Modellbasierte Entwicklung mit Rexroth-Steuergeräten unter Nutzung von offenen Standards
Model-based engineering using Rexroth controllers and open standards

Nils Menager, Lars Mikelsons, Niklas Worschech
Bosch Rexroth AG, 97816 Lohr a. Main, Deutschland,
{nils.menager, lars.mikelsons, niklas.worschech}@boschrexroth.de

Kurzfassung

Aufgrund der steigenden Anforderungen bezüglich Komplexität der Anlage und Verkürzung der Entwicklungszeit wird immer häufiger auf modellbasierte Entwicklungsmethoden zurückgegriffen, bei denen der Entwicklungsprozess bis hin zum Betrieb durch Modelle begleitet wird, was sowohl zu Zeit- als auch zu Kosteneinsparungen führt. Ein wesentlicher Bestandteil des modellbasierten Engineerings ist die Codegenerierung. Neben dem klassischen Rapid Control Prototyping, der Generierung von ausführbarem Regler-Code aus Simulationsmodellen, kann die Methode der Codegenerierung beispielsweise zu Diagnosezwecken (Fehlererkennung sowie Fehlerprädiktion) oder zur Realisierung von neuartigen Regelungsmethoden wie Model Predictive Control verwendet werden. In diesem Beitrag wird eine auf offenen Standards basierende Toolchain vorgestellt, mit deren Hilfe die modellbasierte Entwicklung mit Rexroth-Steuerungen ermöglicht wird. Die vorgestellte Toolchain wird anschließend verwendet, um aus dem Simulationsmodell eines Reglers ausführbaren Code zur Regelung eines hydraulischen Antriebs zu generieren.

Abstract

Due to the increasing requirements regarding the complexity of technical systems and the decrease of development times, model-based engineering methods are used more and more. This means, that the entire development process through to the commissioning is supported by models, which leads to both time and cost reductions. An important part of model-based engineering is the code generation. In addition to the classical Rapid Control Prototyping, i.e. the generation of executable controller code from simulation models, the method of code generation can for example be used for diagnosis (error detection and error prediction) or for the realization of new control strategies such as Model Predictive Control. In this paper, a toolchain based on open standards is presented, which provides the possibility to realize model-based engineering for Rexroth controllers. This toolchain is finally used to generate executable code from a simulation model in order to control a hydraulic drive.

1 Einleitung

1.1 Motivation

Die zunehmende Komplexität heutiger Systeme erfordert die Abkehr von konventionellen Entwicklungsmethoden hin zum modellbasierten Engineering. Hierbei wird der gesamte Entwicklungsprozess bis hin zum Betrieb durch Modelle begleitet. Eine konsequente Verfolgung dieses Ansatzes spart sowohl Zeit als auch Kosten, beispielsweise durch die Vermeidung mehrfacher Implementierungen und kürzerer Iterationen. Eine wesentliche Ausprägung des modellbasierten Engineerings ist die Codegenerierung, also die Erzeugung von Code aus Simulations- bzw. Engineeringtools heraus. Der generierte Code kann für eine Vielzahl von Anwendungen verwendet werden. Ein klassisches Einsatzfeld ist das *Rapid Control Prototyping*. Während des Entwicklungsprozesses einer neuen Anlage wird in der Regel ein Simulationsmodell der Anlage innerhalb einer Simulationsumgebung erstellt. Für eine Untersuchung der Dynamik des Systems ist innerhalb der Simulationsumgebung zusätzlich ein Modell des Reglers erforderlich. In einer späteren Phase des Entwicklungsprozesses ist es notwendig, den Regler unter realen Bedingungen, d.h. auf echter Regelungshardware, zu testen, um beispielsweise die Echtzeitfähigkeit des Regelungs-Algorithmus zu validieren. An dieser Stelle wird bisher die bereits vorhandene Implementierung in Form des Simulationsmodells nicht weiter verwendet. Stattdessen wird innerhalb des Engineeringtools der Steuerung der Regelungs-Algorithmus komplett neu implementiert. Häufig wird dabei auf einen bereits implementierten Standard-Regler zurückgegriffen, auch wenn dieser unter Umständen weniger performant ist. Dieses Vorgehen bringt insgesamt einige wesentliche Nachteile mit sich.

Zunächst bedeutet eine Re-Implementierung bereits vorhandener Teile immer zusätzlichen Zeit- und damit Kostenaufwand sowie eine potentielle Fehlerquelle. Da der Regler bei der Re-Implementierung in der Regel in einer anderen Sprache (zumeist SPS Programmiersprachen nach IEC 61131) implementiert wird als innerhalb der Simulationsumgebung, kann nicht garantiert werden, dass sich beide Regler gleich verhalten. Aus diesem Grund ist es wünschenswert, den bereits vorhandenen Regler aus der Simulationsumgebung auf der Regelungs-Hardware wei-

terverwenden zu können. Ein Lösungsansatz ist es, aus dem Simulationsmodell des Reglers ausführbaren Code zu generieren, der auf der Steuerung in Echtzeit ausgeführt wird.

