Einsatz des FMI/FMU-Standards zur frühzeitigen Simulation von Software- und Hardwaremodellen komplexer mechatronischer Systeme

Uwe Pohlmann[1], Robert Wagner[2]

[1]Heinz Nixdorf Institut, Fachgebiet Softwaretechnik, Fürstenallee 11, D-33102 Paderborn, upohl@upb.de
[2]Solunar GmbH, Softwareentwicklung, Verler Str. 302, D-33334 Gütersloh, wagner@solunar.de

Zusammenfassung: Um im globalen Wettbewerb bestehen zu können, müssen innovative mechatronische Produkte miteinander kommunizieren um sich untereinander zu koordinieren. Die Kommunikation und Koordination der einzelnen Systeme wird überwiegend durch Software realisiert, die durch steigende Anforderungen sowohl von der Struktur als auch vom Verhalten her immer komplexer wird. Zudem stellen kurze Time-To-Market-Zeiten die Entwickler vor eine besondere Herausforderung, da sie ihre Systeme erst dann testen und validieren können, wenn alle Bestandteile, d.h. sowohl die Hardware als auch die Software, fertiggestellt und integriert worden sind. In einem so späten Stadium ist die Behebung von Fehlern häufig mit hohen Kosten verbunden. Im vorliegenden Beitrag wird daher ein Ansatz vorgestellt, mit dem eine werkzeugunabhängige Simulation der Software- und Hardwaremodelle komplexer mechatronischer Systeme durchgeführt werden kann. Ein großer Vorteil dieses Ansatzes besteht darin, dass damit das Zusammenspiel zwischen Hardware und Software frühzeitig, d.h. noch vor der Erstellung physikalischer Prototypen, überprüft und gegebenenfalls korrigiert werden kann.

1 Einleitung

In Folge der zunehmenden Globalisierung der Absatzmärkte sind deutsche Unternehmen einem immer größer werdenden Wettbewerb ausgeliefert. Um in dieser Marktsituation bestehen zu können, müssen die angebotenen Produkte nicht nur immer mehr Funktionen anbieten, sondern auch immer ausgefeilter und intelligenter werden. Hierzu müssen sie sich nahtlos in ihre Umgebung integrieren, miteinander vernetzen und sich untereinander koordinieren, indem sie mit der Umgebung und anderen Geräten kommunizieren. Dies betrifft insbesondere mechatronische und eingebettete Systeme, zu denen nicht nur Produktionsanlagen und Systeme aus dem Automobilsektor [EJ09] sondern mittlerweile immer häufiger auch Produkte wie z.B. Waschmaschinen gehören.

Die hohen Anforderungen an komplexe technische Systeme können nur durch ein geschicktes Zusammenspiel von Mechanik, Elektrik/Elektronik, Regelungstechnik und Softwaretechnik realisiert werden. Hierbei ist in letzter Zeit vermehrt zu beobachten, dass durch immer günstigere und gleichzeitig leistungsfähigere Prozessoren mehr und mehr Funktionen in Software realisiert werden. Anstatt festverdrahtete Hardwareregler

zu installieren wird konfigurierbare Software verwendet, welche individuell an die Bedürfnisse des Kunden angepasst werden kann. Insbesondere die Vernetzung und Kommunikation der Geräte untereinander findet überwiegend in Software statt [EJ09].

Neben den technischen Herausforderungen erwartet der Markt zudem immer kürzere Innovationszyklen. Die kurzen Time-To-Market-Zeiten stellen die Entwickler vor eine besondere Herausforderung, da sie bisher ihre Systeme erst dann vollständig testen und validieren konnten, wenn alle Bestandteile, d.h. sowohl die Hardware als auch die Software, fertiggestellt und miteinander integriert worden sind. Hier können dann unter anderem Konstruktionsfehler, fehlerhaft errichtete Hardware und eine nicht ausgereifte Software zusammen kommen. Zu diesem späten Zeitpunkt ist eine Behebung der Fehler nur mit sehr viel Aufwand möglich. Dies verhindert kurze Time-To-Market-Zeiten.

Ein vielversprechender Ansatz für den disziplinübergreifenden Entwurf von Systemen stellt die modellgetriebene Systementwicklung dar, mit derer Hilfe die Zusammenarbeit der verschiedenen Disziplinen (Maschinenbau, Elektrotechnik, Regelungstechnik und Informatik) durch innovative Prozesse, Methoden und Werkzeuge vereinfacht wird. Hierbei werden die zu entwickelnden Systeme mit Modellen beschrieben. Diese Modelle können anschließend für eine gemeinsame Simulation verwendet werden.

