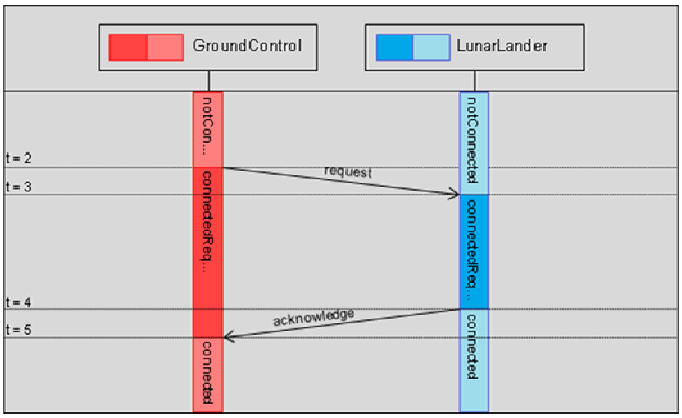

Bild 8: Visualisierung von diskretem Interaktionsverhalten

Zusätzlich zu den bisher vorgestellten Konzepten wurde auch ein Konzept für die Visualisierung des Schaltverhaltens in der Syntax der Real-Time Statecharts erarbeitet. Hierbei werden aktuelle Zustände eines Real-Time Statecharts sowie aktive Transitionen eingefärbt. Diese Konzepte wurden allerdings bisher nicht im Werkzeug umgesetzt.

6 Resümee und Ausblick

Die fortschrittlichen Funktionen in mechatronischen Systemen werden heute maßgeblich durch Software realisiert, die sich typischerweise durch eine enge Kopplung von kontinuierlichen Reglern und diskretem Kommunikationsverhalten auszeichnet. Heutige Werkzeuge unterstützen die Entwicklung dieser Art von Software nur unzureichend. Insbesondere die Simulation dieser hybriden Softwaremodelle wird von heutigen Werkzeugen nicht geeignet unterstützt. Durch die Integration von Real-Time Statecharts in

CAMeL-View ist ein umfangreiches Werkzeug für den modellbasierten Reglerentwurf um ein mächtiges und komfortables Beschreibungselement für den Entwurf von zeitbehafteten Kommunikationsverhalten erweitert worden. Dieses Papier präsentierte nun die Erweiterung der Simulationsumgebung von CAMeL-View. Diese Erweiterung umfasst eine Codegenerierung, welche die Simulation umfangreicher hybrider Modelle ermöglicht. Für die geeignete Darstellung der komplexen Zusammenhänge zwischen diskretem und kontinuierlichem Verhalten wurden verschiedene Konzepte der Visualisierung präsentiert. Die Konzepte wurden prototypisch in CAMeL-View implementiert, sind jedoch auch auf andere Werkzeuge übertragbar. In Zukunft sollen weitere Analysemethoden in die Entwicklungsumgebung integriert werden, vor allem die formale Verifikation, um die Einhaltung von kritischen Anforderungen durch die Software automatisiert beweisen zu können.

Literatur

[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), Seiten 47-61, Springer Verlag, August 2005.

[Gew10] GEWERING, T.: Generierung von Quelltext für Real-Time Statecharts in CAMeL-View. Masterarbeit, Universität Paderborn, September 2010.

[Hah10] HAHN, M.: Vorlesung "Simulation und Numerik von Mehrkörpersystemen", SS 2010, IDS, Leibniz Universität Hannover.

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

[Ric10] RICHTERMEIER, M.: Informationsvisualisierung bei Simulation von Realtime Statecharts in CAMeL-View. Masterarbeit, Universität Paderborn, September 2010.

[THB+10] TICHY, M.; HIRSCH, M.; BRINK, C.; GERKING, C.; HAHN, M.; SCHÄFER, W.: Integration hybrider Modellierungstechniken in CAMeL-View. In 7. Paderborner Workshop Entwurf mechatronischer Systeme, HNI-Verlagsschriftenreihe, vol. 272, Seiten 235-251. März 2010.

[TK03] TICHY, M.; KUDAK, M.: Visualization of the execution of Real-Time Statecharts. In: Proc. of the first International Fujaba Days 2003, Kassel, Germany, 2003.

Autoren

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 im Bereich der Softwareproduktlinien und feature-orientierten Modellierung.

Dipl.-Inform. Joel Greenyer, Jahrgang 1980, ist Doktorand im Fachbereich Softwaretechnik bei Prof. Dr. Wilhelm Schäfer am Institut für Informatik der Universität Paderborn. Nach Abschluss seines Studiums im Jahr 2006 wurde er in das Doktorandenpro-

gramm der International Graduate School Dynamic Intelligent Systems aufgenommen. In seiner Dissertation befasst er sich mit Methoden zu Vermeidung von Inkonsistenzen in szenariobasierten Verhaltensmodellen mechatronischer System und beschäftigt sich zudem mit Techniken zu Transformation und Synchronisation von Softwaremodellen.

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.

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.

Prof. Dr. Matthias Tichy, Jahrgang 1978, vertritt derzeit den Lehrstuhl für Organic Computing an der Universität Augsburg. 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“ und arbeitete danach als Senior Researcher im Software Quality Lab (s-lab) der Universität Paderborn.