Neben der Weiterverwendung des Simulationsmodells des Reglers als Regelungs-Algorithmus auf der Steuerung ist es ebenso möglich, das Simulationsmodell der Regelstrecke weiterzuverwenden und den daraus generierten Code auf der Steuerung auszuführen. Dies kann beispielsweise zu Diagnosezwecken eingesetzt werden. Hierbei wird mit Hilfe des Simulationsmodell das erwartete Verhalten des Systems vorausberechnet und mit dem tatsächlichen, von Sensoren gemessenen Verhalten, verglichen. Abweichungen zwischen erwarteten und gemessenen Ergebnissen lassen auf einen Fehlerfall schließen und ermöglichen, bei entsprechend genauen Fehlermodellen, unter Umständen sogar eine Identifikation der Ursache des Fehlers. Zusätzlich ist es denkbar, auch zukünftig auftretende Fehler frühzeitig detektieren zu können, was zu einer Reduzierung der Stillstandszeiten führen kann.

Das Simulationsmodell der Regelstrecke kann darüber hinaus auch für moderne Regelungsstrategien wie *Model Predictive Control* verwendet werden. Hierbei wird mit Hilfe des dynamischen Simulationsmodells der Anlage, welches ohnehin während des Entwicklungsprozesses entsteht, das zukünftige Verhalten der Anlage vorausberechnet. Diese Vorhersage des zukünftigen Verhaltens wird in Kombination mit der Messung des aktuellen Zustands dazu verwendet, mit Hilfe mathematischer Optimierungsverfahren einen optimalen Systemeingang (bezogen auf einen vorher definierten Sollzustand) für den nächsten Zeitschritt zu berechnen.

Es existieren bereits Möglichkeiten, Code aus Simulationsmodellen zu generieren und diesen auf einer Echtzeit-Hardware auszuführen, beispielsweise bei der Verwendung einer Toolchain auf Basis von MATLAB/Simulink. Für Bosch Rexroth hat diese Toolchain jedoch einige Nachteile. Ein Grund dafür ist, dass die beschriebene Toolchain aufgrund der hohen Lizenzgebühren sehr kostenintensiv ist. Der Großteil der Rexroth-Kunden haben häufig keine Möglichkeit, diese Toolchain zu verwenden, weswegen die Anwendung der Methoden der modellbasierten Entwicklung diesen Kunden nicht möglich ist. Des Weiteren ist das Codegenerierungs-Modul von MATLAB/Simulink eine Blackbox und kann nicht verändert werden. Sollen bereits bestehende Funktionen (z.B. einer API) unmittelbar mit in den generierten Code integriert werden, ist es nicht möglich, das Codegenerierungs-Modul anzupassen. Ein dritter Nachteil ist die Release-Häufigkeit der Software (i.d.R. zweimal jährlich). Da bei einem neuen Release nicht ersichtlich ist, welche Änderungen an der Codegenerierung vorgenommen wurden, müssen alle bestehenden Modelle mit der Erscheinung einer neuen MATLAB/Simulink Version neu getestet werden. Zusätzlich ist MATLAB/Simulink zwar sehr gut zum Regler-Design geeignet, für die Beschreibung der Regelstrecke ist Simulink jedoch nur mäßig geeignet. Hierfür sind objekt-orienterte Modellierungssprachen wie Modelica deutlich besser geeignet. Aus diesem Grund sind häufig zwei unterschiedliche Simulationsumgebungen zur Beschreibung des Reglers und der Regelstrecke notwendig.

Daher wird in diesem Beitrag eine alternative Toolchain vorgestellt, die die zuvor genannten Nachteile nicht aufweist. Die auf offenen Standards basierende Toolchain kann schließlich verwendet werden, um Code aus Simulationsmodellen zu erzeugen und diesen auf Rexroth-Industriesteuerungen auszuführen. Zur Validierung der Funktionsweise der Toolchain wird ein Beispiel für den Anwendungsfall *Rapid Control Prototyping* gewählt. Aus einer Simulationsumgebung heraus wird automatisiert Regler-Code erzeugt, der anschließend in Echtzeit auf der Steuerung ausgeführt wird.

