Multidisziplinäre Modellierung mechatronischer Systeme mit Modelica für Echtzeitanwendungen

Dr.-Ing. Martin Otter, DLR Oberpfaffenhofen
Email: Martin.Otter@DLR.de

Abstract: Es wird eine Übersicht gegeben, wie unter Verwendung der objektorientierten Modellierungssprache Modelica und geeigneter Transformationsalgorithmen große, multidisziplinäre Systemmodelle erstellt und simuliert, sowie in Echtzeit-Anwendungen eingesetzt werden können.

1 Einführung

Zur Simulation von Modellen, die im wesentlichen durch Differentialgleichungen beschrieben werden, steht eine große Anzahl an fachgebietsspezifischen Simulationspaketen zur Verfügung. Beispiele sind SIMULINK [18] für *regelungstechnische Systeme*, ADAMS [1] oder SIMPACK [17] für *mechanische Systeme*, PSPICE [15] für *elektronische Schaltkreise*, Flowmaster [7] für *hydraulische Systeme*, SPEEDUP [19] für *chemische Prozesse*. Alle diese Programme sind jedoch auf *ein* Fachgebiet zugeschnitten, wobei die Modellierung von Komponenten anderer Disziplinen nicht möglich, oder nur in eingeschränktem Umfang durch Aufruf externer C- oder Fortran-Prozeduren möglich ist.

In diesem Artikel wird, basierend auf [13, 12, 6, 16], eine Übersicht über die Modellierungssprache *Modelica* gegeben, mit der "echte" multidisziplinäre Modelle komfortabel definiert und mit Hilfe eines geeigneten Übersetzers und einer Simulationsumgebung effizient simuliert werden können. Aus Benutzersicht werden in Modelica Modelle durch Objektdiagramme beschrieben. In Bild 1 ist eine Collage solcher Modelle aus unterschiedlichen Anwendungsgebieten beispielhaft zusammengestellt.

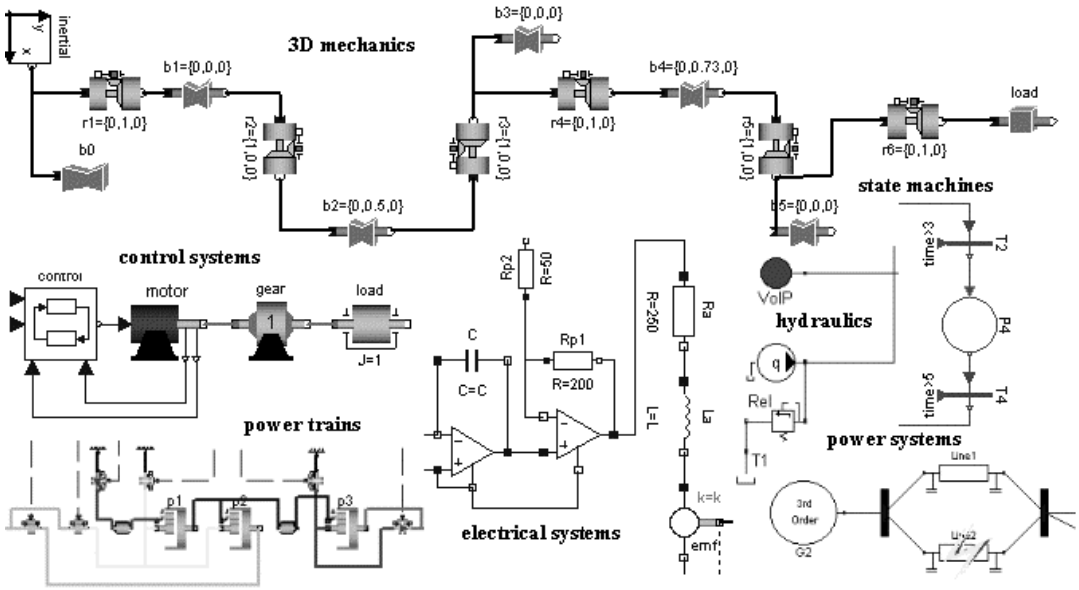

Bild 1: Modelica Objektdiagramme aus unterschiedlichen Fachgebieten.

Ein Objektdiagramm ist dabei eine Verallgemeinerung eines Blockschaltbildes, wobei die Schnittstellen von Komponenten nicht nur Signale sondern auch physikalische Schnittstellen enthalten können (wie mechanischer Flansch, elektrische Klemme). Eine Komponente eines Objektdiagramms ist wiederum hierarchisch aus Objektdiagrammen aufgebaut. Auf unterster Ebene werden Komponente durch Differential-, algebraische und diskrete Gleichungen beschrieben.