Ein Problem stellt allerdings die Simulation diskreter Softwaremodelle für Echtzeitsysteme dar, die von gängigen Simulationswerkzeugen bisher kaum berücksichtigt wird. Zur Lösung dieses Problems wird in diesem Beitrag ein Ansatz vorgestellt, mit dem eine fachgebietsübergreifende Simulation eines zu entwickelnden mechatronischen Systems realisiert wurde. Hierbei liegt der Fokus auf dem Softwareentwurf, der gemeinsam mit einem Modell der Hardware simuliert wird. Für die fachgebietsübergreifende Konzipierung wird die Sprache „CONSENS“ (CONceptual design Specification technique for the ENgineering of complex Systems) verwendet [GFD+09]. Eine wesentliche Grundlage für den Softwareentwurf stellt die im Sonderforschungsbereich 614 und im Projekt ENTIME entwickelte MechatronicUML dar [BBB+12]. Ein Bestandteil dieser Spezifikationstechnik sind Zustandsdiagramme, mit denen Kommunikationsprotokolle mit Zeitannotationen und Prioritäten angegeben werden können. Sie sind zur Modellierung mechatronischer Echtzeitsysteme, die Zeitrestriktionen unterworfen sind, daher besonders gut geeignet.

In diesem Beitrag wird gezeigt, wie das Functional Mock-Up Interface (FMI) zur Simulation der Echtzeit-Zustandsdiagramme eingesetzt wurde. Bei dem FMI-Standard [MOD12] handelt es sich um ein neues Austauschformat für Simulationsmodelle, das im MODELISAR-Projekt entwickelt worden ist und eine immer größere Verbreitung erfährt [OZ12]. Mit Hilfe des FMI-Standards wurde eine Möglichkeit geschaffen, Zustandsdiagramme zusammen mit Modellen anderer Disziplinen zu simulieren. Auf Grundlage der Simulation lassen sich systemrelevante Aussagen über das zu entwickelnde System bereits in einer sehr frühen Phase machen. Insbesondere können dadurch Konstruktions- und Entwurfsfehler frühzeitig erkannt werden. Darüber hinaus ist es möglich, nach einer erfolgreichen Simulation die benötigte Softwareimplementierung automatisch aus dem Modell zu generieren. Dadurch entfällt eine Implementierung von Hand, wodurch einerseits das Risiko für fehlerhaften Code

verkleinert und andererseits die Zeit für die Implementierung drastisch reduziert wird. Der Ansatz wurden im Projekt „Entwurfstechnik Intelligente Mechatronik“ (ENTIME) entwickelt, an dem neun Partner aus der Industrie beteiligt sind.

Der vorliegende Beitrag hat den folgenden Aufbau: Abschnitt 2 gibt einen Überblick zum Entwurf und zur Konzipierung kooperierender mechatronischer Systeme am Beispiel autonom fahrender Miniaturroboter. Die Konzipierung führt zur Prinziplösung, auf deren Grundlage der Softwareentwurf konkretisiert wird. Darauf aufbauend wird in Abschnitt 3 skizziert, wie ein solcher Softwareentwurf mittels FMI/FMU ganzheitlich, d.h. im Zusammenspiel mit mechanischen und regelungstechnischen Systemelementen, werkzeugunabhängig simuliert werden kann. Abschnitt 4 fasst die Ergebnisse zusammen und schließt den Beitrag mit einem Ausblick auf zukünftige Arbeiten.

2 Entwurf kooperierender autonomer Miniaturroboter

Durch den Fortschritt der Kommunikations- und Informationstechnologie wird es möglich komplexe innovative mechatronische Systeme aufzubauen. Hierzu gehört auch eine Kolonnenfahrt von zwei fahrenden Miniaturrobotern. Damit ein solches Szenario erfolgreich, sicher und schnell entwickelt werden kann, bedarf es einer Methodik, welche es ermöglicht Fachdisziplinen übergreifend, gemeinsam zu spezifizieren und ganzheitlich zu simulieren. Die im Soderforschungsbereich (SFB) 614 entwickelte und im Projekt ENTIME verfeinerte Spezifikationsmethode [GSA+11] ermöglicht dies.