1.2 Aufbau des Beitrags

Dieser Beitrag ist wie folgt gegliedert. Im zweiten Abschnitt wird die auf offenen Standards basierende Toolchain vorgestellt. Dabei werden zunächst die Anforderungen an die Toolchain abgeleitet und anschließend die einzelnen Komponenten beschrieben, die darin zum Einsatz kommen. Anschließend werden diese Komponenten miteinander verknüpft und die Gesamtstruktur der Toolchain präsentiert. Dabei werden zwei unterschiedliche Arten von Steuerungen unterschieden (Industriesteuerungen sowie Mobilsteuerungen), welche beide durch die Toolchain unterstützt werden. Der dritte Abschnitt beschäftigt sich mit einer speziellen Komponente der Toolchain, dem Simulationskern. Im Fokus liegen die numerischen Verfahren zum Lösen der Modellgleichungen. Aufgrund der Echtzeitanforderung der Simulation sind lediglich spezielle Verfahren zulässig. Im vierten Abschnitt wird diese Toolchain verwendet, um aus dem Simulationsmodell eines Reglers Code zu erzeugen und diesen Code auf einem Rexroth Industriesteuerungen auszuführen. Dies entspricht dem Einsatzgebiet *Rapid Control Prototyping*. Abschließend folgt eine Zusammenfassung sowie ein Ausblick auf aktuelle Arbeiten.

2 Toolchain für modellbasierte Entwicklung

Wie bereits in der Einleitung erwähnt, basiert eine häufig genutzte Toolchain für modellbasierte Entwicklung auf MATLAB/Simulink. Um die daraus resultierenden Nachteile zu umgehen, wird eine auf offenen Standards basierende Toolchain entwickelt. Im Folgenden werden zunächst die Anforderungen an die Toolchain präsentiert.

2.1 Anforderungen an die Toolchain

Bosch Rexroth vertreibt sowohl Industriesteuerungen (z.B. Rexroth IndraControl XM22) als auch Mobilsteuerungen (z.B. BODAS RC Controller). Eine Anforderung an die Toolchain ist es, dass beide Arten von Steuerungen ohne Modifikationen an den Steuerungen unterstützt wer-

den. Die weiteren Anforderungen ergeben sich unmittelbar aus den Nachteilen der MATLAB/Simulink-Toolchain, die in der Einleitung beschrieben wurden. Die Codegenerierung soll flexibel sein und bei Bedarf angepasst werden können, um beispielsweise bereits vorhandene Funktionen einer Schnittstelle verwenden zu können. Weiterhin soll die Toolchain nicht hauptsächlich auf kommerziellen Tools basieren, sondern stattdessen offene Standards verwenden, um einerseits externe Abhängigkeiten und andererseits Kosten zu vermeiden. Eine weitere Anforderung ist es, dass die Toolchain einfach und intuitiv zu bedienen ist.

2.2 Komponenten der Toolchain

Modelica/Compiler Den Ausgangspunkt der entwickelten Toolchain stellen Modelica-Modelle dar. Modelica ist eine objekt-orientierte Sprache zur Modellierung domänenübergreifender physikalischer Systeme [1]. Um Modelica-Modelle simulieren zu können, wird ein Modelica-Compiler benötigt. Es existieren sowohl kommerzielle (z.B. Dymola, SimulationX) als auch Open-Source-Modelica-Compiler (z.B. OpenModelica, JModelica.org). Für die Entwicklung dieser Toolchain wird der OpenModelica-Compiler verwendet. Ein Teil des Compilers ist das Codegenerierungs-Modul. An dieser Stelle besitzt Bosch Rexroth ein eigenentwickeltes Codegenerierungs-Modul zur Generierung von C++-Code aus den Modelica-Modellen. Aus diesem Grund ist die Codegenerierung komplett flexibel, da die Codegenerierung je nach Bedarf angepasst werden kann. Für den detaillierten Aufbau des OpenModelica-Compilers und der Funktionsweise der einzelnen Komponenten sei auf [2] verwiesen.

Simulationskern Der mit Hilfe des Modelica-Compilers erzeugte C++-Code beinhaltet die Modellbeschreibung bzw. die zugrunde liegenden mathematischen Gleichungen. Allerdings beinhaltet der Code keine Informationen darüber, wie die Gleichungen gelöst werden. An dieser Stelle wird ein Simulationskern benötigt, der unter anderem die numerischen Verfahren zur Lösung der mathematischen Gleichungen beinhaltet. Die Forderung der Echtzeitfähigkeit der Simulation ist eine harte Anforderung an die numerischen Verfahren, weswegen spezielle Methoden verwendet werden müssen. Darauf wird in **Kapitel 3** detaillierter eingegangen. Weiterhin hat der Simulationskern die Aufgabe, den Ablauf der Simulation zu steuern (z.B. auftretende Events zu behandeln) sowie die benötigten Simulationsergebnisse zu speichern. Der verwendete Simulationskern wird ebenfalls von Bosch Rexroth entwickelt [3]. Dieser ist in C++ geschrieben und unterstützt unmittelbar die mit Hilfe der Codegenerierung des OpenModelica-Compilers generierten Modelle. Sofern Teile der Codegenerierung geändert werden, besteht somit unmittelbar die Möglichkeit, die entsprechend notwendigen Änderungen an dem Simulationskern ebenfalls vorzunehmen, was eine vollkommene Flexibilität ermöglicht.