Modelica ist eine frei verfügbare Sprache die zusammen mit der ebenfalls frei verfügbaren Modelica Standard Bibliothek von mehr als 30 Mitarbeitern aus 9 Ländern in den letzten 4 Jahren entwickelt wurde (seit Anfang 2000 im Rahmen der gemeinnützigen Organisation "Modelica Association"). Initiiert und in den ersten 3 Jahren geleitet wurde diese Entwicklung von Hilding Elmqvist. Detaillierte Informationen zu Modelica sind im Web erhältlich:

Homepage:	`www.Modelica.org/`
Tutorial:	`www.Modelica.org/current/ModelicaTutorial14.pdf`
Sprachdefinition:	`www.Modelica.org/current/ModelicaSpecification14.pdf`
Modellbibliotheken:	`www.Modelica.org/library/library.html`
Veröffentlichungen:	`www.Modelica.org/workshop2000/proceedings`
Tools:	`www.Modelica.org/tools.shtml`

2 Grundlagen von Modelica

Wie im letzten Abschnitt skizziert, ist ein Modelica-Modell hierarchisch aus Objektdiagrammen aufgebaut, wobei auf unterster Ebene ein Modell durch mathematische Gleichungen beschrieben ist. Zur Verdeutlichung ist das Objektdiagramm eines einfachen Antriebsstrangmodells in Bild 2 zu sehen. Dieses Modell besteht aus einem elektrischen Motor, einem Getriebe, einer Last und dem Regelungssystem und wird folgendermassen in Modelica definiert (die Grafik eines Objektdiagramms wird innerhalb des Sprachelements **`annotation`** festgelegt. Aus Gründen der Übersichtlichkeit wird diese Definition jedoch in diesem Artikel nicht gezeigt).:

```
model MotorDrive
  PID        controller;
  Motor      motor;
  Gearbox    gear(n=100);
  Inertia    inertia(J=10);
equation
  connect(controller.outPort, motor.inPort);
  connect(controller.inPort , motor.outPort);
  connect(gear.flange_a     , motor.flange_b);
  connect(gear.flange_b     , inertia.flange_a);
end MotorDrive;
```

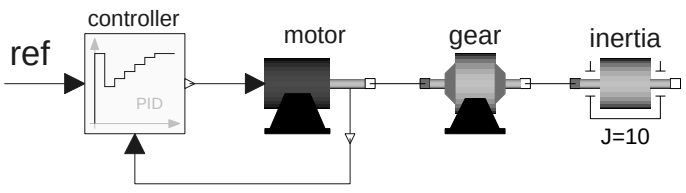

Bild 2: Objektdiagramm eines einfachen Antriebsstrangs.

Am Beginn des Modells werden 4 Komponenten deklariert. Die Anweisung `Gearbox gear(n=100)` definiert eine neue Komponente `gear` von der vorhandenen Modellklasse `Gearbox` und setzt die Übersetzung `n`, einen Parameter, auf den Wert 100. Im unteren Teil des Modells wird definiert, wie die Schnittstellen der Komponenten miteinander verschaltet werden. Das Objektdiagramm des Motors ist in Bild 3 zu sehen. Das Dreieck im linken Teil des Bildes charakterisiert einen Signaleingang, während das Viereck im rechten Teil des Bildes die Schittstelle eines mechanischen Flansches beschreibt. Schnittstellen werden mit der Connector-Klasse definiert, z.B.

```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;
```

```
connector Flange
  Angle       phi;
  flow Torque tau;
end Flange;
```

In einer Connector-Klasse werden alle Variablen der Schnittstelle beschrieben. Zum Beispiel werden in der Connector-Klasse Flange der absoluten Drehwinkel `phi` und das Schnittmoment im Flansch `tau` deklariert. Variablen können entweder vom Typ `Real, Boolean, Integer, String` oder eines davon abgeleiteten Typs sein, z.B.:

```
type Angle  = Real(quantity="Angle" , unit="rad", displayUnit="deg");
type Torque = Real(quantity="Torque", unit="N.m");
```

Eine Verbindungsgleichung der Form **`connect`**`(Pin1, Pin2)`, wobei `Pin1` und `Pin2` Instanzen der Connector-Klasse `Pin` sind, definiert eine physikalische Verbindung der beiden Klemmen und erzeugt implizit die Gleichungen `Pin1.v = Pin2.v` und `Pin1.i + Pin2.i = 0`. Eine ”Null-Summengleichung“ wird verwendet, wenn die entsprechenden Variablen in der Connector-Klasse mit dem Attribut **`flow`** gekennzeichnet sind. Auf unterster Ebene werden Modelle mit Gleichungen beschrieben. Zum Beispiel lautet die Modell-Klasse einer Drehträgheit:

```
model Inertia "Drehtraegheit"
  parameter Real J=1 "Wert der Traegheit";
  Flange a, b;
  Real   w "Drehzahl";
equation
         a.phi = b.phi;   // gleiche Winkel (Kommentar)
    der(a.phi) = w;
  J*der(w)     = a.tau + b.tau;
end Inertia;
```