Der Entwurfsprozess mechatronischer Systeme ist ähnlich wie in der VDI-Richtlinie 2206 [VDI04] definiert in zwei Phasen unterteilt. Bild 1 stellt die erste Phase „fachgebietsübergreifende Konzipierung“ und die zweite Phase „fachgebietsspezifischer Entwurf und Ausarbeitung“ idealisiert dar. Für die fachgebietsspezifischen Anteile werden für die jeweilige Domäne und Aufgabe geeignete Sprachen verwendet.

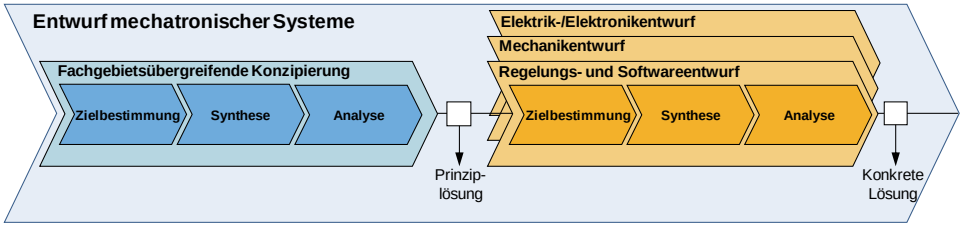

Bild 1: Prozessschritte beim Entwurf mechatronischer Systeme (vgl. [GSA+11])

Die modellgetriebene Methodik MechatronicUML [BBB+12] dient zur verfeinerten Spezifikation des Kommunikationsverhaltens. Die Sprache stellt eine Verfeinerung und Erweiterung der UML bzgl. Echtzeitverhalten, Komponentenspezifikation, Kommunikationsmuster/ -protokolle und Rekonfiguration dar. Sie verfügt über eine formale Semantik, welche es ermöglicht mittels Model-Checking die Modelle bzgl. zeitbehafteten Sicherheitseigenschaften zu verifizieren und mittels MATLAB Simulink [HPR+12] oder FMI/ FMU zu simulieren [PSR+12].

Im folgenden Abschnitt wird die fachgebietsübergreifende Konzipierung am Beispiel eines fahrenden Miniaturroboters erläutert um einzuführen welche Systemkomponenten mittels FMI/ FMU simuliert werden können. Bei dem Miniaturroboter nutzen wir den so genannten BeBot. Hierbei handelt es sich um eine am Heinz Nixdorf Institut entwickelte Testplattform [GSD+11].

2.1 Entwurf und Konzipierung

Die übergreifende Spezifikation der BeBot Kolonnenfahrt gliedert sich in sieben kohärente Partialmodelle. Zusammen bilden sie die sogenannte Prinziplösung. Bild 2 zeigt die unterschiedlichen Aspekte des BeBots.

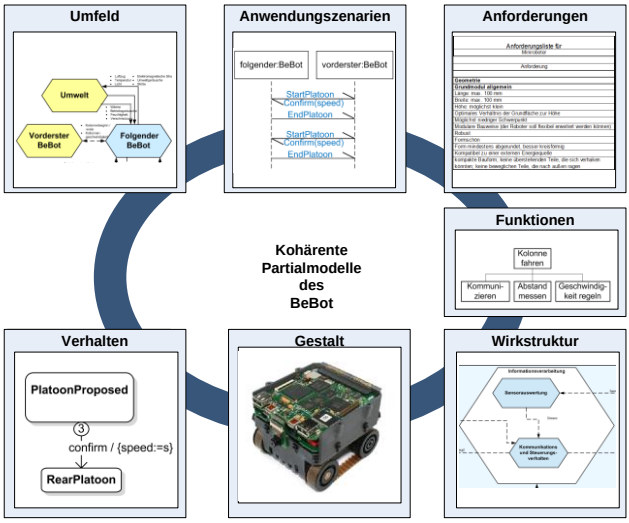

Bild 2: Partialmodelle der Prinziplösung

Das Umfeldmodell betrachtet den BeBot als Black Box und beschreibt die Einflüsse, welche auf den BeBot einwirken und die vom BeBot auf sein Umfeld ausgehen. Es werden auch die Wechselwirkungen mit anderen Systemelementen, wie zum Beispiel einen anderen BeBot, beschrieben.