Compiler für Zielhardware Um den Code auf der Zielhardware ausführen zu können, muss dieser mit Hilfe eines Compilers für das Betriebssystem der Zielhardware kompiliert werden. An dieser Stelle unterscheiden sich die verschiedenen Steuerungen. Auf den Bosch Rexroth Industriesteuerungen läuft das Echtzeit-Betriebssystem *VxWorks*, während die Mobilsteuerungen auf einem TriCore-Chip basieren und, ähnlich wie Mikrocontroller, kein richtiges Betriebssystem verwenden. Folglich werden für die verschiedenen Plattformen unterschiedliche Compiler benötigt. Für die Industriesteuerungen wird ein spezieller VxWorks-Compiler der Firma *Windriver* verwendet, für die Mobilsteuerungen steht ein TriCore-Compiler der Firma *HighTec* zur Verfügung. Mit Hilfe des jeweiligen Compilers kann letztendlich ausführbarer Code für die Zielhardware erzeugt werden. Das Ergebnis des Kompilierungsvorgangs ist entweder eine Bibliothek (Industriesteuerungen) oder eine *.hex*-Datei (Mobilsteuerungen).

Engineering-Umgebung Der letzte notwendige Schritt ist die Integration des kompilierten, ausführbaren Codes in ein Projekt, welches auf die Steuerung geladen werden kann. Auch hier muss zwischen den verschiedenen Steuerungen unterschieden werden. Wünschenswert ist es, dass der Inbetriebnahme-Ingenieur weiterhin die gewohnte Entwicklungsumgebung verwenden kann und die Integration des Codes in das Steuerungsprojekt weitestgehend automatisiert erfolgt.

Im Fall der Industriesteuerung besteht die Möglichkeit, in der Entwicklungsumgebung der Steuerung (*Rexroth IndraWorks*) wie gewohnt ein Projekt anzulegen, welches aus unterschiedlichen Funktionen und Funktionsbausteinen besteht. Statt die Implementierung des Funktionsbausteins in SPS-Programmiersprachen nach IEC 61131 vorzunehmen, besteht unter anderem die Möglichkeit, eine externe Implementierung zu dem Baustein hinzuzulinken. An dieser Stelle wird das *OpenCore Interface* [4] verwendet. Dies ist eine hauseigene Schnittstelle (verfügbar in verschiedenen Hochsprachen wie C/C++, C#, LabView, Matlab, Modelica), um von außen auf die Steuerung zugreifen zu können. Diese beinhaltet unter anderem Funktionen zum Lesen oder Schreiben von Steuerungsparametern und Variablen, Starten oder Stoppen von Applikationen, Triggern von Tasks oder Ausführen von Motion-Befehlen. Weiterhin bietet das OpenCore Interface die zuvor erwähnte Funktionalität, extern implementierte Funktionen zu Funktionsbausteien hinzuzulinken.

Werden Mobilsteuerungen verwendet, ist zum Bespielen der Steuerung die Entwicklungsumgebung *BODAS service* notwendig. Das Tool bietet die Möglichkeit, ein vorhandenes *.hex*-File auf die Steuerung zu laden und stellt darüber hinaus Funktionalitäten zu Diagnosezwecken bereit, beispielsweise das Anzeigen und Ändern von Parametern und Variablen. Die Kopplung zwischen dem auszuführenden Modellcode und der Steuerung basiert auf einer Schnittstelle (API) der Steuerung. Diese API stellt Funktionen bereit, um beispielsweise Tasks anzulegen oder auszuführende Programmteile einem Task zuzuordnen. An dieser Stelle

ist es wichtig, dass die Codegenerierung modifizierbar ist, damit die notwendigen Schnittstellen-Funktionen direkt in den generierten Code mit eingebunden werden können.