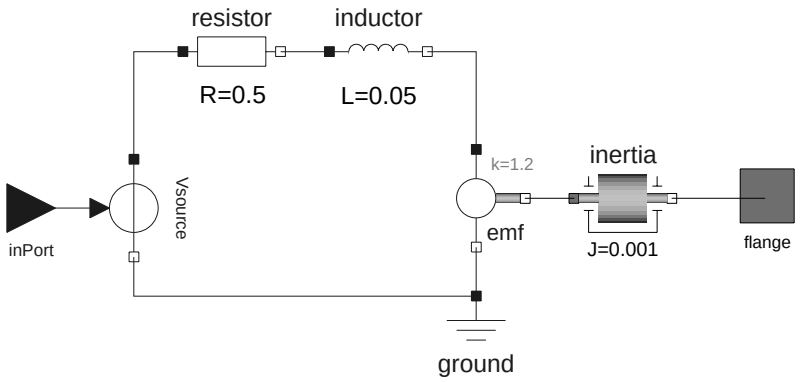

Bild 3: Objektdiagramm eines einfachen elektrischen Motors.

Eine Zeichenkette nach dem Modellnamen bzw. nach einer Deklaration enthält eine Kurzbeschreibung, die von Simulationsumgebungen speziell verwendet werden können, z.B. als Legende in einem Plot. Das Sprachelement **`parameter`** kennzeichnet eine Variable die während einer Simulation konstant ist. Der Operator **`der`**(...) definiert die Zeitableitung.

Neben den obigen Basis-Sprachelementen unterstützt Modelica ein- und mehrdimensionale Felder mit einer Matlab ähnlichen Syntax. Jede Instanz einer Klasse kann als Feldelement verwendet werden. Auf diese Weise können einfach diskretisierbare partielle Differentialgleichungen komfortabel definiert werden. Modelica hat eine sehr ausgefeilte Unterstützung von unstetigen und strukturvariablen Komponenten, wie Schalter, Lagerreibung, Kupplungen, Abtastsysteme etc. Hierbei werden die notwendigen Zeit- und Zustandsereignisse automatisch erzeugt. Schließlich hat Modelica ein mächtiges Bibliothekskonzept um sehr große Komponentenbibliotheken verwalten zu können und um z.B. eine Modell-Klasse im Dateisystem automatisch zu finden, wenn der hierarchische Modelica Name dieser Klasse bekannt ist.

3 Simulation von Modelica Modellen

Die Modelica Sprache hat einen relativ geringen Sprachumfang, z.B. verglichen mit der ähnlich mächtigen Modellierungssprache VHDL-AMS [21]. Hauptgrund ist, dass es nur ein einziges Basis-Strukturierungselement gibt – die Klasse einer Komponente – und alle sonstigen Strukturierungselemente der Sprache, wie **`model`**, **`block`**, **`function`**, **`connector`**, **`package`**, **`record`**, **`type`**, Spezialisierungen einer Klasse sind. Es ist deswegen relativ einfach einen Modelica Übersetzer zu realisieren, der ein Modelica-Modell in eine Menge von Differential-, algebraischen und diskreten Gleichungen überführt. Ein frei verfügbarer Modelica Parser für einen solchen Zweck steht z.B. auf der Modelica Homepage zur Verfügung (`http://www.Modelica.org/tools/parser/Parser.shtml`).

Eine direkte numerische Lösung dieses Ausgangsgleichungssystems ist jedoch in den meisten Fällen höchst *uneffizient* und *unzuverlässig*. Die Modelica Sprache wurde deswegen so entworfen, dass symbolische Transformationsalgorithmen relativ einfach angewandt werden können, um das Ausgangsgleichungssystem in eine Form zu transformieren, die mit Standard-Integrationsverfahren besser gelöst werden kann. Die wesentlichen Schritte bei einer solchen Transformation werden im folgenden näher erläutert.

3.1 Algebraische Schleifen

Ein Modelica Übersetzer stellt das Ausgangsgleichungssystem auf, in dem die Gleichungen von allen Komponenten und die Gleichungen auf Grund von Verbindungen zu einem Gesamtgleichungssystem zusammengefaßt werden. Bei rein kontinuierlichen Modellen führt dies auf ein Gleichungssystem der Form:

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) \tag{1}$$

wobei im Vektor $\mathbf{x}(t)$ alle Variablen zusammengefasst sind, die abgeleitet in Modellen auftreten (d.h. $\dot{\mathbf{x}}$ ist vorhanden), $\mathbf{y}(t)$ ist ein Vektor von algebraischen Variablen und t ist die Zeit als unabhängige Variable. Auf Grund der Konstruktion der Modelle gibt es auch schon bei relativ einfachen Systemen viele Gleichungen, wobei jedoch nur wenige Unbekannte in einer Gleichung enthalten sind. In einem ersten Schritt wird deswegen das Gleichungssystem (1) durch Umsortieren der Gleichungen und durch