Das Partialmodell Anwendungsszenarien beschreibt situationsspezifische Sichten auf das System. Sie können durch natürlichen sprachlichen Text und Querverweise auf Elemente der Prinziplösung oder durch sogenannte Modal Sequence Diagrams [AGD+12] spezifiziert werden. Diese stellen eine modale Erweiterung der UML 2 Sequenzdiagramme dar und haben eine formal definierte Semantik [HM06]. In unserem Fall beschreiben sie den Ablauf der auszutauschenden Nachrichten zwischen den BeBots beim Bilden einer Kolonne.

Die Anforderungen werden durch eine strukturierte Sammlung natürlich sprachlicher Texte und durch formale Beschreibungen in Form von zum Beispiel Timed Computation

Tree Logic (TCTL) Formeln, bei Sicherheitseigenschaften, beschrieben. Die Einhaltung der formalen Anforderungen kann später automatisch durch Model Checking auf dem erstellten Verhaltensmodell überprüft werden. Eine Sicherheitseigenschaft bei der Kolonnenfahrt ist, dass der Kolonnenkoordinator im Kooperationsmodus ist während der Kolonnenteilnehmer nicht im Kooperationsmodus ist.

Das Partialmodell Funktionen besteht aus einer hierarchischen Abgliederung der Funktionalität des Systems. Zum Beispiel werden für die Funktionen „Kolonne fahren“ die Teilfunktionalitäten „Kommunizieren“, „Abstand messen“ und „Geschwindigkeit regeln“ benötigt.

Die Wirkstruktur beschreibt die prinzipielle Wirkungsweise des Systems. Hierin sollten sich beteiligten Fachdisziplinen wiederfinden. Systemelemente, welche durch Sechsecke dargestellt werden, stellen Systeme, Module, Bauteile, oder Softwarekomponenten dar. Wechselwirkungen können Stoff-, Energie, Informationsflüsse, oder logische Beziehungen sein. Bild 3 zeigt einen Ausschnitt aus der Wirkstruktur des BeBots. Das Bild zeigt, dass ein folgender BeBot einen vorderen BeBot mittels IR-Sensor erkennen und den Abstand berechnen kann und dass er mittels eines Kommunikationsmoduls mit ihm kommunizieren kann.

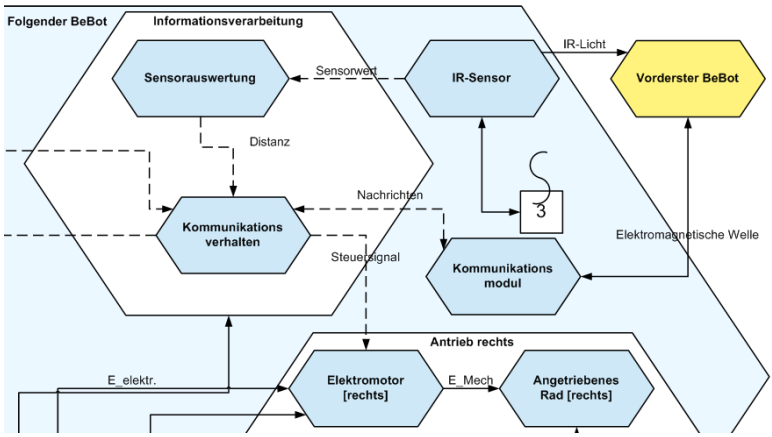

Bild 3: Ausschnitt aus der Wirkstruktur des BeBots

Das Gestaltmodell stellt eine 3D-Skizze des Systems dar. Beim BeBot können Abmessungen für Platinen und Bauteile bestimmt werden.

Eine wichtige Rolle bei modernen mechatronischen Systemen stellt das Verhalten dar. Dies wird mehr und mehr durch Software umgesetzt und ermöglicht somit eine hohe Variabilität. Verhalten kann durch Zustandsmodelle oder Aktivitäten dargestellt werden und durch modellgetriebene Techniken in simulierfähige Sprachen abgebildet werden. Dies ist bereits in der frühen Phase der Konzipierung wichtig, um die Umsetzung und prinzipielle Machbarkeit zu zeigen. Hierbei darf die Software nicht losgelöst von den

regelungstechnischen und mechanischen Eigenschaften betrachtet werden, da viele Einschränkungen hieraus resultieren.