2.3 Gesamtstruktur der Toolchain

Die zuvor dargestellten Komponenten werden schließlich miteinander verbunden, sodass eine durchgängige Toolchain vom Modelica-Modell hin zu dem Projekt entsteht, welches auf der jeweiligen Steuerung ausgeführt wird. Die Gesamtstruktur ist in **Bild 1** dargestellt. Die Abarbeitung der aufeinanderfolgenden Schritte lässt sich vollständig automatisieren, sodass der Anwender lediglich das Modelica-Modell aufbaut und dieses Modelica-Modell einem Skript übergibt. Als Ergebnis kann unmittelbar die Bibliothek erhalten werden, die der Anwender durch Anhängen an einen Funktionsbaustein in sein SPS-Projekt einbinden kann. Auf diese Weise ist es möglich, ein Steuerungsprojekt wie gewohnt zu erstellen, wobei einzelne Funktionsbausteine eine Implementierung in SPS-Sprachen und einzelne Funktionsbausteine in C/C++-Code beinhalten können. Im Fall der Mobilsteuerungen ist die Ausgabe des Skriptes ein *.hex*-File, welches anschließend mit Hilfe des gewohnten Programms *BODAS service* auf die Steuerung gespielt werden kann.

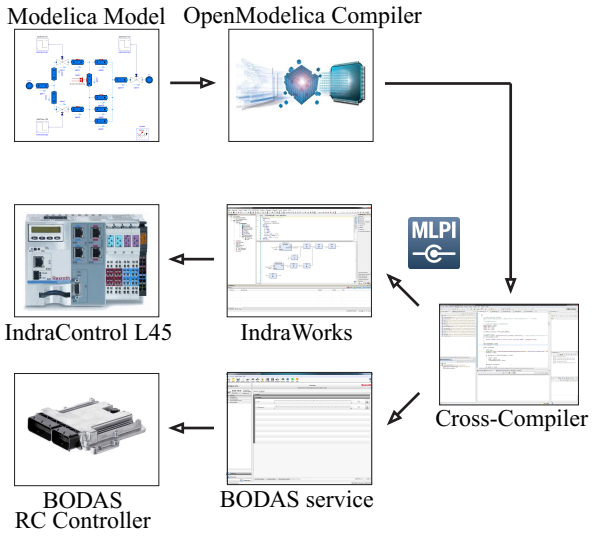

Bild 1 Struktur der Toolchain

3 Numerische Verfahren für die Echtzeitsimulation

Um die mathematischen Gleichungen (bei denen es sich in der Regel um gewöhnliche Differentialgleichungen (DGL) handelt), die in dem generierten Code integriert sind, lösen und den Simulationsablauf steuern zu können, wird der Simulationskern benötigt. Dieser Simulationskern beinhaltet den DGL-Löser und wird zusammen mit dem Modellcode für das Zielsystem kompiliert und dort ausgeführt. Im Folgenden werden zunächst allgemeine Anforderungen an die Funktionsweise von echtzeitfähigen Lösungsverfahren diskutiert. Anschließend wird auf eine spezielle Klasse von geeigneten Methoden eingegangen, die linear-impliziten Runge-Kutta-Verfahren (Rosenbrock-Verfahren).

3.1 Anforderungen an Echtzeit-Löser

Die Entwicklung von DGL-Lösern versucht in der Regel, die Berechnungszeit so minimal wie möglich zu halten, ohne jedoch auf Genauigkeit oder Stabilität der Lösung zu verzichten. Dafür verwenden die gebräuchlichen Löser wie das Dassl- oder das Radau-Verfahren unter anderem Methoden wie eine variable Schrittweitensteuerung, die eine geringe Schrittweite nur dann verwendet, sofern dies erforderlich ist. Zusätzlich wird die Jacobi-Matrix lediglich dann neu ausgewertet, wenn keine Konvergenz erreicht werden konnte.

Da die Ausführung des Codes hier in Echtzeit erfolgen muss, müssen entsprechend harte Bedingungen an die numerischen Verfahren gestellt werden. Es muss in jedem Fall garantiert werden, dass nach Ablauf eines Zyklus die Berechnung des Modells um eben diese Zykluszeit fortgeschritten ist. Daher muss immer die worst-case-Laufzeit betrachtet werden. Falls eine geringe Schrittweite auch nur an einer Stelle benötigt wird, so muss mit dieser Schrittweite kalkuliert werden. Weiterhin tritt jedoch das Problem auf, dass die gängigen impliziten Verfahren in jedem Zeitschritt mindestens ein nichtlineares Gleichungssystem lösen müssen. Diese Systeme erfordern ein iteratives Verfahren wie das Newton-Verfahren zur Lösung, für welches jedoch keine feste Anzahl an Iterationen vorgegeben werden kann, innerhalb der die Lösung konvergiert.