Umstellen von Ausdrücken in einer Gleichung symbolisch in die folgende Form transformiert:

$$\begin{bmatrix} \mathbf{0} \\ \dot{\mathbf{x}}^e \\ \mathbf{y}^e \end{bmatrix} = \begin{bmatrix} \mathbf{f}^r\ (\dot{\mathbf{x}}^i, \mathbf{x}, \mathbf{y}^i, t) \\ \mathbf{f}^x\ (\dot{\mathbf{x}}^i, \mathbf{x}, \mathbf{y}^i, t) \\ \mathbf{f}^y\ (\dot{\mathbf{x}}^i, \mathbf{x}, \mathbf{y}^i, t) \end{bmatrix} \qquad \begin{bmatrix} \mathbf{x}^i \\ \mathbf{x}^e \end{bmatrix} = \mathbf{P}^x\mathbf{x}, \quad \begin{bmatrix} \mathbf{y}^i \\ \mathbf{y}^e \end{bmatrix} = \mathbf{P}^y\mathbf{y} \tag{2}$$

Das heißt, das Ausgangsgleichungssystem wird in einen *implizit* und in einen *explizit* lösbaren Teil umgeformt. Hierbei werden die Vektoren **x** und **y** entsprechend dieser Aufspaltung mit Hilfe der Permutationsmatrizen $\mathbf{P}^x, \mathbf{P}^y$ in einen impliziten (Index = i) und in einen expliziten (Index = e) Teilvektor umgeordnet.

Praktisch wird versucht das implizite Teilsystem $\mathbf{f}^r(\ldots)$ in möglichst kleine, voneinander unabhängige algebraische Schleifen aufzuteilen. Hierfür stehen effiziente Algorithmen zur Verfügung, siehe z.B. [4]. Diese Transformation wird auch als BLT-Transformation bezeichnet (BLT = Block Lower Triangular). Fast alle physikalischen Modelle führen auf algebraische Gleichungssysteme. Diese sind nach einer Sortierung meist immer noch unnötig groß. Durch "intelligente Variablensubstitution" mit dem Tearing-Verfahren, siehe z.B. [12], kann die Größe der impliziten Teilsysteme in der Regel noch einmal stark reduziert werden.

Für eine numerische Lösung von (2) mit Standard-Integrationsverfahren können nun *alle* explizit auflösbaren algebraischen Variablen $\mathbf{y}^e$ vor dem Integrator „versteckt" werden, da diese Variablen aus den anderen Variablen explizit berechnet werden. Aus Sicht des Integrators hat sich damit die Systemordnung erniedrigt. In Modelica-Modellen gibt es meist sehr viele algebraische Variablen, da z.B. in der Regel alle Schnittstellen-Variablen rein algebraisch sind, so daß die Ordnungsreduktion oft beträchtlich ist.

Es werden nun (vorzugsweise) Integrationsverfahren eingesetzt die implizit gegebene Gleichungsysteme lösen können, z.B. DASSL [2]. Wenn das nicht möglich ist, z.B. wenn das sortierte Modell in eine Simulationsumgebung wie SIMULINK [18] eingebunden werden soll, die nur die Zustandsform unterstützt, werden die impliziten Gleichungsteile im Modell mit entsprechenden Verfahren zur Lösung von linearen oder nicht-linearen algebraischen Gleichungssystemen numerisch gelöst.

3.2 Gekoppelte Speicherelemente

Beim Zusammenschalten von Modelica Komponentenmodellen kann es vorkommen, dass im verschalteten System die beschreibenden Variablen **x** der Komponenten nicht mehr unabhängig voneinander sind. Mathematisch drückt sich dies dadurch aus, dass die Jacobimatrix von (1) bezüglich $\dot{\mathbf{x}}$ und **y** singulär ist. Ein typisches Beispiel ist in Bild 4 zu sehen, bei dem zwei Rotationsträgheiten direkt mit einander gekoppelt sind. Die sortierten und vereinfachten Gleichungen dieses Modells lauten.

$$\dot{\varphi}_1 = \omega_1 \tag{3a}$$
$$J_1 \cdot \dot{\omega}_1 = \tau - \tau_2 \tag{3b}$$
$$\dot{\varphi}_2 = \omega_2 \tag{3c}$$
$$J_2 \cdot \dot{\omega}_2 = \tau_2 \tag{3d}$$
$$\varphi_1 = \varphi_2 \tag{3e}$$

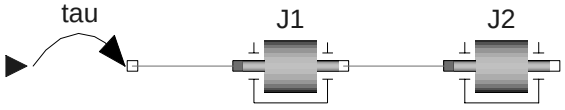

Bild 4: Zwei gekoppelte Trägheiten.