2.2 Verhaltensmodelle im Entwurf und der Konzipierung

Das Verhaltensmodell zur Kolonnenfahrt besteht aus zwei parallelen Zustandsmaschinen (siehe Bild 4). Die Zustandsmaschine *Kolonnenbildung* steuert das Kommunikationsverhalten. Die Zustandsmaschine *Geschwindigkeitsteuerung* steuert die Geschwindigkeit des BeBots.

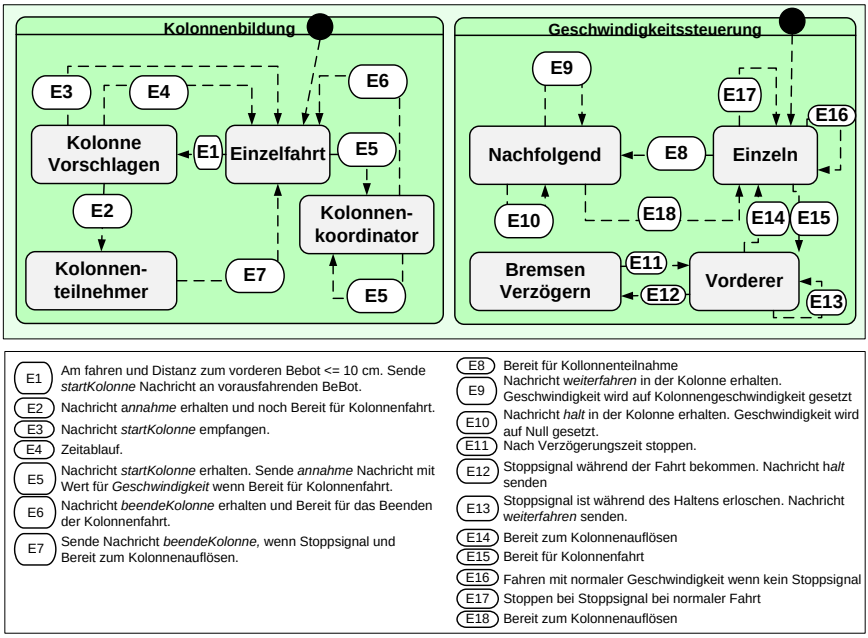

Bild 4: Verhalten für die Kolonnenfahrt

Es wird zwischen den Rollen Einzelfahrt als Einzelner, Kolonnenkoordinator als Vorderer und Kolonnenteilnehmer als Nachfolgender unterschieden. Je nach dem welches der achtzehn Ereignisse auftritt wird der Zustand gewechselt. Damit dieses Verhaltensmodell simuliert werden kann müssen zunächst die bisher informal beschriebenen Ereignisse formalisiert werden. Hierzu werden sie in asynchrone Nachrichten, Synchronisationen zwischen den Zustandsmaschinen und Guards an den Zustandsübergängen verfeinert. Für Zeitabläufe werden Uhren spezifiziert welche als Time Guards abgefragt werden können und durch betreten oder verlassen von Zuständen oder durch schalten von Zustandsübergängen zurückgesetzt werden. Weiterhin müssen noch konkurrierende Transitionen priorisiert werden. Eine Formalisierung könnte, ähnlich wie in [GTB03] für das Railcab gezeigt, realisiert werden. Folgend werden wir skizzieren, dass ein solches Modell mittels FMI ganzheitlich, d.h. im Zusammenspiel mit mechanischen und regelungstechnischen Systemelementen simuliert werden kann.

3 Functional Mockup Interface und Functional Mockup Unit

Die modellgetriebene Systementwicklung ermöglicht eine disziplinübergreifende Simulation mechatronischer Systeme, mit derer Hilfe systemrelevante Aussagen gemacht und beispielsweise Entwurfs- und Konstruktionsfehler noch vor der Entwicklung physikalischer Prototypen erkannt werden können. Hierzu ist es allerdings notwendig, die Prinziplösung zu konkretisieren und die Modelle der Software zusammen mit den Modellen der Hardware zu simulieren.

Auf Grund der großen Bandbreite der Anwendungsbereiche und der Vielzahl verschiedenartiger Entwicklungsaufgaben lässt es sich nicht vermeiden, dass sehr unterschiedliche und teilweise sehr spezialisierte Werkzeuge zur Modellierung in den jeweiligen Disziplinen eingesetzt werden müssen. Die Verwendung unterschiedlicher Werkzeuge führt spätestens bei der gemeinsamen Simulation der Modelle zu Problemen, da die Modelle nur schwer miteinander integriert werden können. Einen Ausweg aus diesem Dilemma verspricht der FMI-Standard.

3.1 Grundlagen

Der FMI-Standard dient als offener Standard für die Co-Simulation und den werkzeugübergreifenden Modellaustausch [MOD12]. Während der FMI-Standard für die Co-Simulation es ermöglicht, mehrere Simulationswerkzeuge miteinander zu koppeln, ermöglicht der FMI-Standard für den Modellaustausch, dass die Modelle zwischen den unterschiedlichen Werkzeugen ausgetauscht und simuliert werden können. Auf diese Art und Weise ist es möglich, in unterschiedlichen Werkzeugen erstellte Modelle in einem einzigen Werkzeug zu integrieren und gemeinsam zu simulieren.

Hierzu muss eine sogenannte Functional Mock-up Unit (FMU) erstellt werden. Eine FMU nutzt die im FMI-Standard definierten Schnittstellen, um ein Verhaltensmodell zu realisieren. Physikalisch ist eine FMU ein komprimiertes ZIP-Archiv, in dem neben der Implementierung alle zur Simulation benötigten Informationen in einer XML-Datei abgelegt werden. So enthält die XML-Datei beispielsweise eine Liste aller Variablen, deren Werte während der Simulation zwischen dem Simulator und der FMU ausgetauscht und aktualisiert werden. Der FMI-Standard definiert darüber hinaus Funktionen, die für die Interaktion zwischen Modell und Simulator benötigt werden. Diese Funktionen müssen innerhalb einer FMU in der Programmiersprache C implementiert werden.

3.2 Generierung von C-Code zur FMU-Erstellung

Eine manuelle Implementierung und Überführung von Softwaremodellen in eine FMU zu Zwecken der Simulation ist allein aus zeitlichen Gründen einem Anwender bzw. Produktentwickler nicht zumutbar. Im ENTIME-Projekt wurde daher ein Code-Generator entwickelt, der sowohl die Struktur als auch das Verhalten der Softwaremodelle automatisch in die Programmiersprache C überführt. Damit konnte die Erstellung einer FMU aus den MechatronicUML-Softwaremodellen automatisiert

werden. Bild 5 zeigt einen Überblick der Integration von Software- und Hardwaremodellen.

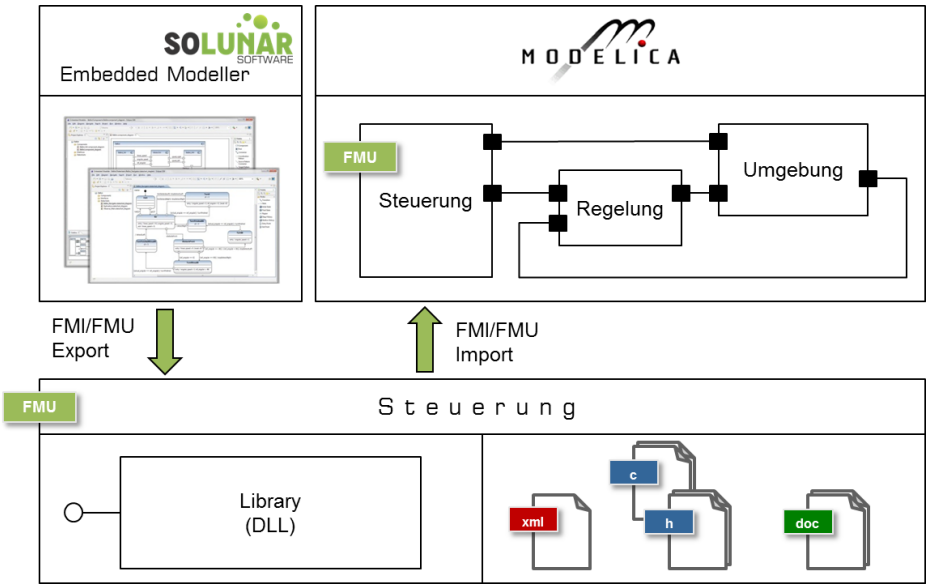

Bild 5: Überblick zur Integration von Software- und Hardwaremodellen zur Simulation am Beispiel der Simulationswerkzeuge Modelica/Dymola

Die Softwaremodelle werden im Embedded Modeller mit Hilfe von Komponenten- und Zustandsdiagrammen der MechatronicUML konkretisiert. Ein Komponentendiagramm bildet eine Hierarchie von zusammengesetzten Komponenten. Die Hierarchie spiegelt die Struktur der Software wieder. Komponenten auf der untersteten Hierarchiestufe besitzen ein erweitertes Zustandsdiagramm, mit dem das Verhalten einer Komponente spezifiziert wird. Die Erweiterungen für Zustandsdiagramme betreffen insbesondere Zeitannotationen, die zur Modellierung von Echtzeitverhalten eingesetzt werden.

Die Kommunikation und damit auch die Koordination zwischen Komponenten erfolgt über das Versenden und Empfangen diskreter Nachrichten. Hierzu sind die Komponenten über Kommunikationskanäle miteinander verbunden. Darüber hinaus können Komponenten über kontinuierliche Ein- und Ausgänge verfügen, über die Signale von Sensoren empfangen und an Aktuatoren gesendet werden können. Die Ein- und Ausgänge für kontinuierliche Signale werden später zur Simulation mit dem Regler- und Umgebungsmodell verbunden.

Die Softwaremodelle werden bei der Generierung auf verschiedenen Dateien aufgeteilt. Dabei wird aus jeder Komponente und aus jedem Verhaltensmodell eine eigene Übersetzungseinheit generiert. Dadurch wird eine Modularisierung der Software erreicht. Zur Kommunikation der Komponenten untereinander wird auf spezielle Bibliotheksfunktionen zurückgegriffen, die speziell für diesen Zweck implementiert worden sind.

Damit eine simulierbare FMU entsteht, muss der generierte C-Code konform zum FMI-Standard sein und zu einer Dynamic Link Library (DLL) übersetzt werden. Darüber hinaus muss eine XML-Datei erstellt und zur FMU hinzugefügt werden, in der alle während der Simulation einsehbaren Variablen beschrieben sind. Zusätzlich ist es möglich, den Quellcode (.h- und .c-Dateien) sowie weitere Dokumente und Bilder zur FMU hinzuzufügen (vgl. Bild 5).

Eine FMU kann in Software-Werkzeuge mit einer FMI/FMU-Import-Unterstützung eingelesen und dort ausgeführt werden. Zur Simulation müssen dafür zunächst die Ein- und Ausgänge der FMU mit den Ein- und Ausgängen der Regler- und/oder Umgebungsmodellen verbunden werden. Anschließend kann das Softwaremodell zusammen mit den Hardwaremodellen simuliert werden. Im ENTIME-Projekt wurde zur Simulation das Werkzeug Modelica/Dymola verwendet.

4 Resümee und Ausblick

Mit dem Aufkommen signalflussorientierter und gleichungsorientierter Sprachen sind mächtige Werkzeuge wie z.B. Matlab/Simulink oder Modelica/Dymola entstanden, mit denen Regelungstechniker und Maschinenbauer ihre technischen Systeme beschreiben und anschließend simulieren können. Für Softwaretechniker, die überwiegend diskrete Systeme mit einem hohen Aufkommen von Kommunikation und Koordination beschreiben müssen, sind die Möglichkeiten derzeit noch sehr eingeschränkt. Insbesondere wenn es sich um sicherheitskritische Echtzeitsysteme handelt, existieren nur sehr wenige Lösungsansätze [PDS+12].

In diesem Beitrag wurde eine Methode vorgestellt, mit der eine Prinziplösung erarbeitet, konkretisiert und anschließend simuliert und überprüft werden kann, noch bevor physikalische Prototypen erstellt werden. Hierbei wurde anhand der domänenspezifischen Modellierungssprache MechatronicUML gezeigt, wie diskrete Softwaremodelle zusammen mit Hardwaremodellen integriert und simuliert werden können, ohne dass physikalische Prototypen realisiert werden müssen. Der in diesem Beitrag dargestellte Ansatz ist nicht auf die MechatronicUML beschränkt, d.h. er lässt sich auf andere Modellierungssprachen, denen eine formale Semantik zugrunde liegt, wie z.B. Petri-Netze, übertragen.

In der derzeitigen Ausbaustufe müssen die Modelle aus der Prinziplösung von Hand in Modelle der MechatronicUML überführt werden. In zukünftigen Arbeiten wird dieser Schritt automatisiert werden, so dass es noch einfacher wird, die erstellten Modelle zu simulieren. Trotz erfolgreicher Simulation ist es bei der Entwicklung neuer Produkte unabdingbar, die Innovationen anhand physikalischer Prototypen zu überprüfen. Hierfür wird ausführbare Software benötigt. Diese wird bereits automatisch aus den vorhandenen Modellen mit Hilfe eines Codegenerators abgeleitet. Zur Unterstützung anderer Zielsysteme werden weitere Codegeneratoren erstellt. Zudem entwickelt sich der FMI-Standard weiter – die Version 2.0 steht kurz vor der Veröffentlichung. Eine weitere Arbeit wird daher darin bestehen, den Ansatz auf den neuen Standard zu migrieren.

Danksagung

Diese Arbeit ist 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.

Literaturverzeichnis

[AGD+12] Anacker, H..; Gausemeier, J..; Dumitrescu, R..; Dziwok, S..; Schäfer, W.: Solution Patterns of Software Engineering for the System Design of Advanced Mechatronic Systems: In Proc. of the 13th International Workshop on Research and Education in Mechatronics (REM 2012). Paris, 2012

[BBB+12] Becker, S.; Brenner, C.; Brink, C.; Dziwok, S.; Loeffler R.; Heinzemann, C.; Pohlmann, U.; Schäfer, W.; Suck, J.; Sudmann, O.: The MechatronicUML Method - Process, Syntax, and Semantics, 2012

[EJ09] Ebert C.; Jones C.: Embedded software: Facts, figures, and future. IEEE Computer, 42(4): 42-52, 2009

[GFD+09] Gausemeier, J.; Frank, U.; Donoth, J.; Kahl, S: Specification technique for the description of self-optimizing mechatronic systems. In: Research in Engineering Design vol. 20, Springer, London 2009, Nr. 4, S. 201–223.

[GSA+11] Gausemeier, J.; Schäfer, W.; Anacker, H.; Bauer, F.; Dziwok, S: Einsatz semantischer Technologien im Entwurf mechatronischer Systeme. In: 8. Paderborner Workshop Entwurf mechatronischer Systeme, 2011

[GSD+11] Gausemeier J.; Schierbaum T.; Dumitrescu R.; Herbrechtsmeier S.; Jungmann A.: Miniature robot bebot: Mechatronic test platform for self-x properties. In Proc. Of the 9th IEEE International Conference on Industrial Informatics (INDIN 2011), S. 451-456, 2011

[GTB+03] Giese, H.; Tichy, M.; Burmester, S.; Schäfer, W.; Flake, S.: In Proc. of the 9th European Software Engineering Conference held jointly with the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE '03) Helsinki, 2003

[HM06] Harel, D; Maoz, S: Assert and negate revisited: modal semantics for UML sequence diagrams. In Proc. of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools (SCESM '06). ACM, New York 2006, 13-20.

[HPR+12] Heinzemann, C.; Pohlmann, U.; Rieke, J.; Schäfer, W.; Sudmann, O.; Tichy, M: Generating Simulink and Stateflow Models From Software Specifications. In: Proc. of the 12th International Design Conference DESIGN 2012, 2012

[MOD12] MODELISAR Consortium. Functional mock-up interface for model exchange. Version 1.0, 2010. www.functional-mockup-interface.org

[OZ12] Otter, M; Zimmer, D.: Proc. of the 9th International Modelica Conference, Fürstenfeldbruck, Deutschland, Linköping University Electronic Press, 2012.

[PDS+12] Pohlmann, U.; Dziwok, S.; Suck, J.; Wolf B.; Loh C. C.; Tichy M.: A Modelica Library for Real-Time Coordination Modeling. In: Proc. of the 9th International Modelica Conference 2012, 2012

[PSR+12] Pohlmann, U.; Schäfer, W.; Reddehase, H.; Röckemann, J.; Wagner, R: Generating Functional Mockup Units from Software Specifications. In: Proc. of the 9th International Modelica Conference 2012, 2012

[VDI04] VEREIN DEUTSCHER INGENIEURE (VDI): Entwicklungsmethodik für mechatronische Systeme. VDI-Richtlinie 2206,. In: Beuth-Verlag, Berlin, 2004