Zur Lösung der Gleichungen werden folglich Verfahren mit einer festen Schrittweite benötigt, die zudem keine nichtlinearen Gleichungen lösen müssen. Es ist offensichtlich, dass die expliziten Runge-Kutta-Verfahren diese Anforderungen erfüllen. Der einfachste Vertreter dieser Klasse, das explizite Euler-Verfahren, ist eines der am häufigsten eingesetzten Lösungsverfahren zur Lösung von Differentialgleichungen unter Echtzeitbedingungen. Leider weisen explizite Verfahren zumeist einen stark eingeschränkten Stabilitätsbereich auf, was vor allem bei steifen Systemen problematisch ist. Daher ist es notwendig, den Simulationskern um Verfahren zur Lösung von steifen Systemen, die mit Hilfe des expliziten Euler-Verfahrens oder des klassischen Runge-Kutta-Verfahrens 4. Ordnung nicht gelöst werden können, zu ergänzen.

In der Regel besitzen implizite Runge-Kutta-Verfahren gute Stabilitätseigenschaften. Jedoch erfordern diese in der Regel die Lösung eines nichtlinearen Gleichungssystems. Eine Linearisierung der impliziten Runge-Kutta-Verfahren führt auf die Klasse der Rosenbrock-Verfahren, auch bekannt als linear-implizite Runge-Kutta-Verfahren. Diese bieten die selben Stabilitätseigenschaften, ersetzen jedoch die Lösung des nichtlinearen Gleichungssystems durch die Lösung eines linearen Gleichungssystems, was nichtiterativ möglich ist. Auf diese Klasse von Lösungsverfahren wird im nächsten Abschnitt eingegangen.

3.2 Rosenbrock-Verfahren

Für Modellgleichungen, gegeben in der Zustandsform

$$\frac{\partial y}{\partial t} = f(y,t) \qquad y(t_0) = y_0 \tag{1}$$

$$f\colon \mathbb{R}^n \times [t_0,\infty] \to \mathbb{R}^n \quad t_0 \in \mathbb{R} \quad y_0 \in \mathbb{R}^n, \tag{2}$$

ist der einfachste Vertreter der Klasse der Rosenbrock-Verfahren das linear-implizite Euler-Verfahren, welches folgendermaßen definiert ist:

$$\left(\frac{1}{h}I - \frac{\partial f}{\partial y}\left(t_n,y_n\right)\right)u = f\left(t_n,y_n\right) + \frac{\partial f}{\partial t}\left(t_n,y_n\right) \tag{3}$$

$$y_{n+1} = y_n + u. \tag{4}$$

Für Methoden mit einer höheren Ordnung werden mehrere Stufen benötigt. Dafür wird die effiziente Implementierung von [5] verwendet. Ein Schritt ist hierbei gegeben durch

$$\left(\frac{1}{h\gamma_{ii}}I - \frac{\partial f}{\partial y}\left(t_n,y_n\right)\right)u_i = f\left(t_n + \alpha_i h, y_n + \sum_{j=1}^{i-1} a_{ij}u_j\right) \tag{5}$$

$$+ \sum_{j=1}^{i-1} \frac{c_{ij}}{h}u_j + \gamma_i h \frac{\partial f}{\partial t}\left(t_n,y_n\right) \qquad i = 1,\ldots,s \tag{6}$$

$$y_{n+1} = y_n + \sum_{j=1}^{s} m_j u_j, \tag{7}$$

wobei $\gamma_{ij}, \alpha_i, c_{ij}, m_i$ Konstanten sind. Werden spezielle Kombinationen von Konstanten gewählt, können Methoden mit verschiedener Ordnung erhalten werden, die jeweils eine unterschiedliche Anzahl an Funktionsauswertungen benötigen. In dem verwendeten Simulationskern werden insbesondere die Methoden verwendet, die eine, zwei bzw. drei Funktionsauswertungen benötigen, was jeweils zu einem Rosenbrock-Verfahren der Ordnung eins (linear-implizites Euler-Verfahren), drei (ROS3P [6]) bzw. vier (L-stable-Methode nach [5]) führt.

Die auftretende Jacobi-Matrix wird mit Hilfe der Methode der finiten Differenzen numerisch berechnet und wird einmal pro Schritt neu berechnet. Für die Auswertung der Jacobi-Matrix wird die effiziente Methodik der *Colored Jacobians* verwendet [7].

4 Anwendungsfall Rapid Control Prototyping

Die in **Kapitel 2** vorgestellte Toolchain soll im Folgenden verwendet werden, um einen in Modelica entwickelten Regler auf einer Steuerung auszuführen. Prinzipiell funktioniert die Toolchain sowohl für Industriesteuerungen als auch für Mobilsteuerungen sehr ähnlich. Im Rahmen dieses Beitrages wird daher lediglich die modellbasierte Entwicklung für ein Bosch Rexroth Industriesteuerung (Rexroth IndraControl XM22) durchgeführt. Als Beispiel dient die Auslegung eines hydro-mechanischen Systems, konkret die Auslegung einer Presse. Die Struktur des Modelica-Modells ist in **Bild 2** dargestellt. Das Modell besteht aus einer Druckversorgung (innerhalb der Komponente HPU), einem Ventil sowie einem Differentialzylinder. Für die Regelung der Anlage ist ein Soll-Geschwindigkeitsprofil für den Zylinderkolben vorgegeben. Das daraus abgeleitete Positionsprofil ist in **Bild 3** in Form der blauen Kurve gezeigt.