Hierbei sind φ_i der absolute Drehwinkel und ω_i die absolute Drehzahl der Trägheit i, τ ist das bekannte, externe Antriebsmoment von Trägheit 1 und τ_2 ist das Schnittmoment zwischen den beiden Trägheiten. Dies sind 4 Differentialgleichungen und 1 algebraische Gleichung zur Berechnung der 5 Unbekannten $\varphi_1(t), \varphi_2(t), \omega_1(t), \omega_2(t), \tau_2(t)$. Eine direkte numerische Lösung von (3) ist problematisch. Dies kann schon daran erkannt werden, dass z.B. mit

$$\varphi_1(t_0) = \dot{\varphi}_1(t_0) = \dot{\omega}_1(t_0) = \varphi_2(t_0) = \boldsymbol{\omega}_1(t_0) = \mathbf{0},$$
$$\tau_2(t_0) = \tau(t_0), \quad \dot{\varphi}_2(t_0) = \boldsymbol{\omega}_2(t_0) = \mathbf{1}, \quad \dot{\omega}_2(t_0) = \tau(t_0)/J_2$$

Anfangsbedingungen vorgegeben werden können, die zwar *alle* Gleichungen (3) des Modells erfüllen, aber offensichtlich widersprüchlich sind, da $\omega_1(t_0) = 0$ und $\omega_2(t_0) = 1$ auf Grund der starren Kopplung nicht gleichzeitig erfüllt sein können.

Lineare Systeme der Form (3) können nach einer geeigneten Variablentransformation in der Regel zuverlässig gelöst werden. Bei nichtlinearen Differential- und algebraischen Gleichungen gibt es größere Schwierigkeiten. Eine robuste Lösung besteht darin, (3e) zweimal, sowie (3a) und (3c) einmal zu differenzieren und mit Ausnahme von $\dot{\varphi}_1, \dot{\omega}_1$, alle anderen Ableitungen als rein algebraische Variablen aufzufassen. Das so entstehende Gleichungssystem kann dann in die Zustandsform mit den Zustandsgrößen φ_1, ω_1 transformiert werden. Man beachte, dass in diesem Fall die oben gewählten Anfangsbedingungen das neue Gleichungssystem nicht mehr erfüllen und somit vom Simulator als "nicht konsistent" erkannt werden können. Dieses Verfahren wird als "Dummy Derivative" Methode bezeichnet. Welche Gleichungen wie oft differenziert werden müssen, kann effizient und zuverlässig mit dem Algorithmus von Pantelides [14] ermittelt werden. Die Auswahl der Zustände, bzw. die Festlegung welche Ableitungen als rein algebraische Variablen zu betrachten sind, kann zum Teil statisch beim Übersetzen des Modells ermittelt werden, bzw. kann zum Teil auch erst dynamisch während einer Simulation bestimmt werden. Auch hierfür stehen geeignete Algorithmen zur Verfügung [9, 10].

3.3 Reell-Boolsche Gleichungssysteme

Um die Effizienz (insbesondere für die Echtzeitsimulation) zu steigern, sowie um die Modellidentifikation zu vereinfachen, werden idealisierte Modelle eingesetzt. Beispiele sind ideale Dioden- und Thyristormodelle, aber auch Reibmodelle bei denen im Haften keine Relativbewegung auftritt. Solche Modelle können in Modelica relativ problemlos formuliert werden, führen jedoch auf strukturvariable Gleichungssysteme, bei denen neben reellen Größen auch Bool'sche bzw. Integer-Variablen als Unbekannte auftreten. Damit müssen die Standardverfahren zur Lösung von rein reellen Gleichungssystemen entsprechend erweitert werden, siehe [11]. Generell sind solche gemischt reell/Bool'schen Gleichungssysteme zu lösen, wenn eine if-Anweisung in einer algebraischen Schleife auftritt und die Umschaltbedingung der if-Anweisung von den Unbekannten der Schleife abhängt. Diese Eigenschaft wird an einem einfachen Beispiel kurz erläutert:

In Bild 5 ist ein Teil eines Antriebsstrangs abgebildet. Die Verluste auf Grund von Zahnflankenreibung werden hier näherungsweise mit einem konstanten Wirkungsgrad beschrieben:

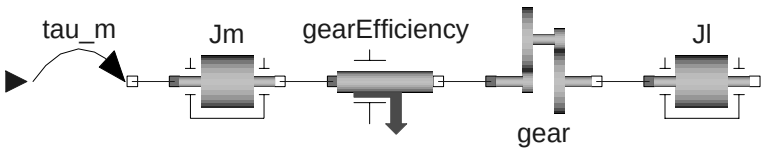

Bild 5: Modelica Modell eines Antriebsstrangs mit Getriebeverlusten.

$$\tau_2 = \bar{\eta} \cdot i \cdot \tau_1 \qquad \bar{\eta} = \begin{cases} \eta & für \quad \tau_1 \cdot \omega_1 > 0 \\ 1/\eta & für \quad \tau_1 \cdot \omega_1 < 0 \\ \text{undefiniert} & für \quad \tau_1 \cdot \omega_1 = 0 \end{cases} \qquad 0 < \eta \leq 1 \qquad (4)$$

hierbei ist τ_1 das Antriebsmoment des Getriebes, i die Getriebeübersetzung, τ_2 das Lastmoment und $\bar{\eta}$ der Getriebewirkungsgrad. Der Wirkungsgrad $\bar{\eta}$ hängt vom Energiefluss $(\tau_1 \cdot \omega_1)$ ab. Bei verschwindendem Energiefluss wechselt die Kontaktfläche der Zähne von der Vorder- zu der Rückseite eines Zahns bzw. umgekehrt. In dieser kurzen Phase wird kein Moment übertragen. Zur Vereinfachung wird dieser komplexe Übergang vernachlässigt. Dieses einfache Modell von Getriebeverlusten führt auf die folgenden (vereinfachten und sortierten) Gleichungen des Antriebsstrangs von Bild 5:

$$\begin{aligned} b &= \tau_1 \cdot \omega_l > 0 \\ \tau_2 &= (\textbf{if } b \textbf{ then } \eta \textbf{ else } 1/\eta) \cdot \tau_1 \\ J_m \cdot i \cdot \dot{\omega}_l &= \tau_m - \tau_1 \\ J_l \cdot \dot{\omega}_l &= i \cdot \tau_2 \end{aligned}$$

wobei J_m die Trägheit des Motors, J_l die Trägheit der Last, ω_l die Absolutdrehzahl der Last und $\tau_m(t)$ das (bekannte) antreibende Moment des Motors ist. In Modelica lösen alle Wert-Änderungen von logischen Ausdrücke, wie $\tau_1 \cdot \omega_l > 0$, standardmäßig ein Ereignis aus, da hierbei potentiell eine Unstetigkeit eingeführt wird. Nach Detektion des Ereignisses wird die Integration angehalten, die Änderung des logischen Ausdrucks (und damit z.B. ein Wechsel des if-Zweiges) durchgeführt und die Integration neu gestartet. Bei dieser Vorgehensweise ist die Bool'sche Variable b während der Integration bekannt (= Wert vom letzten Ereignis). An einem Ereignispunkt ist b aber unbekannt. Dann sind die obigen 4 Gleichungen ein gekoppeltes System von 4 Gleichungen in 3 reellen Unbekannten $\dot{\omega}_l$, τ_1, τ_2 und 1 Boolsch'en Unbekannten b, das nicht mehr mit Standardalgorithmen gelöst werden kann. In [11] werden benötigte Erweiterungen dieser Algorithmen skizziert.

3.4 Simulation von Modelica Modellen mit Dymola

Zusammengefasst kann festgehalten werden, dass die Transformation des Ausgangsgleichungssystems in eine numerisch robust auszuwertende Form aufwendig ist und viel Know-How erfordert. Aus diesem Grunde gibt es zur Zeit keine frei verfügbare Modelica Simulationsumgebung.

Seit Juni 2000 wird nahezu die gesamte Modelica Sprache von der kommerziellen Modellierungs- und Simulationsumgebung Dymola [5] unterstützt. Dymola enthält alle oben skizzierten Algorithmen, sowie eine ganze Reihe weiterer Algorithmen um die Effizienz und Robustheit zu steigern. Zum Beispiel, werden die Gleichungen so umsortiert, dass alle *konstanten Gleichungen* vor Beginn der Simulation nur einmal und alle *"Ausgangsgleichungen"* nur an Kommunikationszeitpunkten ausgewertet werden. Dymola kann sehr grosse Systeme behandeln (siehe z.B. [3, 20]) und kann die transformierten Gleichungen auch in Form von SIMULINK S-Functions als C-Code ausgeben, der einfach in SIMULINK [18] als Ein-/Ausgangsblock des Modells verwendet werden kann.

4 Echtzeit-Simulation von Modelica Modellen

Bei Echtzeit-Simulationen besteht eine Schwierigkeit darin, ein Modell mit einer festen Schrittweite zu integrieren, wobei für jeden Schritt eine maximal vorgegebene Zeitdauer zur Verfügung steht. Üblicher-

weise wird entweder das explizite oder das linear-implizite Euler-Verfahren eingesetzt. Die Vorgehensweise wird an einem System in Zustandsform demonstriert:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t), \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{5}$$

Beim expliziten Euler-Verfahren wird die Ableitung des Zustandsvektors durch eine Vorwärtsdifferenz approximiert:

$$\dot{\mathbf{x}}(t_n) = \dot{\mathbf{x}}_n \approx \frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{h} \qquad \Rightarrow \qquad \mathbf{x}_{n+1} := \mathbf{x}_n + h \cdot \mathbf{f}(\mathbf{x}_n, t_n) \tag{6}$$

Hierbei ist $\mathbf{x}_{n+1} = \mathbf{x}(t_{n+1})$ der unbekannte Wert von $\mathbf{x}$ zum neuen Zeitpunkt $t_{n+1} = t_n + h$ mit der Schrittweite h. Die jeweils neuen Werte von $\mathbf{x}$ können problemlos über eine Vorwärtsrekursion bestimmt werden.

Beim impliziten Euler-Verfahren wird die Ableitung des Zustandsvektors durch eine Rückwärtsdifferenz approximiert:

$$\dot{\mathbf{x}}(t_n) = \dot{\mathbf{x}}_n \approx \frac{\mathbf{x}_n - \mathbf{x}_{n-1}}{h} \qquad \Rightarrow \qquad \mathbf{x}_{n+1} := \mathbf{x}_n + h \cdot \mathbf{f}(\mathbf{x}_{n+1}, t_{n+1}) \tag{7}$$

Dies führt auf ein nichtlineares Gleichungssystem zur Bestimmung der Unbekannten $\mathbf{x}_{n+1}$. Da dieses Gleichungssystem nur iterativ gelöst werden kann, ist das Verfahren für eine Echtzeitsimulation nicht geeignet. Die Lösung kann jedoch durch Linearisierung von $\mathbf{f}$ um den vorherigen Punkt approximiert werden ($\mathbf{E}$ ist die Einheitsmatrix):

$$\left(\mathbf{E} - h \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_n, t_n)\right) \cdot \Delta\mathbf{x}_{n+1} = h \cdot \mathbf{f}(\mathbf{x}_n, t_n), \qquad \mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_{n+1} \tag{8}$$

Hierfür muß in jedem Zeitschritt die Jacobimatrix $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ berechnet und das lineare Gleichungssystem (8) gelöst werden.

Das explizite Euler-Verfahren hat den Nachteil, dass steife Systemteile nur mit sehr kleinen Schrittweiten integriert werden können, da das Verfahren sonst eine qualitativ völlig falsche, instabile Lösung liefert. Das linear-implizite Euler-Verfahren kann in diesem Fall mit sehr viel größeren Schrittweiten arbeiten. Dafür muß in jedem Zeitschritt aber die Jacobimatrix berechnet und ein lineares Gleichungssystem gelöst werden.

Um die Vorteile beider Verfahren zu kombinieren, wird in [16, 6] folgende Vorgehensweise vorgeschlagen:

1. Die Zustände werden in "schnelle" Zustände $\mathbf{x}^s$ und "langsame" Zustände $\mathbf{x}^l$ aufgeteilt.
2. Die Ableitungen der "schnellen" Zustände werden mit einer Rückwärts- und die der "langsamen" Zustände mit einer Vorwärts-Diskretisierung approximiert.
3. Die Diskretisierungsformeln werden in das Model eingesetzt und die Standardalgorithmen (wie Sortieren + Tearing von Gleichungssystemen) werden zur Vereinfachung des Gleichungssystems verwendet. Bei auftretenden nichtlinearen Gleichungssystemen wird nur eine Iteration des Newton-Verfahrens durchgeführt.

Praktisch wird dieses Vorgehen einfach dadurch realisiert, in dem alle abgeleiteten Größen als rein algebraische Variablen aufgefasst werden, die Diskretisierungsformeln der Zustände zum Gleichungssystem

hinzugefügt werden und die skizzierten Standard-Algorithmen auf das neu entstehende Gleichungssystem

$$\begin{aligned} \mathbf{0} &= \mathbf{f}(\dot{\mathbf{x}}_n, \mathbf{x}_n, \mathbf{y}_n, t_n) \\ \mathbf{x}_n^s &= \mathbf{x}_{n-1}^s + h \cdot \dot{\mathbf{x}}_n^s \\ \mathbf{x}_n^l &= \mathbf{x}_{n-1}^l + h \cdot \dot{\mathbf{x}}_{n-1}^l \end{aligned}$$

angewandt werden. Bei dieser Vorgehensweise kann die Schrittweite h auch zu Null gesetzt werden, um z.B. das Ausgangsgleichungssystem wieder zu erhalten, das in der Regel bei der Initialisierung benötigt wird.

Bei geschickter Aufteilung des Zustandsvektors können damit die Vorteile beider Verfahren kombiniert werden: Durch Vorwärtsdifferenzen wird das große Gleichungssystem in kleinere, oft völlig entkoppelte Subysteme aufgeteilt. Durch Rückwärtsdifferenzen können auch bei steifen Systemen große Schrittweiten verwendet werden. In [16] wird gezeigt, dass die Simulation mit diesem Vorgehen bei einem großen Gesamtmodell eines Roboters mit ca. 80 Zuständen und 1000 algebraischen Gleichungen um rund den Faktor 15 schneller ist, als bei Verwendung eines rein expliziten oder rein impliziten Euler Verfahrens und damit auf heutigen PC's leicht in Echtzeit rechenbar ist.

5 Zusammenfassung und Ausblick

Es wurde ein Überblick über die objektorientierte Modellierungssprache Modelica gegeben und die Basis-Algorithmen skizziert, um ein Modelica Modell so zu transformieren, dass das entstehende Gleichunsystem mit Standardverfahren effizient und robust gelöst werden kann.

In [3, 20] wird das wohl größte bisher erstellte und mit Dymola simulierte multidisziplinäre Modell eines Gesamtfahrzeugs beschrieben. Dieses besteht aus einem detaillierten Fahrzeugdynamik-Modell (60 Gelenke, 70 Körper), dem Motor inklusive einem Verbrennungsmodell, dem Antriebsstrang mit detailliert modelliertem Automatik-Getriebe, sowie einem Teil der hydraulischen Ansteuerung des Automatikgetriebes. Die symbolische Transformation dieses Modelica-Modells mit Dymola führt auf ein System von rund 25000 nichttrivialen algebraischen Gleichungen und 320 Differentialgleichungen.

Modelica und die Modelica Simulationsumgebung Dymola werden insbesondere auch für Echtzeitanwendungen eingesetzt: Zum Beispiel bei verschiedenen KFZ-Firmen zur Hardware-in-the-Loop Simulation von Automatikgetrieben, sowie zur Generierung von inversen Dynamikmodellen für "embedded control" bei Robotersteuerungen und bei neuartigen, im Flugversuch schon erfolgreich getesteten, Autopilot-Reglern zum automatischen Landen von Verkehrsflugzeugen [8].

Literatur

[1] ADAMS: Homepage: http://www.adams.com/

[2] *Brenan K.E., Campbell S.L.* und *Petzold L.R*: Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations. Elsevier Science Publishers, 2. Auflage, 1996.

[3] *Bowles P., Tiller M., Elmqvist H., Brück D., Mattsson S.E., Möller A., Olsson H.* und *Otter M.:* Feasibility of Detailed Vehicle Modeling. SAE 2001 World Congress, Detroit, U.S.A., Paper Nr. 2001-01-0334, 2001.

[4] *Duff I. S., Erisman A. M.* und *Reid J. K.:* Direct Methods for Sparse Matrices. Oxford Science Publication, 1986.

[5] *Dymola:* Homepage: http://www.dynasim.se/

[6] *Dynasim: Dymola User's Manual, Version 4.1b, 2001.*

[7] Flowmaster: Homepage: http://www.flowmaster.com/

[8] *Joos H.-D., Looye G.* und *Willemsen D.:* Application of Optimization-based Design Process for Robust Autoland Control Laws. AIAA, 2001.

[9] *Mattsson S.E.* und *Söderlind G.:* Index reduction in differential-algebraic equations using dummy derivatives. SIAM Journal of Scientific and Statistical Computing, Vol. 14, S. 677–692, 1993.

[10] *Mattsson S.E., Olsson H.* und *Elmqvist H.:* Dynamic Selection of States in Dymola. Modelica Workshop 2000, S. 61-67, 23.-24. Oct. Lund, Schweden, 2000 (http://www.Modelica.org/modelica2000/proceedings.html).

[11] *Otter M., Elmqvist H.* und *Mattsson S.E.:* Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. CACSD'99, 22.-26. August, Hawaii, U.S.A., 1999.

[12] *Otter M.:* Objektorientierte Modellierung Physikalischer Systeme, Teil 4, Transformationsalgorithmen. Automatisierungstechnik at4/99, S. A13-A16, April 2000.

[13] *Otter M.* und *Elmqvist H.:* Modelica – Language, Libraries, Tools, Workshop and EU-Project RealSim. Simulation News Europe, S. 3-8, Dez. 2000.

[14] *Pantelides C.:* The Consistent Initialization of Differential-Algebraic Systems. SIAM Journal of Scientific and Statistical Computing, S. 213–231, 1988.

[15] PSPICE: Homepage: http://www.microsim.com/

[16] *Schiela A.* und *Olsson H.:* Mixed-mode Integration for Real-time Simulation. Modelica Workshop 2000, 23.-24. Oct., Lund, Schweden, 2000. (http://www.Modelica.org/modelica2000/proceedings.html)

[17] SIMPACK: Homepage: http://www.simpack.de/

[18] SIMULINK: Homepage: http://www.Mathworks.com/

[19] SPEEDUP: http://www.aspentec.com/pspsd/speedup.htm

[20] *Tiller M., Bowles P., Elmqvist H., Brück D., Mattsson S.E., Möller A., Olsson H.* und *Otter M.:* Detailed Vehicle Powertrain Modeling in Modelica. Modelica Workshop 2000, S. 169-178, 23.-24. Oct., Lund, Schweden, 2000 (http://www.Modelica.org/modelica2000/proceedings.html).

[21] *VHDL-AMS:* IEEE Standard VHDL Analog and Mixed-Signal Extensions. IEEE Std. 1076.1-1999 (http://standards.ieee.org/catalog/design.html#1076.1-1999).