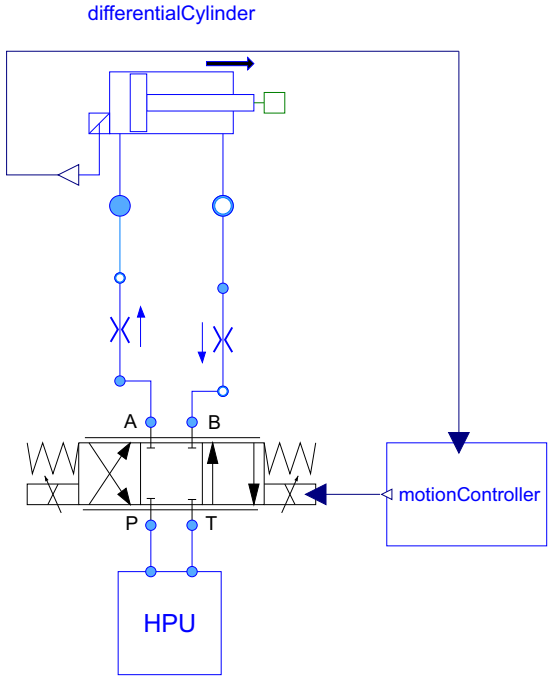

Bild 2 Modell der Regelstrecke

Um die Dynamik des Systems untersuchen zu können, befindet sich innerhalb des *MotionController*-Blockes der Regler. Dabei handelt es sich um einen Positionsregler mit einer Geschwindigkeitsvorsteuerung. Aus diesem Regler wird mit Hilfe der vorgestellten Toolchain Code zur Ausführung auf der Steuerung generiert. Nachdem der OpenModelica-Compiler aus dem Modelica-Modell C++-Code generiert hat und dieser, gemeinsam mit dem Simulationskern, mit Hilfe des Windriver VxWorks Compilers kompiliert wurde, befindet sich der gesamte kompilierte Code in einer Bibliothek. Diese Bibliothek wird auf den internen Speicher der Industriesteuerung verschoben und wie gewohnt ein SPS-Projekt innerhalb von IndraWorks angelegt. Für den Regler wird innerhalb des Projektes ein neuer Funktionsbaustein angelegt, welcher einen Eingang und einen Ausgang besitzt. Die Anzahl der Ein- und Ausgänge entspricht dabei genau der Anzahl im Modelica-Modell (vgl. **Bild 2**). Innerhalb IndraWorks muss in den Funktionsbaustein-Eigenschaften noch die Option *externe Implementierung* ausgewählt werden. Die Zuordnung zwischen der Bibliothek und dem Funktionsbaustein erfolgt mit Hilfe von OpenCore Engineering-Befehlen. Der benötigte Code wird bereits von der Codegenerierung mit erzeugt und befindet sich innerhalb der Bibliothek. Die Ausführung des Codes erfolgt automatisch beim Starten der Steuerung.

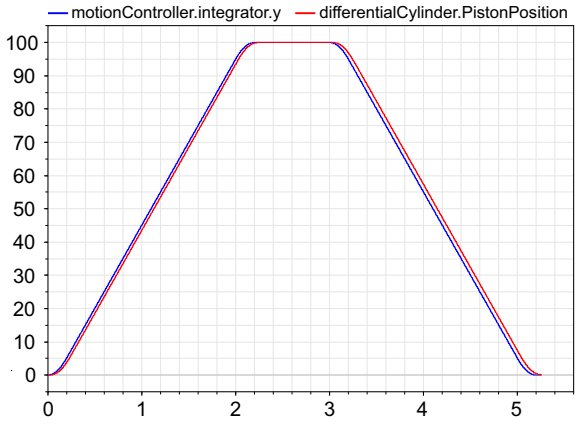

Bild 3 Sollposition (blau) sowie Istposition (rot) des Zylinderkolbens nach der HiL-Simulation

Virtuelle Inbetriebnahme Die virtuelle Inbetriebnahme ist eine weitere Ausprägung modellbasierter Entwicklung. Die Validierung des Reglers erfolgt in diesem Beitrag an der virtuellen Regelstrecke. Dazu ist eine Hardware-In-The-Loop-Kopplung zwischen Simulationsumgebung und Steuerung notwendig. Wie bereits in Abschnitt 2.2 erwähnt, besteht die Möglichkeit, mit Hilfe des *OpenCore Interfaces* von außen auf die Funktionen der Steuerung zugreifen zu können. Um diese Funktionen in Modelica verwenden zu können, wurde die Modelica-Bibliothek *mlpi4Modelica* entwickelt, die die C/C++-Funktionen kapselt und als Modelica-Funktionen verfügbar macht. Diese Funktionen werden verwendet, um eine Modelica-Komponente *MLPICoupler* zu entwickeln, die sämtliche notwendige Aufgaben für eine HiL-Simulation übernimmt, unter anderem das Setzen der Eingänge der Steuerung, das Lesen der Ausgänge der Steuerung sowie die Synchronisation zwischen Simulationsumgebung und Steuerung.

Für die Hardware-In-The-Loop-Simulation wird im Simulationsmodell der Block *motionController* durch die *MLPICoupler*-Komponente ersetzt. Das Simulationsergebnis der Zylinderkolbenposition ist als rote Kurve in **Bild 3** dargestellt. Zwischen der Sollkurve und der Istkurve ist bereits eine gute Übereinstimmung zu erkennen. Für den Betrieb an der finalen Anlage sind die Reglerparameter aufgrund des unterschiedlichen Verhaltens von realer Anlage und Simulationsmodell in der Regel noch in geringem Rahmen nachzujustieren. Insgesamt ergibt sich durch die Verwendung von modellbasierten Entwicklungsmethoden eine deutliche Zeitreduzierung, da der Regler rein virtuell ausgelegt wird und virtuell in Betrieb genommen wird. Es muss für die Inbetriebnahme des Reglers folglich nicht gewartet werden, bis die Anlage verfügbar ist. An der realen Anlage ist lediglich die Nachjustierung vorzunehmen.

5 Zusammenfassung und Ausblick

Eine häufig zur modellbasierten Entwicklung verwendete Toolchain basiert auf der Verwendung von MATLAB/Simulink. Aufgrund einiger Nachteile, die mit der Verwendung dieser Toolchain einhergehen, wird in diesem Beitrag eine alternative Lösung vorgestellt. Diese verwendet offene Standards (Modelica als offener Sprachstandard zur Beschreibung physikalischer Systeme, OpenSource-Modelica-Compiler) und erlaubt das automatisierte Ausführen von aus Simulationsmodellen generiertem Code direkt auf Bosch Rexroth Steuerungen. Die Methoden der modellbasierten Entwicklung lassen sich auf unterschiedliche Einsatzgebiete anwenden, darunter Rapid Control Prototyping, Model Predictive Control oder die Fehlererkennung im Rahmen der Diagnose.

Die Toolchain wird in diesem Beitrag auf einen Anwendungsfall (hydro-mechanische Presse) angewendet, um die Methodik des Rapid Control Prototypings zur modellbasierten Reglerentwicklung anzuwenden. Die Validierung des Reglers erfolgt in diesem Beitrag über eine Hardware-In-The-Loop-Kopplung an einer virtuellen Regelstrecke der Presse, die ebenfalls in Modelica vorhanden ist. Da die Validierung des Reglers bisher nur an virtuellen Regelstrecken erfolgt ist, besteht eine Hauptbestrebung momentan darin, den generierten Regler an einer realen Anlage zur Regelung einzusetzen. Ein weiteres Thema für die Zukunft ist die Umsetzung von *Model Predictive Control* für den Einsatz auf Bosch Rexroth Steuerungen. Dazu ist der Simulationskern zusätzlich um die mathematischen Methoden zur Optimierung des Systemeingangs zu erweitern. Diese müssen schließlich so effizient implementiert werden, dass die Durchführung der Optimierung innerhalb der vorgegebenen Rechenzeit (Zykluszeit) abgeschlossen ist.

6 Literatur

[1] Fritzson, P.: *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley, 2014.

[2] Fritzson, P. et al.: *The open source Modelica project*. In: Proceedings of the 2nd International Modelica Conference, March 18-19, 2002, Oberpfaffenhofen, Germany.

[3] Worschech, N.; Mikelsons, L.: *A Toolchain for Real-Time Simulation using the OpenModelica Compiler*. In: Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany.

[4] Engels, E.; Gabler, T.: *Universelle Programmierschnittstelle für Motion-Logic Systeme*. In: Struktur, Funktionen und Anwendung in Forschung und Lehre, Tagungsband AALE 2012.

[5] Hairer, E.; Wanner, G.: *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer, 2002.

[6] Lang, J.; Verwer, J.: *ROS3P - an accurate third-order Rosenbrock solver designed for parabolic problems*. BIT Numerical Mathematics. (2001) vol. 41, num. 4, S. 731-738.

[7] Braun, W. et al.: *Fast simulation of fluid models with colored jacobians*. In: Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany.