Objektorientierte Modellierung Physikalischer Systeme, Teil 1

Martin Otter, DLR Oberpfaffenhofen

Dr.-Ing. Martin Otter ist wissenschaftlicher Mitarbeiter beim DLR Institut für Robotik und Systemdynamik, leitet dort das Teilprogramm "Robotsystemdynamik" und ist Lehrbeauftragter an der TU München. Hauptarbeitsgebiete: Steuerung und Regelung von Industrierobotern, Antriebsstrangmodellierung für Echtzeitanwendungen, objektorientierte Modellierung.

Adresse: Deutsches Zentrum für Luft- und Raumfahrt, Institut für Robotik und Systemdynamik, Postfach 1116, D-82230 Weßling, Tel.: 08153/28-2473, Email: Martin.Otter@DLR.de

In einer Folge von Artikeln wird eine Übersicht über die Methodik der objektorientierten Modellierung physikalischer Systeme gegeben. Diese Modellierungstechnik eignet sich insbesondere für die multidisziplinäre Systemsimulation und kann als eine Verallgemeinerung der Blockdiagramm-Modellierung angesehen werden. In Teil 1–7 werden die Grundlagen an Hand einfacher Beispiele besprochen, um kontinuierliche Systeme mit unstetigen und strukturvariablen Komponenten deklarativ zu beschreiben und in die Zustandsform zu transformieren. In weiteren Folgen werden dann komplexere Anwendungen vorgestellt, wie die Modellierung von 3-dimensionalen mechanischen Systemen und Thermo-Fluid Rohrströmungen.

Objectoriented Modeling of Physical Systems

In a series of articles an overview is given for the object-oriented modeling of physical systems. This technique is especially suited for multidisciplinary system simulations and can be seen as a generalization of block diagram modeling. In part 1–7 the fundamentals are explained at hand of simple examples to model continuous systems with discontinuous and variable structure components in a declarative way and to transform such models into state space form. Afterwards, more complex applications are discussed such as the modelling of 3-dimensional mechanical systems and thermo-fluid pipe flow.

1 Übersicht

Zur Auslegung, Analyse und Test von dynamischen Systemen wird die Modellierung und Simulation immer wichtiger. Hierfür gibt es eine Vielzahl von Softwaresystemen, die auf bekannten Prinzipien und Algorithmen beruhen. Beispiele hierfür sind SIMULINK [8] oder SystemBuild [10] für *regelungstechnische Systeme*, ADAMS [1], SIMPACK [7] oder WorkingModel [11] für *mechanische Systeme*, SPICE und SPICE-Derivate wie PSPICE [6] für *elektronische Schaltkreise*, Flowmaster [5] für *hydraulische Systeme*, SPEEDUP [9] für *chemische Prozesse*. Diese Art von Programmen 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. Zum Beispiel sind Blockschaltbild-Simulatoren wie SIMULINK ungeeignet, um 3-dimensionale mechanische Systeme zu modellieren, während dies mit Mehrkörperprogrammen wie SIMPACK sehr einfach geht. Andererseits ist es nicht praktikabel, mit Mehrkörperprogrammen Blockschaltbilder zu simulieren.

Seit Ende der siebziger Jahre wird an Verfahren gearbeitet, die „echte" *multidisziplinäre* Modellierung und Simulation *komplexer* und *großer* dynamischer Systeme erlauben sollen. Diese Verfahren werden im folgenden unter dem Begriff *objektorientierte Modellierung* zusammengefaßt. Die grundlegende Methodik wurde von Hilding Elmqvist am Lund Institute of Technology in Schweden entwickelt [4]. Es gibt viele Varianten. In den letzten Jahren wurde die Theorie teilweise zu einem Abschluß gebracht und erste Softwaresysteme sind für den praktischen Einsatz verfügbar. Veröffentlichungen zu diesem Gebiet sind, mit Ausnahme einer kurzen Beschreibung in dem Standardwerk von Cellier [2], in Konferenz- und Zeitschriftenartikeln, sowie in Dissertationen verstreut. Deswegen wird in einer Folge von Artikeln eine *zusammenfassende Darstellung* dieser vielversprechenden Methodik gegeben, die insbesondere für den praktischen Einsatz der Modellierung und Simulation in Zukunft eine hohe Bedeutung haben wird.

1.1 Objektdiagramme

Die grundlegende Idee der objektorientierten Modellierung ist einfach. Schwierigkeiten gibt es in Detailproblemen, die in den Folgeartikeln diskutiert werden.

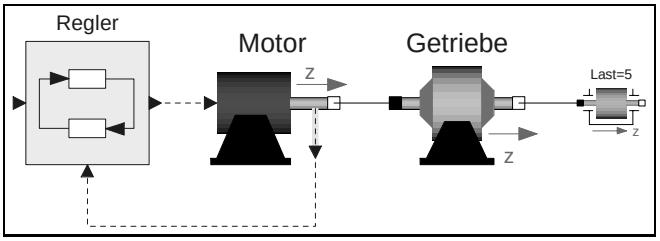


Bild 1: Objektdiagramm eines Antriebsstrangs.

Aus *Benutzersicht* wird ein Modell durch ein *Objektdiagramm* beschrieben. Ein typisches Objektdiagramm eines Antriebsstrangs, bestehend aus Regler, Elektromotor, Getriebe und Last ist in Bild 1 zu sehen, einem Bildschirmabzug des Programms Dymola [3]. Die durchgezogenen Verbindungslinien kennzeichnen starre, mechanische Verbindungen zwischen den Flanschen von Wellen. Die hier gestrichelt gezeichneten Linien sind Signalflüsse. Objektdiagramme sind hierarchisch aufgebaut: Der Regler wird durch ein Blockschaltbild, einem Spezialfall eines Objektdiagramms, modelliert (hier nicht dargestellt). Das elektrische Schaltbild des Motors, siehe Bild 2, ist ebenfalls ein Objektdiagramm. Es besteht aus Stromrichter, Widerstand, Induktivität,

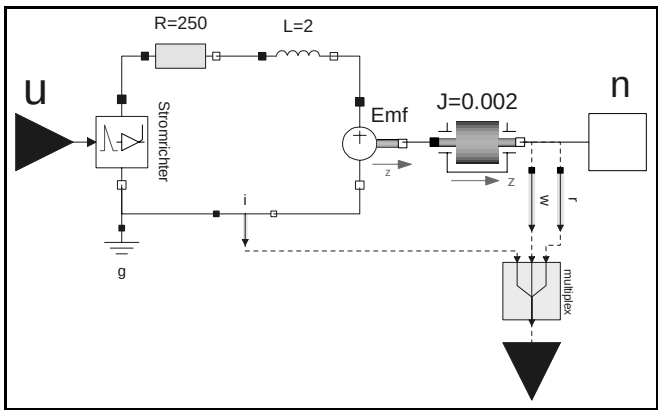


Bild 2: Objektdiagramm der Motor-Komponente.

Elektro-Motorischer Kraft und der Motorträgheit. Die durchgezogenen Linien im linken Teil kennzeichnen elektrische Leitungen. Das linke Dreieck ist der Signal-Eingang für die Sollspannung, das rechte Dreieck ist der Signal-Ausgangsvektor von gemessenem Strom, Drehzahl und Drehwinkel. Das Rechteck am rechten Rand stellt den mechanischen Flansch des Motors dar. Das Getriebemodell ist in Bild 3 zu sehen. Es besteht aus einer idealen Untersetzung, Lose, Steifigkeit und Dämpfung, sowie Lagerreibung. Die Verbindungslinien stellen wiederum starre mechanische Verbindungen dar.

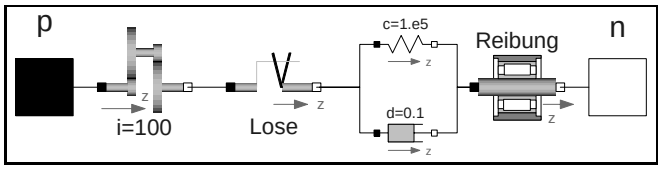


Bild 3: Objektdiagramm der Getriebe-Komponente

Objektdiagramme sind Verallgemeinerungen von Blockdiagrammen und bestehen aus den folgenden Teilen:

1. Einer *grafischen Darstellung* der physikalischen Komponente.
2. *Schnittstellen*, mit denen eine Komponente mit anderen Bauteilen verbunden werden kann.
3. *Gerichtete* oder *ungerichtete Verbindungslinien* zwischen Schnittstellen charakterisieren die physikalischen Verbindungen, z.B. elektrische oder hydraulische Leitungen, bzw. mechanisch starre Verbindungen, aber auch *Signalflüsse*.
4. Eine Komponente wird *unabhängig* von der *Umgebung* definiert, in der sie eingesetzt wird. Zur Beschreibung werden nur die Variablen der Schnittstellen sowie lokale Variablen benutzt. Es ist in der Regel nicht vorab bekannt, ob eine Schnittstellenvariable eine Ein- oder Ausgangsgröße ist.
5. Eine Komponente ist wiederum *hierarchisch* aus einer Verschaltung von Komponenten aufgebaut oder wird durch *algebraische* Gleichungen bzw. durch *Differentialgleichungen* beschrieben.

Basierend auf einem Objektdiagramm erstellt ein objektorientiertes Modellierungssystem ein großes, dünnbesetztes differential-algebraisches Gleichungssystem (abgekürzt DAE, für Differential-Algebraic Equation system). Hierbei werden die *lokalen Gleichungen* aller Komponenten, sowie die Gleichungen auf Grund von *Komponenten-Verbindungen*, zu einem Gesamtgleichungssystem zusammengefaßt. Die direkte numerische Lösung eines solchen Gleichungssystems ist für eine große Klasse von Modellen uneffizient, selbst wenn gute „Sparse-Matrix“-Verfahren eingesetzt werden. Der wesentliche Schritt besteht deswegen darin, diese DAE mittels symbolischer Transformationsalgorithmen in eine sortierte DAE oder in eine Zustandsform umzuformen, die effizienter gelöst werden kann. Die zur Verfügung stehenden Algorithmen sind sehr leistungsfähig. Z.B. kann eine DAE mit mehr als 10000 Gleichungen auf einem PC innerhalb von wenigen Sekunden in die Zustandsform überführt werden. Schließlich werden die üblichen numerischen Integrationsverfahren eingesetzt, um die erhaltene Zustandsform bzw. die sortierte DAE zu lösen.

Es stellt sich die Frage, wie ein allgemeines objektorientiertes Modellierungssystem die korrekten *Gleichungen* für eine (abstrahierte) *Verbindung* zwischen Bauteilen erstellen kann? Es zeigt sich, daß in allen Fachgebieten nur zwei Arten von Verbindungsgleichungen auftreten (siehe auch Teil 3 dieser Serie):

1. Verbundene Variable haben *denselben Wert*, z.B. elektrisches Potential, Weg, Geschwindigkeit, Druck, Dichte. Diese Variablen werden als *Potential*-Variablen bezeichnet.
2. Die *Summe* der Variablen, welche miteinander verbunden sind, *verschwindet*, z.B. bei elektrischem Strom, Kraft, Moment, Wärmefluß, Volumenstrom, Massenstrom. Diese Variablen werden als *Fluß*-Variablen bezeichnet. Hier ist es wichtig, daß an allen Schnittstellen, dieselbe *positive* Flußrichtung gewählt wird, z.B. *in das Element* gerichtet.

Damit genügt es, in einer Modell-Bibliothek zu definieren, von welchem Typ eine Variable ist (Potential- oder Fluß-Variable). Wenn eine Schnittstelle mit einer anderen verbunden wird, kann das objektorientierte Modellierungssystem dadurch die korrekten Gleichungen

erstellen, ohne daß z.B. die Kirchhoff'schen Gesetze explizit angegeben werden müssen.

Erstaunlicherweise werden bei dieser Vorgehensweise die sonst eingesetzten *globalen* Formalismen, wie die modifizierte Knotenpunktsanalyse für elektrische Systeme oder die Lagrange'schen bzw. Kane'schen Gleichungen für mechanische Systeme, nicht benötigt. Neben rein kontinuierlichen Systemen können auch unstetige, strukturvariable oder diskrete Komponenten mit der objektorientierten Modellierungstechnik behandelt werden.

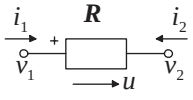
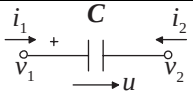
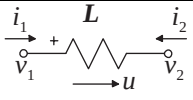
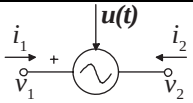
Verfügbare Programmsysteme unterstützen zur Zeit nur 2D-Objektdiagramme. Diese eignen sich sehr gut zur Visualisierung 1- oder 2-dimensionaler Repräsentationen von Systemen, wie elektrische Schaltkreise, Antriebsstränge, Blockdiagramme, hydraulische Systeme, endliche Automaten, Petrinetze, Statecharts. Sie sind nur bedingt geeignet zur Visualisierung 3-dimensionaler Systeme, wie 3D-Mechanik, 3D-Wärmeleitung, 3D-Strömungen. Die objektorientierte Modellierungstechnik ist jedoch unabhängig von der Art der Visualisierung. Anstatt Komponenten nur als Icon darzustellen, könnte man auch 3D-Konstruktionen verwenden. Wie im 2D-Bereich wird eine solche Komponente durch lokale Gleichungen beschrieben und die Gleichungsgenerierung läuft vollkommen analog ab, nur die Darstellung ist realistischer. Es ist abzusehen, daß die Hersteller ihre Systeme in dieser Hinsicht erweitern werden.

1.2 Ein vollständiges Beispiel

Zur einführenden Übersicht, wird an einem einfachen Beispiel der vollständige Zyklus — vom Objektdiagramm bis zur Zustandsform — vorgeführt. Hierzu wird eine kleine Bibliothek idealer elektrischer Bauteile erstellt, siehe Tabelle 1.

Die Komponenten-Schnittstellen sind die kleinen Kreise am linken und rechten Teil eines Bauteils und stellen elektrische Pins dar. Ein Pin wird mathematisch durch 2 Variablen beschrieben: Durch das elektrische *Potential v* am Pin (Typ = Potential-Variable)

Tabelle 1: Objektgleichungen idealer elektrischer Komponenten.

Widerstand		$0 = i_1 + i_2$ $u = v_1 - v_2$ $u = R \cdot i_1$
Kapazität		$0 = i_1 + i_2$ $u = v_1 - v_2$ $i_1 = C \cdot du/dt$
Induktivität		$0 = i_1 + i_2$ $u = v_1 - v_2$ $u = L \cdot di_1/dt$
Spannungsquelle		$0 = i_1 + i_2$ $u = v_1 - v_2$ $u = A \cdot \sin(\omega t)$
Erdung		$v = 0$

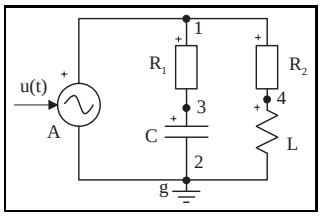


Bild 4: Objektdiagramm eines elektrischen Schaltkreises.

und durch die Größe des einfließenden *Stroms i* (Typ = Fluß-Variable). Basierend auf diesen Schnittstellenvariablen sind im rechten Teil von Tabelle 1 die *lokalen* Gleichungen der Komponenten aufgeführt. Diese Gleichungen sind nur Funktionen der Schnittstellenvariablen und der lokalen Variablen. Sie sind unabhängig davon, wie die Komponente mit anderen Komponenten verschaltet wird. Man beachte, daß die aufgeführten Zusammenhänge *mathematische Gleichungen* und keine Zuweisungen einer Programmiersprache sind.

Die erstellte Bibliothek wird jetzt benutzt, um den elektrischen Schaltkreis von Bild 4 zu modellieren.

Hierzu werden die benötigten Komponenten der Bibliothek entnommen und entsprechend dem Diagramm miteinander verschaltet, d.h. es werden Linien zwischen den Komponenten-Pins gezogen. Wie schon in Abschnitt 1.1 kurz skizziert, wird das Gesamtgleichungssystem des Modells aufgestellt, indem die Gleichungen aller verwendeten Komponenten, ergänzt um die Verbindungsgleichungen, zusammengefaßt werden. Diese Gleichungen sind in Tabelle 2 zusammengestellt. Um die Variablen unterschiedlicher Komponenten voneinander unterscheiden zu können, wird der Komponentenname der entsprechenden Variablen vorangestellt. Zum Beispiel ist $\mathrm{R1}.i_1$ der Strom i_1 vom Bauteil R1. Die *Gleichungen* der *Komponenten* sind eine direkte *Kopie* aus der Bibliothek. Die *Gleichungen* für die *Verbindungen* an den Knoten $1, 2, 3, 4$ ergeben sich daraus, daß alle elektrischen Potentiale als Potential-Variablen und alle Ströme als Fluß-Variablen in der Bibliothek definiert sind. Da an einer Verbindungsstelle alle Potential-Variablen gleichgesetzt werden und die Summe der Fluß-Variablen verschwindet, werden die korrekten Verbindungsgleichungen erstellt. Diese entsprechen den

Tabelle 2: Gesamtgleichungssystem.

R1	$0 = \mathrm{R1}.i_1 + \mathrm{R1}.i_2$ $\mathrm{R1}.u = \mathrm{R1}.v_1 - \mathrm{R1}.v_2$ $\mathrm{R1}.u = \mathrm{R1}.R \cdot \mathrm{R1}.i_1$	**R2**	$0 = \mathrm{R2}.i_1 + \mathrm{R2}.i_2$ $\mathrm{R2}.u = \mathrm{R2}.v_1 - \mathrm{R2}.v_2$ $\mathrm{R2}.u = \mathrm{R2}.R \cdot \mathrm{R2}.i_1$
C	$0 = \mathrm{C}.i_1 + \mathrm{C}.i_2$ $\mathrm{C}.u = \mathrm{C}.v_1 - \mathrm{C}.v_2$ $\mathrm{C}.i_1 = \mathrm{C}.C \cdot d\mathrm{C}.u/dt$	**L**	$0 = \mathrm{L}.i_1 + \mathrm{L}.i_2$ $\mathrm{L}.u = \mathrm{L}.v_1 - \mathrm{L}.v_2$ $\mathrm{L}.u = \mathrm{L}.L \cdot d\mathrm{L}.i_1/dt$
A	$0 = \mathrm{A}.i_1 + \mathrm{A}.i_2$ $\mathrm{A}.u = \mathrm{A}.v_1 - \mathrm{A}.v_2$ $\mathrm{A}.u = \mathrm{A}.A \cdot \sin(\mathrm{A}.w * t)$	**g**	$\mathrm{g}.v = 0$
1	$\mathrm{A}.v_1 = \mathrm{R1}.v_1$ $\mathrm{A}.v_1 = \mathrm{R2}.v_1$ $0 = \mathrm{A}.i_1 + \mathrm{R1}.i_1 + \mathrm{R2}.i_1$	**2**	$\mathrm{g}.v = \mathrm{C}.v_2$ $\mathrm{g}.v = \mathrm{L}.v_2$ $\mathrm{g}.v = \mathrm{A}.v_2$ $0 = \mathrm{A}.i_2 + \mathrm{C}.i_2 + \mathrm{L}.i_2 + \mathrm{g}.i$
3	$\mathrm{R1}.v_2 = \mathrm{C}.v_1$ $0 = \mathrm{R1}.i_2 + \mathrm{C}.i_1$	**4**	$\mathrm{R2}.v_1 = \mathrm{L}.v_1$ $0 = \mathrm{R2}.i_2 + \mathrm{L}.i_1$

Kirchhoff'schen Gesetzen.

Tabelle 2 ist eine DAE mit 27 Gleichungen, die in die Zustandsform

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \tag{1.1}$$

transformiert werden soll. Die Zustandsform (1.1) kann aus der DAE erhalten werden, wenn bei *bekanntem* Zustandsvektor $\mathbf{x}$ die *Ableitung* des Zustandsvektors $\dot{\mathbf{x}}$ berechnet wird. In Tabelle 2 treten nur die Ableitungen dC.u/dt und dL.i_1/dt auf. Damit sind C.u und L.i_1 Zustandsgrößen: $\mathbf{x} = [\text{C}.u; \text{L}.i_1]$.

Die Aufgabe besteht damit darin, bei *bekannten* Zuständen C.u, L.i_1 und *bekannten* Parametern R1.R, R2.R, C.C, L.L, A.A, A.w aus der DAE die Ableitungen der Zustände, d.h. dC.u/dt und dL.i_1/dt, zu berechnen. Mit diesen Vorgaben ist die DAE in Tabelle 2 ein algebraisches Gleichungssystem mit 27 Gleichungen in den 27 Unbekannten

R1.i_1	R1.i_2	R1.v_1	R1.v_2	R1.u
R2.i_1	R2.i_2	R2.v_1	R2.v_2	R2.u
C.i_1	C.i_2	C.v_1	C.v_2	dC.u/dt
dL.i_1/dt	L.i_2	L.v_1	L.v_2	L.u
A.i_1	A.i_2	A.v_1	A.v_2	A.u
g.i	g.v			

Manuell ist eine Auflösung nach den eigentlich interessierenden zwei Zustandsableitungen aufwendig. Doch mit den noch zu besprechenden *Algorithmen* kann ein Programm sehr schnell die DAE *sortieren* und Variablen aus trivialen Gleichungen der Form $a = b$ *substituieren*. Als Ergebnis erhält man die folgende rekursive Berechnungsvorschrift zur Bestimmung der Zustandsableitungen (in der linken Spalte sind die Komponenten angegeben, aus denen die Gleichungen entnommen wurden):

R2:	$\text{R2}.u := \text{R2}.R \cdot \text{L}.i_1$
A:	$\text{R1}.v_1 := \text{A}.A \cdot \sin(\text{A}.w * t)$
L:	$\text{L}.u := \text{R1}.v_1 - \text{R2}.u$
R1:	$\text{R1}.u := \text{R1}.v_1 - \text{C}.u$
R1:	$\text{C}.i_1 := \text{R1}.u / \text{R1}.R$
L:	$d\text{L}.i_1/dt := \text{L}.u / \text{L}.L$
C:	$d\text{C}.u/dt := \text{C}.i_1 / \text{C}.C$

Man könnte nun alle Zwischenvariablen (R2.u, ..., C.i_1) in die letzten zwei Gleichungen einsetzen und hätte dann nur noch zwei Gleichungen. Für größere Systeme ist ein solches Vorgehen jedoch unsinnig: Wenn eine Zwischenvariable, wie R1.v_1, an mehreren Stellen auftritt, und die Variable überall durch ihre Definitionsgleichung ersetzt wird, dann wird diese Gleichung *mehrmals* ausgewertet, statt nur *einmal*, wie in der obigen rekursiven Berechnungsvorschrift.

Das Beispiel zeigt, daß das Vorgehen der objektorientierten Modellierung *systematisch* und recht *einfach* ist. Auch wird das Verständnis erleichtert: Es genügt, die *lokalen* Gleichungen einer Komponente zu verstehen. Die Komplexität ergibt sich durch das Zusammenschalten von Komponenten und die nachfolgende Transformation. Diese Systematik führt schon bei dem obigen Trivialbeispiel auf 27 Gleichungen, so daß diese für das *manuelle* Erstellen von Gleichungen ungeeignet ist und eine Rechnerunterstützung unabdingbar ist.

1.3 Verfügbare Programmsysteme

Das erste objektorientierte Modellierungssystem, *Dymola*, wurde 1978 erstellt [4]. In den 90er Jahren wurden eine ganze Reihe von Softwaresystemen an Universitäten realisiert, zum Teil mit einer nachfolgenden Kommerzialisierung. In Tabelle 3 ist eine (unvollständige) Liste dieser Programme zusammen mit Internet-Adressen aufgeführt, bei denen detailliertere Informationen erhalten werden können. Kommerzielle Programme sind mit einem * gekennzeichnet.

Tabelle 3: Programme zur objektorientierten Modellierung.

Programm	**Internet-Adresse**
ABACUS	http://yoric.mit.edu/abacuss/abacuss.html
ASCEND	http://www.cs.cmu.edu/~ascend/
DIVA	http://www.isr.uni-stuttgart.de/diva/diva.html
Dymola*	http://www.dynasim.se/
gPROMS*	http://www.psenterprise.com/gPROMS/
MOSES	http://www.elet.polimi.it/section/automeng/control/oo
NMF/IDA*	http://www.brisdata.se/
Modelica	**http://www.modelica.org/**
ObjectMath	http://www.ida.liu.se/labs/pelab/omath/
Omola	http://www.control.lth.se/~cace/omsim.html
Saber*	http://www.analogy.com/
Shift	http://www.path.berkeley.edu/shift/
SIDOPS+	http://www.rt.el.utwente.nl/proj/modsim/modsim.htm
Smile	http://www.first.gmd.de/smile/
VHDL-AMS	http://www.vhdl.org/analog

Modelica ist eine objektorientierte Modellierungssprache, die von den Entwicklern von Allan, Dymola, NMF, ObjectMath, Omola, SIDOPS+, Smile und einer Reihe von Anwendern seit 1996 entwickelt wird, um einen Standard auf diesem Gebiet zu schaffen. Auf Grund der Mächtigkeit von Modelica und da die Sprache nicht an einen kommerziellen Hersteller gebunden ist, werden die Beispiele in dieser Artikelserie exemplarisch mit dieser Modellierungssprache formuliert.

Literatur

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

[2] *Cellier, F. E.:* Continuous System Modeling. Springer Verlag, New York, 1991.

[3] Dymola. Homepage: http://www.dynasim.se/

[4] *Elmqvist, H.:* A Structured Model Language for Large Continuous Systems. PhD dissertation, Depart. of Automatic Control, Lund Institute of Technology, Lund, Schweden, 1978.

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

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

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

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

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

[10] SystemBuild. http://www.isi.com/Products/MATRIXx/

[11] WorkingModel. Homepage: http://www.workingmodel.com/

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 2

Martin Otter, DLR Oberpfaffenhofen

2 Modelica – Kontinuierliche Systeme

In Teil 1 wurden die Grundideen der *objektorientierten Modellierung* physikalischer Systeme erläutert und insbesondere gezeigt, wie Modelle komponentenweise mittels *Objektdiagrammen* grafisch definiert werden können. Damit Anwender neue Basiskomponenten in ein Objektdiagramm einführen können, wird üblicherweise eine Modellierungssprache zur Verfügung gestellt, mit der dann auch ein vollständiges Modell beschrieben werden kann. Dadurch ist ein Objektdiagramm relativ einfach in eine rein textuelle Beschreibung überführbar, die auf einer Datei gespeichert und transportiert werden kann. Um die im ersten Teil dieser Serie übersichtsartig eingeführten Grundelemente der objektorientierten Modellierung präziser zu fassen, ist es zweckmässig eine formalere Beschreibung an Hand einer geeigneten Modellierungssprache vorzunehmen.

In Tabelle 3 von Teil 1 wurden Verweise auf eine ganze Reihe von objektorientierten Modellierungssprachen gegeben. Es würde den kompakten Rahmen dieser Artikelserie sprengen, wenn auf jede dieser Sprachen im einzelnen eingegangen werden würde. Stattdessen werden die Grundelemente an Hand der Sprache *Modelica*[1] eingehender erläutert. Modelica wird von den Entwicklern der Modellierungssprachen Allan, Dymola, NMF, ObjectMath, Omola, SIDOPS+, Smile, sowie einer Reihe von Anwendern, seit 1996 entwickelt, um einen Standard auf diesem Gebiet zu schaffen. Das vorliegende Kapitel gibt eine Einführung in die Modellierung *kontinuierlicher* Systeme auf der Grundlage von Modelica und basiert auf [13, 16]. Auf die objektorientierte Modellierung unstetiger, strukturvariabler und diskreter Systeme wird in einem späteren Artikel eingegangen.

2.1 Hierarchische Modelle

Die Grundlagen von Modelica werden an Hand des einfachen Antriebsstrangs von Bild 5 erläutert (Bild 5 und Bild 6 sind Bildschirmabzüge des Programms Dymola [12]). Der Antriebsstrang besteht aus Regler, Elektromotor, Getriebe und Last. Ein Objektdiagramm-Editor erzeugt daraus das folgende Modelica Modell, wobei auch die grafische Information des Objektdiagramms im Modelica Modell als *annotation* gespeichert wird. Aus Gründen der Übersichtlichkeit wird diese Information jedoch nicht dargestellt:

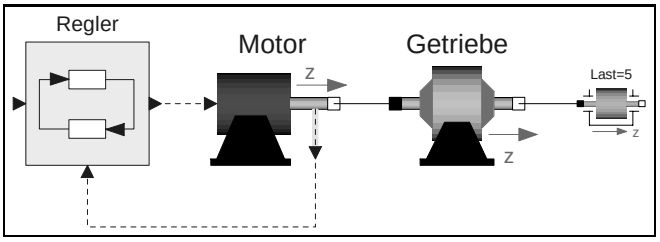


Bild 5: Objektdiagramm eines Antriebsstrangs.

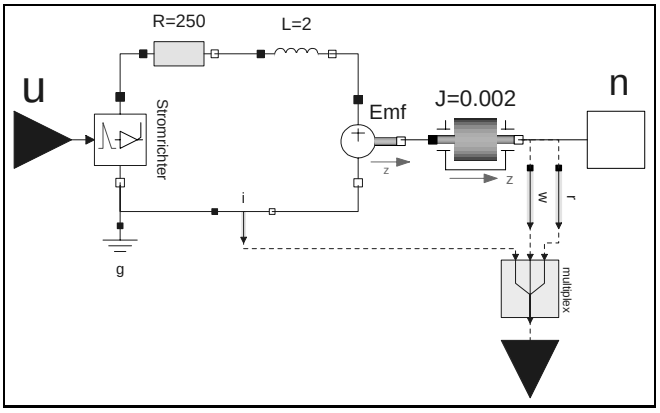


Bild 6: Objektdiagramm der Motor-Komponente.

```
model Antriebsstrang
  Control Regler;
  Motor   Motor;
  GearBox Getriebe;
  Shaft   Last (J=5);
equation
  connect(Regler.out, Motor.in);
  connect(Motor.out , Regler.in2);
  connect(Motor.n   , Getriebe.p);
  connect(Getriebe.n, Last.p);
end Antriebsstrang;
```

Mit diesem Modell werden die Komponenten `Regler`, `Motor`, `Getriebe` und `Last` definiert, sowie deren Verschaltung. Mit der Anweisung `Shaft Last(J=5);` wird die neue Komponente `Last` von der Modell-Klasse `Shaft` deklariert und das Trägheitsmoment `J` der Last auf `5` $\mathrm{kg\,m^2}$ gesetzt. Der **`equation`** Teil enthält die Modellgleichungen. Im obigen Beispiel werden die Modellgleichungen implizit durch die **`connect`** Anweisungen definiert, die festlegen, wie die Schnittstellen von Komponenten verschaltet sind. Eine Komponente kann wiederum hierarchisch aufgebaut sein, wie es beim Motor der Fall ist. Das Objektdiagramm des Motors ist in Bild 6 zu sehen. Das entsprechende Modelica Modell lautet:

[1] Modelica™ ist ein Warenzeichen der „Modelica Design Group“.

```
model Motor
  Resistor R(R=250);
  Inductor L(L=2);
  Shaft    inertia(J=0.002);
  Flange   n;
   ...
equation
  connect(R.n, L.p);
  connect(inertia.n, n);
    ...
end Motor;
```

Mit der zweiten Anweisung wird die Komponente `R` der Modell-Klasse `Resistor` definiert, wobei der Widerstandswert auf `250` Ω gesetzt wird.

2.2 Variablen

Komponenten enthalten Variablen, mit denen die Gleichungen formuliert werden. Diese Variablen haben eine *physikalische* Bedeutung. In der Standard-Bibliothek von Modelica werden die wichtigsten Variablentypen vordefiniert zur Verfügung gestellt, z.B.:

```
type Voltage = Real(quantity   = "Voltage",
                    unit       = "V");
type Angle   = Real(quantity   = "Angle",
                    unit       = "rad",
                    displayUnit= "deg");
type Radius  = Real(quantity   = "Length",
                    unit       = "m",
                    min        = 0.0);
```

Hierbei ist `Real` eine vordefinierte Typ-Klasse für Gleitpunktzahlen, die einige Parameter, wie Einheit, Anfangswert, minimaler oder maximaler Wert, besitzt. Mit dem Attribut `unit` wird die Einheit definiert, in der die *Gleichungen* formuliert werden. Das Attribut `displayUnit` gibt an, welche Einheit als *Voreinstellung* für die *Ein-* und *Ausgabe* benutzt werden soll. Mit der Voreinstellung wird z.B. ein Winkel in einem Objektdiagramm standardmäßig in Grad eingegeben, bzw. der Zeitverlauf eines Winkels in Grad dargestellt. In den Gleichungen wird ein Winkel jedoch in Radiant benutzt. Die Umrechnung nimmt das Modellierungssystem vor.

2.3 Schnittstellen

Schnittstellen einer Komponente definieren, wie die Komponente mit anderen Bauteilen in Kontakt treten kann. Eine Schnittstelle enthält alle Variablen mit denen über die Schnittstelle Informationen ausgetauscht werden können. Zum Beispiel wird eine elektrische Schnittstelle, ein *Pin*, eindeutig durch das Potential `v` am Pin und durch den einfließenden Strom `i` definiert. Zur Schnittstellen-Definition wird in Modelica die Connector-Klasse benutzt, siehe die erste Zeile in Tabelle 4. Connector-Klassen werden ebenso wie Modell-Klassen benutzt:

```
   ...
  Pin p1, p2, p3;
equation
  p1.v = 0;
  connect(p1, p2);
  connect(p1, p3);
   ...
```

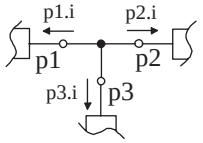
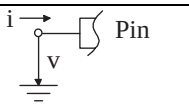
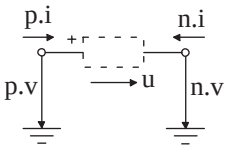
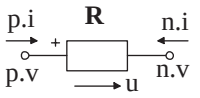
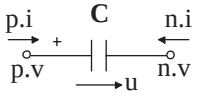


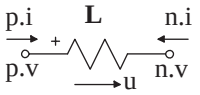
Tabelle 4: Modelica Modelle von elektrischen Komponenten.

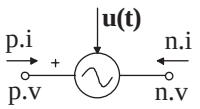
	`connector Pin`
	`  Voltage  v;`
	`  flow Current i;`
	`end Pin;`

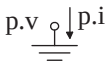
	`partial model TwoPin`
	`  Pin  p, n;`
	`  Voltage u;`
	`equation`
	`  u = p.v - n.v;`
	`  0 = p.i + n.i;`
	`end TwoPin;`

	`model Resistor`
	`  extends TwoPin;`
	`  parameter Real R;`
	`equation`
	`  u = R*p.i;`
	`end Resistor;`

	`model Capacitor`
	`  extends TwoPin;`
	`  parameter Real C;`
	`equation`
	`  C*der(u) = p.i;`
	`end Capacitor;`

	`model Inductor`
	`  extends TwoPin;`
	`  parameter Real L;`
	`equation`
	`  L*der(p.i) = u;`
	`end Inductor;`

	`model Vsource`
	`  extends TwoPin;`
	`  extends Modelica.Math;`
	`  parameter Real A, w;`
	`equation`
	`  u = A*sin(w*time);`
	`end Vsource;`

	`model Ground`
	`  Pin p;`
	`equation`
	`  p.v = 0;`
	`end Ground;`

In der zweiten Zeile werden 3 Pins definiert. Zur eindeutigen Identifikation muß beim Zugriff auf eine Variable der Komponentenname mit angegeben werden (z.B. `p1.v` = Variable `v` der Komponente `p1`). Mit den **`connect`** Anweisungen werden die 3 Pins verschaltet. Das Modellierungssystem erstellt hieraus die Gleichungen: `p1.v = p2.v`, `p1.v = p3.v` und `p1.i + p2.i + p3.i = 0`. Eine Null-Summengleichung wird erzeugt, wenn verbundene Variable das Attribut **`flow`** besitzen, siehe erste Zeile von Tabelle 4, d.h. wenn die Variablen explizit als Flußvariablen deklariert sind. Dies wurde schon kurz in Teil 1 erläutert.

2.4 Unvollständige Modelle und Vererbung

Für den Aufbau von komplexen Modellen ist es wichtig, daß gemeinsame Eigenschaften nur *einmal* definiert werden. Z.B. haben eine Reihe von elektrischen Komponenten, wie Widerstand, Kapazität, Induktivität, jeweils zwei Pins. Außerdem wird zur Formulierung des physikalischen Gesetzes der Spannungsabfall `u` benötigt. Deswegen ist es sinnvoll eine „unvollständige“ Modell-Klasse `TwoPin` zur Verfügung zu stellen, in der diese Eigenschaften nur einmal für entsprechende elektri-

sche Komponenten definiert werden, siehe zweite Zeile von Tabelle 4. Diese Modell-Klasse hat zwei Pins `p`, `n` und den Spannungsabfall `u`. Der **`equation`** Teil enthält die allen Komponenten gemeinsamen Gleichungen in Form von *mathematischen Gleichungen*. Damit könnte `0 = p.i + n.i` alternativ auch als `n.i = -p.i` geschrieben werden. Statt Gleichungen können in Modelica auch *Zuweisungen* unter Verwendung des Operators `:=` in einer **`algorithm`** Sektion verwendet werden, um z.B. diskrete Regler zu beschreiben.

Mit dem Schlüsselwort **`partial`** wird festgelegt, daß eine Instanziierung des Modells nicht möglich ist, d.h. daß diese unvollständige Modell-Klasse nur zum Aufbau weiterer Modell-Klassen verwendet werden kann. Dies wird in den restlichen Zeilen von Tabelle 4 für die einfachen elektrischen Komponenten aus Tabelle 1 von Teil 1 gezeigt.

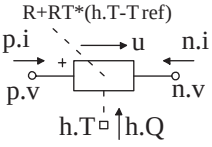
Mit der Anweisung **`extends`** `TwoPin` *erbt* eine Modell-Klasse alle Eigenschaften vom Modell `TwoPin`, d.h. alle Deklarationen und Gleichungen von `TwoPin` stehen direkt in dem neuen Modell zur Verfügung. Die **`extends`** Anweisung kann mehrmals in einem Modell auftreten. Z.B. erbt die Modell-Klasse `Vsource` in Tabelle 4 nicht nur von `TwoPin`, sondern auch von der Standard-Bibliothek `Modelica.Math`, die mathematische Funktionen, wie `sin(..)`, zur Verfügung stellt.

Mit einer **`parameter`** Anweisung wird eine Variable definiert, die während einer Simulation *konstant* ist. Bei der Deklaration einer Komponente kann jedem Parameter ein neuer Wert zugewiesen werden (siehe auch das obige Motor Modell). Die Zeitableitung einer Variablen wird durch den Operator **`der`** definiert, siehe die `Capacitor` und `Inductor` Modell-Klassen in Tabelle 4.

2.5 Parametrisierung von Modellen

In Modelica können von einer höheren Modellhierarchie aus nicht nur Parameterwerte, sondern auch komplette *Komponenten* oder gar *Klassen* eines Submodells ausgetauscht werden. Zum Beispiel soll für den Motor von Bild 6 ein genaueres Modell für den Widerstand `R` benutzt werden, bei dem die *Temperaturabhängigkeit* des Widerstandes wie folgt berücksichtigt wird:

```
model TempResistor
  extends TwoPin;
  heatTransition h;
  parameter Real R, RT=0;
  parameter Real Tref=20;
equation
  u=(R+RT*(h.T-Tref))*p.i;
  h.Q = -u*p.i;
end TempResistor;
```



Die Modell-Klasse `TempResistor` kann *nicht* von `Resistor` durch Vererbung abgeleitet werden, weil der Spannungsabfall über eine andere Gleichung berechnet wird. Deswegen wird wiederum von `TwoPin` abgeleitet, und es werden eine zusätzliche Schnittstelle `h` für den Wärmeübergang mit den Schnittstellen-Variablen `T` (Temperatur) und `Q` (Wärmefluß), sowie Gleichungen zur Bestimmung des Spannungsabfalls `u` und der erzeugten Wärme `h.Q` hinzugefügt. Basierend auf dem ursprünglichen Motor-Modell kann jetzt die Modell-Klasse `Resistor` der Komponente `Motor.R` durch das genauere Modell `TempResistor` ersetzt werden. Weiterhin werden noch einige Elemente zur Beschreibung des Wärmeflusses zwischen Widerstand und Umgebung hinzugefügt:

```
model Motor2
  extends Motor( redeclare
                 TempResistor R(RT=0.1) );
  Tsource  Umgebung(T0=20);
  HeatFlow Hflow(...);
equation
  connect(Umgebung.p, Hflow.n);
  connect(Hflow.p   , R.h);
end Motor2;
```

Durch die Anweisung **`redeclare`** wird die Ersetzung durchgeführt, wobei neue Parameterwerte (hier: `RT`) festgelegt werden. Ein solcher Komponenten-Austausch ist genau dann möglich, wenn die neue Modell-Klasse (hier: `TempResistor`) *alle* Schnittstellen-Variablen und Parameter der zu ersetzenden Klasse (hier: `Resistor`) enthält, wobei Namen und Datentypen übereinstimmen müssen. Dies ist hier der Fall, da `TempResistor` die Schnittstellen-Variablen von `Resistor`, d.h. `p`, `n`, `R`, besitzt.

Der Vorteil dieser Vorgehensweise besteht darin, daß jede Modifikation des `Motor`-Modells, sofort auch beim `Motor2`-Modell zur Auswirkung kommt. Damit können z.B. ein einfaches Modell für den Entwurf und ein komplexeres für die Validierung zuverlässig gewartet werden. Es ist auch möglich eine ganze Klasse von Komponenten mit einem Befehl auszutauschen, wenn dies im ursprünglichen Modell vorgesehen wird:

```
model Circuit
  replaceable model Resistor2 = Resistor;
protected
  Resistor2 R1(R=100), R2(R=200), R3(R=300);
    ...
end Circuit;

model Circuit2
  extends Circuit ( redeclare
          model Resistor2 = TempResistor );
    ...
end Circuit2;
```

Mit dem Sprachelement **`replaceable`** wird definiert, daß die lokale Modellklasse `Resistor2` *austauschbar* ist. Als Voreinstellung wird die `Resistor` Modell-Klasse benutzt. Jetzt können viele Komponenten der Klasse `Resistor2` deklariert werden. Durch nachträgliches Austauschen von `Resistor` mit `TempResistor`, werden *alle* Widerstands-Komponenten durch diese neue Modell-Klasse beschrieben.

Diese Vorgehensweise erleichtert insbesondere die Modellierung von Fluid-Strömungen, siehe [15, 14], und wird in einem späteren Artikel detaillierter erläutert: Eine Komponente, wie eine Pumpe oder ein Wärmetauscher, wird hier durch *allgemeine* Gleichungen, wie Energie-, Impuls-, Stoffbilanzgleichungen, sowie durch *Medium*-Gleichungen beschrieben, mit denen die Stoffeigenschaften des eingesetzten Mediums, wie Dichte

oder spezifische Wärmekapazität, als Funktion der Zustandsgrößen berechnet werden. Zum einen gibt es unterschiedlich detallierte Mediummodelle, zum anderen können meist auch unterschiedliche Medien für dieselbe Komponente verwendet werden. Es ist nun zweckmäßig, die Mediumgleichungen als austauschbare Klasse zu definieren, so daß z.B. nur *eine* Wärmetauscher-Klasse für *n* Mediummodelle zur Verfügung gestellt wird, und nicht *n* Wärmetauscher-Klassen für *n* Mediummodelle. Zum Beispiel wird eine neue Wärmetauscher-Komponente mit der in [15] beschriebenen generischen Modell-Klasse `HEXn` für ein einfaches Wassermodell `BasicLinearWaterModel` folgendermaßen instanziiert:

```
HEXn heatExchanger1 (redeclare model
     Liquid=BasicLinearWaterModel, ...);
```

2.6 Eingeschränkte Klassen

Es hat den Anschein, als ob die Strukturierungselemente von Modelica, wie **`model`**, **`type`**, **`connector`**, voneinander unabhängige Sprachelemente sind. Dies ist jedoch nicht der Fall. Modelica kennt nur *ein* Strukturierungselement: die Klasse **`class`**. Es gibt Klassen mit speziellen Namen, die alle Eigenschaften von **`class`** besitzen, wie Syntax, Semantik, Definition, Vererbung, Parametrisierung, die jedoch nur in eingeschränkter Form benutzt werden können:

`connector`	Wird in Verbindungen benutzt und hat keine Gleichungen.
`model`	Darf nicht in Verbindungen benutzt werden.
`record`	**`model`** ohne Gleichungen.
`type`	Nur durch Vererbung von einem **`type`** oder einem vordefinierten Typ wie `Real` ableitbar.
`block`	**`model`** wobei alle Schnittstellen-Variablen als **`input`** oder **`output`** deklariert sind.
`function`	**`block`** mit *einer* Algorithmus-Sektion.
`package`	**`model`** das nur Klassen-Deklarationen enthält.

Die **`record`** Klasse wird zum Aufbau von hierarchischen Datenstrukturen eingesetzt. Mit der **`block`** Klasse werden Ein-/Ausgangsblöcke definiert. Hierdurch wird u.a. erreicht, daß es Einschränkungen bei der Verschaltung gibt, so daß z.B. Eingänge nicht mit Eingängen verschaltet werden können. Die **`function`** Klasse ist eine spezielle Block-Klasse und entspricht einer Funktion in einer prozeduralen Programmiersprache. Mit der **`package`** Klasse werden hierarchische Komponenten-Bibliotheken aufgebaut, wobei, ähnlich wie in Java, die Korrespondenz zwischen Klassenname und Directory/Filename genau definiert ist, so daß die Modellierungsumgebung eine gewünschte Klassendefinition im Filesystem finden kann. Durch die Technik der *eingeschränkten* Klassen muß der Anwender nur die Verwendung der Klasse **`class`** verstehen, und nicht sieben unterschiedliche Konzepte. Darüber hinaus wird das Erstellen von Modelica Compilern vereinfacht, da nur die Syntax und Semantik einer **`class`** implementiert werden muß, sowie einige zusätzliche Überprüfungen, ob die geforderten Restriktionen erfüllt sind.

Die **`block`**-Klasse wird vor allem zur Beschreibung von Ein-/Ausgangsblöcken verwendet. Zum Beispiel kann ein lineares Zustandsraummodell als

```
block StateSpace
  parameter Real A[:,:],
                B[size(A,1),:],
                C[:,size(A,2)],
                D[size(C,1),size(B,2)]=0;
  input  Real u[size(B,2)];
  output Real y[size(C,1)];
protected
  Real x[size(A,2)];
equation
  der(x) = A*x + B*u;
       y = C*x + D*u;
end StateSpace;
```

definiert und folgendermaßen benutzt werden:

```
StateSpace S(A = [0.12, 2; 3, 1.5],
     B = [2, 7; 3, 1], C = [0.1, 0.4]);
```

2.7 Sonstige Sprachelemente

Es werden mehrdimensionale Felder, Matrix-Operatoren, -Funktionen und -Gleichungen unterstützt. Damit können z.B. einfach Regelungssysteme, Mehrkörpersysteme, Komponenten-Felder und reguläre Verschaltungsstrukturen beschrieben werden. Dies erlaubt Orts-Diskretisierungen von partiellen Differentialgleichungen, siehe [15]. Es können auch Abtastsysteme, unstetige und strukturvariable Systeme modelliert werden. Auf Grund der Bedeutung und des Umfangs wird dieser Teil in einem späteren Artikel erläutert. Die Modelica-Gruppe arbeitet zur Zeit an einer umfangreichen Modelica-Standard-Bibliothek, damit die wichtigsten Komponenten aus unterschiedlichen Fachgebieten schon vordefiniert zur Verfügung stehen. Weitere Informationen können der Modelica-Homepage http://www.modelica.org/ entnommen werden.

Literatur

[12] Dymola. Homepage: http://www.dynasim.se

[13] *Elmqvist, H., Boudaud, F., Broenink, J., Brück, D., Ernst, T., Fritzson, P., Jeandel, A., Juslin, K., Klose, M., Mattsson, S.E., Otter, M., Sahlin, P., Tummescheit, H.* und *Vangheluwe, H.:* Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Version 1, 1997 Modelica homepage: http://www.modelica.org/.

[14] *Ernst, T., Klose, M.* und *Tummescheit, H.:* Modelica and Smile – A Case Study Applying Object-Oriented Concepts to Multi-facet Modeling. Hahn und Lehmann (Editors), *1997 European Simulation Symposium (ESS'97)*, Passau, 1997. Siehe auch http://www.modelica.org/papers/smet20.pdf.

[15] *Mattsson, S.E.:* On Modeling of Heat Exchangers in Modelica. Hahn und Lehmann (Editors), European Simulation Symposium (ESS'97), S. 127–133, Passau, 1997. Siehe auch http://www.modelica.org/papers/ESS97HEX.pdf.

[16] *Mattsson, S.E., Elmqvist, H.* und *Otter, M.:* Physical System Modeling with Modelica. Control Engineering Practice. Accepted for publication.

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 3

Martin Otter, DLR Oberpfaffenhofen

3 Komponenten-Schnittstellen

In Teil 1 wurden die Grundlagen der *objektorientierten Modellierung* skizziert und in Teil 2 wurden die benötigten Grundelemente an Hand der Modellierungssprache Modelica diskutiert. Im vorliegenden Kapitel wird nun, basierend auf [23, 24, 22, 17], detaillierter erläutert, wie die *Schnittstellen* von Komponenten entworfen werden. Der Schnittstellenentwurf ist ein zentraler Schritt, da damit die voneinander unabhängigen Teile eines komplexen Modells festgelegt werden. Erst *nach* der Schnittstellen-Definition können Komponenten *unabhängig* voneinander entwickelt und ausgetestet werden. Da in der Regel bei physikalischen Modellen Schnittstellen *nicht* über Ein-/Ausgangsbeziehungen festgelegt werden, muß eine andere Methodik herangezogen werden. Die Grundregel des „richtigen" Schnittstellen-Entwurfs besteht darin, sich an der physikalischen *Realität* zu orientieren:

Wenn eine reale Komponente, wie ein Elektromotor oder eine hydraulische Pumpe, gedanklich freigeschnitten wird, sollte die abstrahierte Schnittstelle des Modells alle Variablen enthalten, die notwendig sind, um die beabsichtigten Effekte in der realen Schnittstelle zu beschreiben.

Der Einsatz dieser einleuchtenden Grundregel in einem konkreten Anwendungsfall ist oft verblüffend schwierig. Aus diesem Grunde wird versucht, eine systematische Einführung zu geben, die auf Methoden der Thermodynamik und der Bondgraphentheorie basiert.

3.1 Energiefluß

Physikalische Systeme haben die gemeinsame Eigenschaft, daß Energie zwischen den Komponenten ausgetauscht wird. Eine Komponenten-Schnittstelle sollte deswegen primär alle Variablen enthalten, mit denen der *Energiefluß* in der Schnittstelle eindeutig beschrieben werden kann. In Bild 7 ist ein freigeschnittenes System zu sehen, bei dem angenommen wird, daß die Gesamtenergie E des Systems eine Funktion der Schnittstellen- und lokalen Variablen ξ_i ist. Durch Ableiten ergibt sich die *Gibb'sche Fundamentalgleichung*, siehe z.B. [26, 18]:

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E(\xi_1, \ldots, \xi_n)}{\partial \xi_i} \cdot \frac{d\xi_i}{dt} = \sum_{i=1}^{n} e_i \cdot f_i$$

Diese gibt an, wie sich die Energie eines Systems durch Zufuhr von Energieflüssen $P_i = e_i \cdot f_i$ ändert, wobei ein

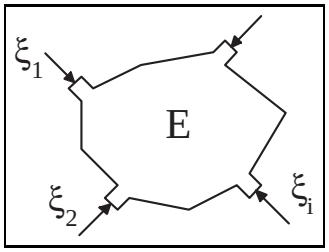


Bild 7: Gesamtenergie eines freigeschnittenen Systems.

Energiefluß durch das Produkt der *Potential*-Variablen e_i und der *Fluß*-Variablen f_i beschrieben wird. Wenn der Energiefluß *nicht* durch Wechselwirkung mit einem Feld entsteht, kann die Variable ξ_i als *Träger* interpretiert werden, der die Energie transportiert, z.B. Ladung oder Masse. Der Fluß f_i ist die Menge dieses Trägers pro Zeit, d.h. $f_i = d\xi_i/dt$. Die Variable e_i wird auch als intensive Größe und die Variable ξ_i als extensive Größe bezeichnet, siehe [18, 19].

Typischerweise hat jedes Fachgebiet *eine* (abstrahierte) Art der physikalischen Verbindung zwischen zwei Komponenten, über die der Energietransport stattfindet, z.B. leitendes Material bei elektrischen Schaltkreisen, mechanische Flansche bei Antriebssträngen, Körper-Oberflächen beim Wärmetransport. Das Potential e ist in diesen Fällen diejenige physikalische Größe, die bei einer Schnittstelle zwischen zwei oder mehr Komponenten denselben Wert besitzt, z.B. elektrisches Potential, Drehzahl oder Temperatur. Da sich bei dieser Betrachtungsweise das Potential an der typischen Art der physikalischen Verschaltung orientiert, ist die Wahl der Potential-Variablen in der Regel eindeutig[1].

Wenn Energie an einem Punkt zusammenfließt, *und in diesem Punkt keine Energie gespeichert wird*, dann muß die Summe der zufließenden Energie auf Grund des Energiesatzes verschwinden, d.h.

$$0 = \sum_i P_i = \sum_i e_i \cdot f_i = e \cdot \sum_i f_i \;\Rightarrow\; \sum_i f_i = 0.$$

Da die Potentiale e_i an einem Verbindungspunkt definitionsgemäß identisch sind $e_i = e$, folgt daraus, daß die Summe der Flußvariablen verschwinden muß. Diese Eigenschaft ist in vielen Fachgebieten bekannt, z.B. als Kirchhoffsches Stromgesetz (die Summe der Ströme in einem elektrischen Knoten verschwindet), oder

[1] In der Bondgraphentheorie ist dies *nicht* der Fall. Dort charakterisiert ein Bond nicht notwendigerweise eine physikalische Verbindung, so daß die Wahl von Effort-Variable e und Flow-Variable f nicht eindeutig ist und immer vertauscht werden kann.

Tabelle 5: Energiefluß-Variablen in verschiedenen Fachgebieten.

Typ	*Potential e*	*Träger ξ*	*Fluß f*
elektrisch	V : elektrisches Potential	q : Ladung	$\dot{q} = i$: Ladungsfluß = Strom
translatorisch	$\mathbf{v}$: Geschwindigkeit	$\mathbf{p}$: Impuls	$\dot{\mathbf{p}} = \mathbf{f}$: Impulsstrom = Kraft
rotatorisch	$\boldsymbol{\omega}$: Winkelgeschwindigkeit	$\mathbf{L}$: Drehimpuls	$\dot{\mathbf{L}} = \boldsymbol{\tau}$: Drehimpulsstrom = Moment
hydraulisch	p : Druck	V: Volumen	$\dot{V}$: Volumenstrom
thermisch	T : Temperatur	S : Entropie	$\dot{S}$: Entropiestrom
chemisch	μ : chem. Potential	N: Teilchen	$\dot{N}$: Teilchenstrom

als Newtonsches Gesetz *actio = reactio* (die Summe der Schnittkräfte an einer mechanischen Schnittstelle verschwindet).

Nach diesen vorbereitenden Überlegungen kann eine allgemeine Grundregel zum Schnittstellen-Entwurf angegeben werden:

> *Eine Komponenten-Schnittstelle sollte die Potentialvariable e und die Flußvariable f enthalten, mit der die über die Schnittstelle zugeführte Leistung $P = e \cdot f$ berechnet wird.*

Mit der in Kapitel 2 eingeführten Modelica Modellierungssprache sollte deswegen eine Elementar-Schnittstelle folgendermaßen definiert werden:

```
connector X
  Real      e;
  flow Real f;
end X;
```

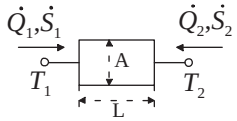
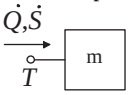
Die Flußvariable f muß als **flow** definiert werden, da, wie oben erläutert, bei einer Verbindung die Summe der Flußvariablen verschwindet. In Tabelle 5 sind die Potential- und Flußvariablen für einige Fachgebiete aufgeführt. Hierbei steht *translatorisch* und *rotatorisch* für translatorische und rotatorische mechanische Systeme, sowie *hydraulisch* für inkompressible Rohrströmungen bei geringen Geschwindigkeiten[2].

Man kann aus Tabelle 5 einige interessante Analogien aufstellen: Impuls, Drehimpuls und Entropie sind Energieträger und haben damit eine ähnliche physikalische Bedeutung wie andere Energieträger, wie Masse oder Ladung. Man kann diese Energieträger deswegen als „Teilchen“ ansehen, die die Energie zwischen zwei Potentialen transportieren. Die sich hieraus ergebende anschauliche Definition der Entropie als *Wärmemenge* wird z.B. in [21, 19] genauer ausgeführt.

3.2 Praktische Gesichtspunkte

Die im letzten Abschnitt eingeführte Grundregel sollte als erster Ansatz dienen. Es gibt jedoch eine ganze Reihe von praktischen Gründen, die Schnittstelle letztendlich etwas anders auszuführen. In Tabelle 7 sind diejenigen Schnittstellen-Variablen aufgeführt, die sich in verschiedenen objektorientierten Modellierungssystemen bewährt haben. Die Unterschiede zu Tabelle 5 werden im folgenden diskutiert. Man beachte, daß auch mit den modifizierten Schnittstellen primär immer noch der Energiefluß beschrieben wird.

Tabelle 6: Objektgleichungen idealer thermischer Komponenten.

	$T,\ \dot{Q}$	$T,\ \dot{S}$
Wärmewiderstand 	$0 = \dot{Q}_1 + \dot{Q}_2$ $\dot{Q}_1 = \frac{\lambda A}{L}(T_1 - T_2)$	$0 = T_1\dot{S}_1 + T_2\dot{S}_2$ $\dot{S}_1 = \frac{\lambda A}{L}\frac{(T_1 - T_2)}{T_1}$
Wärmekapazität 	$\dot{Q} = c \cdot m \cdot \dot{T}$	$\dot{S} = c \cdot m \frac{\dot{T}}{T}$

Thermische Systeme

Entgegen der Systematik von Tabelle 5, werden thermische Systeme traditionell mit dem Wärmefluß $\dot{Q}$ und der Temperatur T beschrieben. In Tabelle 6 sind die einfachst möglichen Basiselemente aufgeführt, siehe z.B. [20]: Der *Wärmewiderstand* ist ein ideales Element, welches keine Wärmeenergie speichert und nur die Wärme zwischen den beiden Schnittstellen 1 und 2 transportiert. Die *Wärmekapazität* ist ein ideales Element, welches nur Wärmeenergie speichert oder abgibt. Beide Elemente können entweder mit der Temperatur T und dem Wärmefluß $\dot{Q}$ oder mit der Temperatur T und dem Entropiestrom $\dot{S}$ beschrieben werden, da an einer Schnittstelle 1 gilt: $\dot{Q}_1 = T_1 \cdot \dot{S}_1$.

Es zeigt sich, daß bei diesen einfachst möglichen Elementen die Gleichungen linear in den Variablen $T, \dot{Q}$ sind und nichtlinear in den Variablen $T, \dot{S}$. Weiterhin kann die häufig auftretende Randbedingung der vollständigen Isolation im ersten Fall viel einfacher formuliert werden ($\dot{Q} = 0$). Schließlich sind Konsistenzprüfungen einfacher, da es einen Erhaltungssatz für die Wärmeenergie, aber nicht für die Entropie gibt. Aus diesen Gründen ist es besser, als Schnittstellenvariablen $\dot{Q}$ und T zu verwenden.

Mechanische Systeme

Die Behandlung von 3-dimensionalen mechanischen Systemen in einer objektorientierten Form benötigt eine längere Erläuterung und wird separat in einem der folgenden Artikel diskutiert. Bei 1-dimensionalen *rotatorischen* mechanischen Systemen, im folgenden durch

[2] Man beachte, daß die Bondgraph-Literatur bei mechanischen Systemen die Geschwindigkeit als Flow-Variable und die Kraft als Effort-Variable benutzt. Für Objektdiagramme ist diese Wahl nicht möglich, da bei einer starren Verbindung von z.B. Körpern und Gelenken die Geschwindigkeit und nicht die Kraft am Verbindungspunkt identisch ist. D.h. die Geschwindigkeit *muß* eine Potentialvariable sein.

Tabelle 7: Schnittstellen Variablen in verschiedenen Fachgebieten.

Typ	Potential-Variablen	Fluß-Variablen	Bemerkungen
elektrisch	V : elektrisches Potential	i : elektrischer Strom	
1D-translatorisch	s : Weg	f : Kraft	in lokalem Flanschsystem
Stellantrieb	φ : Winkel	τ : Moment	in lokalem Flanschsystem
Leistungsantrieb	ω : Winkelgeschwindigkeit	τ : Moment	in lokalem Flanschsystem
hydraulisch	p : Druck	$\dot{V}$: Volumenstrom	inkompressibel
Rohrströmung	p, T: Druck und Temperatur	$\dot{m}$: Massenstrom	1-phasig, 1-Stoff
thermisch	T : Temperatur	$\dot{Q}$: Wärmestrom	
chemisch	μ : chem. Potential	$\dot{N}$: Teilchenstrom	

Antriebsstrang abgekürzt, gibt es in der Technik zwei unterschiedliche Anwendungen:

(a) Bei *Stellantrieben* ist es das Ziel, eine gewünschte Position anzufahren oder einer vorgegebenen Bahn möglichst genau zu folgen. Beispiele hierfür sind Roboter oder Aufzüge. Da der *Drehwinkel* der Last geregelt werden soll, muß dieser in einer Schnittstelle enthalten sein. Die Drehzahl sollte *nicht* zusätzlich in die Schnittstelle mit aufgenommen werden, da dann der enge Zusammenhang zwischen Drehwinkel und Drehzahl dem Modellierungssystem nicht bekannt ist, so daß einige Algorithmen, wie der noch zu besprechende Pantelides-Algorithmus, nicht angewandt werden können. Wenn benötigt, kann die Drehzahl in einer objektorientierten Modellierungssprache durch Differentiation aus dem Drehwinkel erhalten werden ($\omega = \dot{\varphi}$). Der Compiler einer Sprache wie Modelica transformiert diese Anweisung in eine numerisch robust auszuwertende Form, so daß während der Simulation keine numerische Differentiation erfolgt (mehr dazu in Teil 5).

(b) Bei *Leistungsantrieben* ist es das vorrangige Ziel, eine bestimmte mechanische Leistung zu übertragen. Beispiele hierfür sind Antriebsstränge von Fahrzeugen, Pumpen oder Generatoren. Hier spielt der Drehwinkel überhaupt keine Rolle. Für eine Simulation wäre der absolute Drehwinkel einer Welle auch eine kritische Variable, da dieser monoton anwächst, also eine „instabile" Variable ist. Hier sollte man deswegen der ursprünglichen Grundregel folgen und die Drehzahl in der Schnittstelle benutzen.

Bei 1-dimensionalen *translatorischen* mechanischen Systemen können theoretisch auch die beiden bei rotatorischen Systemen auftretenden Fälle auftreten. In der Technik gibt es jedoch nur translatorische Stellantriebe und keine translatorischen Leistungsantriebe, so daß es hier genügt,2 nur mit einer Art von Schnittstellen zu arbeiten, in der die absolute Verschiebung, und nicht die Geschwindigkeit, auftritt.

Generell gibt es bei mechanischen Systemen das Problem, daß die auftretenden Größen, wie Geschwindigkeit oder Kraft, Vektoren sind. Die Elemente dieser Vektoren beziehen sich auf ein bestimmtes Koordinatensystem, das bei mathematischen Operationen zu berücksichtigen ist. Eine sinnvolle Vorgehensweise besteht darin, daß in *jedem* mechanischen Flansch ein *lokales* Koordinatensystem definiert wird, wobei die z-Achsen der Flansch-Koordinatensysteme von *einer* Komponente alle in dieselbe Richtung zeigen, und diese Richtung im Icon der Komponente dargestellt wird, siehe Bild 8.

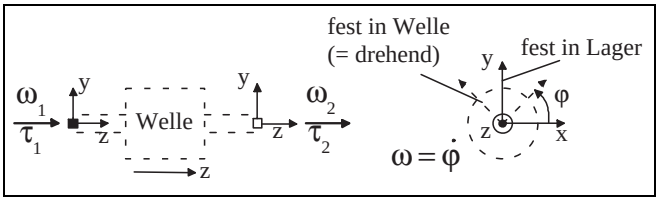


Bild 8: Lokale Koordinatensysteme bei Antriebssträngen.

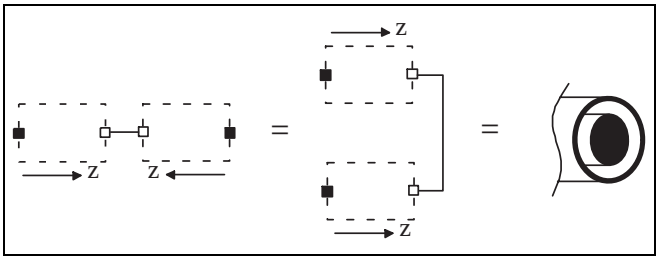


Bild 9: Verschaltung von Welle mit Hohlwelle.

Alle vektoriellen Größen in einem Flansch werden bezüglich des Flansch-Koordinatensystems dargestellt, z.B. Schnittmoment $\vec{\tau} = \tau \cdot \vec{e}_z$ bzw. Winkelgeschwindigkeit $\vec{\omega} = \omega \cdot \vec{e}_z$, wobei $\vec{e}_z$ ein Einheitsvektor in Richtung der positiven z-Achse ist und τ bzw. ω die skalaren Größen sind, die in der Schnittstelle benutzt werden. Eine Verbindung von zwei oder mehr Flanschen (= **connect**-Anweisung bei Modelica) bedeutet nun, daß die lokalen Flansch-Koordinatensysteme übereinstimmen.

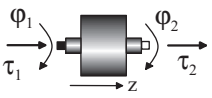
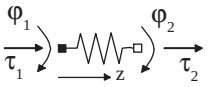
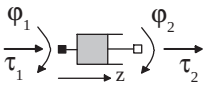
Bei der Verschaltung von Antriebsstrang-Komponenten muß man etwas vorsichtig sein: Wenn die z-Achsen der Flansche bei einer Verschaltung nicht gleichgerichtet sind, siehe linker Teil von Bild 9, dann ist das zulässig, entspricht aber der Verschaltung von einer Hohlwelle mit einer Welle wie durch Umzeichnen im rechten Teil von Bild 9 zu sehen ist.

Basierend auf den erläuterten mechanischen Schnittstellen für Stellantriebe, können z.B. die einfachen mechanischen Komponenten von Tabelle 8 erstellt werden.

Einfache Rohrströmungen

Es werden hier nur Schnittstellen für 1-dimensionale, 1-phasige, 1-Stoff Strömungen betrachtet. Diese Annahmen sind oft für technische Leitungen mit dem Medi-

Tabelle 8: Objektgleichungen idealer Stellantriebskomponenten.

Drehträgheit		$\varphi_1 = \varphi_2$ $\dot{\varphi}_1 = \omega_1$ $J \cdot \dot{\omega}_1 = \tau_1 + \tau_2$
Elastizität		$0 = \tau_1 + \tau_2$ $\tau_2 = c \cdot (\varphi_2 - \varphi_1)$
Dämpfung		$0 = \tau_1 + \tau_2$ $\tau_2 = d \cdot (\dot{\varphi}_2 - \dot{\varphi}_1)$

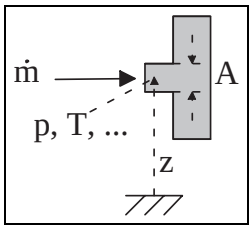


Bild 10: Schnittstelle für einfache Rohrströmungen.

um Wasser, Hydrauliköl, Wasserdampf, Luft (Pneumatik) erfüllt. Der physikalische Zustand in der (abstrahierten) Schnittstelle, siehe Bild 10, wird durch die folgenden Variablen beschrieben: Massenstrom $\dot{m}$, Geschwindigkeit v, Höhe z, Querschnittsfläche A, Druck p, Temperatur T, Dichte ρ, spez. innere Energie u, spez. Enthalpie h und spez. Entropie s. Der einströmende Energiefluß berechnet sich zu (siehe z.B. [25]):

$$P = \dot{m} \cdot \left(h + \frac{v^2}{2} + g \cdot z \right),$$

wobei g die Gravitationskonstante ist. Die thermodynamischen Variablen (p, T, ρ, u, h, s) werden bei einem 1-phasigen 1-Stoff Fluid eindeutig durch zwei dieser Größen charakterisiert. Die anderen Variablen können hieraus berechnet oder durch Interpolation aus Tabellen ermittelt werden. Hier werden Temperatur T und Druck p ausgewählt, da diese Größen am anschaulichsten sind, leicht gemessen werden können und Stoffgesetze häufig bezüglich dieser beiden Zustandsgrößen angegeben sind. Die Konstanten im Energiefluß, d.h. Höhe z und Querschnittsfläche A, werden auch in die Schnittstelle aufgenommen. In Modelica werden Konstante in einer Schnittstelle nicht verschaltet. Stattdessen muß der Wert von Konstanten in jeder Instanz einer Schnittstelle angegeben werden. Als *Flußvariable* enthält die Schnittstelle den Massenstrom. Die Geschwindigkeit v der Strömung wird, so benötigt, berechnet:

$$\dot{m} = \rho \cdot A \cdot v \;\Rightarrow\; v = \frac{\dot{m}}{\rho(p, T) \cdot A}$$

Bei Mehrphasen-Strömungen kann die Temperatur nicht mehr als Zustandsgröße gewählt werden, da diese im Mehrphasengebiet konstant ist. Stattdessen wird dann oft die spezifische Enthalpie als Schnittstellen-Variable benutzt. In einigen technisch wichtigen Fällen kann die Schnittstelle auf Grund weiterer Annahmen vereinfacht werden, z.B. bei hydraulischen Systemen.

Die hier diskutierte Schnittstelle ist nur für einfache Modelle geeignet. In einem späteren Artikel dieser Serie wird erläutert, wie komplexere Thermo-Fluid Systeme basierend auf der finiten Volumenmethode behandelt werden können.

3.3 Komplexe Schnittstellen

Komplexere Schnittstellen sollten hierarchisch aus den im vorigen Abschnitt besprochenen Elementar-Schnittstellen aufgebaut werden. Z.B. könnte die Schnittstelle zwischen Flügel und Triebwerk eines Flugzeugs in Modelica Notation den folgenden Aufbau besitzen

```
connector Engine
  Frame a  "3D-Mechanik Verbindung";
  Pin   p  "elektrische Versorgung";
  Hyd   h  "hydraulische Versorgung";
    ...
end Engine;
```

wobei `Frame, Pin, Hyd` vordefinierte Elementar-Schnittstellen sind.

Literatur

[17] *Cellier, F. E.:* Continuous System Modeling. Springer Verlag, New York, 1991.

[18] *Falk, G.* und *Ruppel, W.:* Energie und Entropie. Springer-Verlag, Berlin, 2. Auflage, 1980.

[19] *Fuchs, H.U.:* Dynamics of Heat. Springer Verlag, 1996.

[20] *Holman, J. P.:* Heat Transfer. McGraw–Hill, New York, 8. Auflage, 1996.

[21] *Job, G.: Neudarstellung der Wärmelehre*. Akademische Verlagsgesellschaft, Frankfurt am Main, 1972.

[22] *Karnopp, D. C., Margolis, D. L.* und *Rosenberg, R. C.:* System Dynamics: A Unified Approach. John Wiley, Cambridge, Mass., 2. Auflage, 1990.

[23] *Otter, M.* und *Elmqvist, H.:* Energy Flow Modeling of Mechatronic Systems via Object Diagrams. Proceedings of 2nd Mathmod Vienna, IMACS Symposium on Mathematical Modelling, S. 705–710, Wien, 1997.

[24] *Paynter, H. M.:* Analysis and Design of Engineering Systems. MIT Press, Cambridge, Mass., 1961.

[25] *Schade, H.* und *Kunz, E.:* Strömungslehre. Walter de Gruyter, 2. Auflage, 1989.

[26] *Stephan, K.* und *Mayinger, F.:* Thermodynamik. Band 2: Mehrstoffsysteme und chemische Reaktionen. Springer-Verlag, Berlin, 13. Auflage, 1992.

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 4

Martin Otter, DLR Oberpfaffenhofen

4 Transformationsalgorithmen

In Teil 1–3 wurden die Grundlagen der *objektorientierten Modellierung* physikalischer Systeme skizziert und an einfachen Systemen dargestellt, wie diese Technik praktisch eingesetzt werden kann. Im vorliegenden Kapitel werden nun die wesentlichen *Algorithmen* erläutert, mit denen ein objektorientiertes Modell in eine effizient auswertbare Form, wie die Zustandsform, *automatisch* transformiert werden kann.

4.1 Ziel der Transformation

Im ersten Schritt der Transformation wird ein Objektdiagramm, bzw. das in einer Modellierungssprache wie Modelica beschriebene System, in ein differential-algebraisches Gleichungssystem (abgekürzt DAE, für „Differential Algebraic Equation system“) umgewandelt. Hierzu werden die *lokalen Gleichungen* aller *Komponenten*, sowie die Gleichungen aller *Verbindungen* zu einem Gleichungssystem zusammengefaßt. Dies wurde in Teil 1 an einem einfachen Beispiel gezeigt. Die auftretenden Variablen werden folgendermaßen katalogisiert und zu Vektoren zusammengefaßt:

p	*Parameter*	Als **parameter** deklariert.
u(t)	*Eingang*	Auf *oberster* Modellebene als **input** deklariert.
x(t)	*Zustand*	Tritt differenziert auf.
y(t)	*Algebraisch*	Alle anderen Variablen.

Damit wird das Gesamtmodell in die DAE

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{p}, t), \quad \left[\frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}} : \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right] \text{ regulär} \qquad (4.1)$$

überführt, wobei *vorerst* die Annahme getroffen wird, daß die Jacobimatrix bezüglich $\dot{\mathbf{x}}$ und $\mathbf{y}$ regulär ist. Auf Grund des Satzes über implizite Funktionen ist dies eine *notwendige* und *hinreichende* Bedingung, um die DAE (4.1) durch rein *algebraische* Umformungen, ohne Differentation oder Integration, zumindest numerisch in die *Zustandsform*

$$\dot{\mathbf{x}} = \mathbf{f}_1(\mathbf{x}, \mathbf{u}, \mathbf{p}, t) \qquad (4.2a)$$

$$\mathbf{y} = \mathbf{f}_2(\mathbf{x}, \mathbf{u}, \mathbf{p}, t) \qquad (4.2b)$$

transformieren zu können. Wie Systeme behandelt werden, bei denen die Jacobi-Matrix *singulär* ist, wird im nächsten Artikel erläutert.

Eine rein numerische Lösung der DAE (4.1), die ja einfach alle Gleichungen des objektorientierten Modells in einer unstrukturierten Form enthält, ist für eine große Klasse von Systemen selbst bei Einsatz guter „Sparse-Matrix“ Verfahren *uneffizient*. Deswegen ist es in einem Folgeschritt das Ziel, die DAE (4.1) in die folgende *sortierte DAE* umzuwandeln:

$$\begin{bmatrix} \mathbf{0} \\ \dot{\mathbf{x}}^e \\ \mathbf{y}^e \end{bmatrix} = \begin{bmatrix} \mathbf{f}^r(\dot{\mathbf{x}}^i, \mathbf{x}, \mathbf{y}^i, \mathbf{u}, \mathbf{p}, t) \\ \mathbf{f}^x(\dot{\mathbf{x}}^i, \mathbf{x}, \mathbf{y}^i, \mathbf{u}, \mathbf{p}, t) \\ \mathbf{f}^y(\dot{\mathbf{x}}^i, \mathbf{x}, \mathbf{y}^i, \mathbf{u}, \mathbf{p}, t) \end{bmatrix} \qquad (4.3)$$

Das heißt, die DAE soll in einen *implizit* und in einen *explizit* lösbaren Teil umgeformt werden. Hierbei werden die Vektoren $\mathbf{x}$ und $\mathbf{y}$ 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:

$$\begin{bmatrix} \mathbf{x}^i \\ \mathbf{x}^e \end{bmatrix} = \mathbf{P}^x \mathbf{x}, \quad \begin{bmatrix} \mathbf{y}^i \\ \mathbf{y}^e \end{bmatrix} = \mathbf{P}^y \mathbf{y} \qquad (4.4)$$

Für eine numerische Lösung von (4.3) 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. Bei der objektorientierten Modellierung gibt es viele algebraische Variablen, da z.B. in der Regel alle Schnittstellen-Variablen rein algebraisch sind, so daß die Ordnungsreduktion oft beträchtlich ist.

Weiterhin können *konsistente* Anfangsbedingungen durch Vorgabe von $\mathbf{x}(t_0)$ und Vorgabe von Schätzwerten für $\dot{\mathbf{x}}^i(t_0), \mathbf{y}^i(t_0)$ berechnet werden, da nur das nichtlineare Gleichungssystem $\mathbf{f}^r(..)$ nach $\dot{\mathbf{x}}^i(t_0), \mathbf{y}^i(t_0)$ gelöst werden muß und die anderen Anfangswerte $\dot{\mathbf{x}}^e(t_0), \mathbf{y}^e(t_0)$ explizit aus $\mathbf{f}^x(..), \mathbf{f}^y(..)$ berechnet werden können.

Die sortierte DAE (4.3) kann auch mit einem Integrator für Zustandssysteme (4.2a), z.B. einem Runge-Kutta Verfahren, gelöst werden. Dann muß eine Funktion zur Verfügung gestellt wird, mit der die Ableitung des Zustandsvektors $\dot{\mathbf{x}}$, bei gegebenem Zustand $\mathbf{x}$, berechnet wird. In dieser Funktion werden die impliziten Variablen $\dot{\mathbf{x}}^i, \mathbf{y}^i$ durch *numerisches* Lösen von $\mathbf{0} = \mathbf{f}^r(..)$ bestimmt. Mit $\dot{\mathbf{x}}^e = \mathbf{f}^x(..)$ werden dann die restlichen Zustandsableitungen berechnet. Wenn das implizite Teilsystem linear in den Unbekannten ist, muß nur ein lineares Gleichungssystem gelöst werden. Dies ist unproblematisch. Das Modell kann dann auch für *Echtzeit-Simulationen* eingesetzt werden, da die Rechenzeit für einen Funkti-

onsaufruf immer gleich ist und z.B. ein Euler-Verfahren zur Lösung verwendet werden kann.

Die DAE (4.1) wird mittels einer *symbolischen* Transformation in die sortierte DAE (4.3) umgeformt. Ist der implizite Teil in (4.3) immer noch „groß“ (z.B. $\dim(\mathbf{f}^r) > 100$) sollten auf jeden Fall zur weiteren Lösung *numerische* „Sparse-Matrix“ Verfahren eingesetzt werden, siehe z.B. [28]. Die vorstehenden Überlegungen können folgendermaßen zusammengefaßt werden:

Ein Objektdiagramm wird in ein nichtlineares Gleichungssystem der Form

$$\mathbf{0} = \mathbf{f}(\mathbf{z}, \mathbf{x}, \mathbf{u}, \mathbf{p}, t), \quad \text{mit } \mathbf{z} = \begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{y} \end{bmatrix} \tag{4.5}$$

überführt. Ziel ist es, dieses Gleichungssystem durch algebraische, symbolische Umformungen möglichst explizit nach den Unbekannten $\mathbf{z}$ *aufzulösen.*

4.2 BLT-Transformation

Der wichtigste Algorithmus zur Lösung dieser Aufgabenstellung ist die *Block-Lower-Triangular* Transformation (abgekürzt: BLT-Transformation). Hier wird durch Permutation der Gleichungen und Permutation der Unbekannten versucht (4.5) in eine *rekursiv auflösbare* Form zu transformieren, siehe z.B. [30, 28, 33]. Die Grundidee wird an dem folgenden Beispiel mit 5 Gleichungen in den 5 Unbekannten $z_1, \ldots, z_5$ erläutert:

$$\begin{array}{ll} f_1(z_3, z_4) & = 0 \\ f_2(z_2) & = 0 \\ f_3(z_2, z_3, z_5) & = 0 \\ f_4(z_1, z_2) & = 0 \\ f_5(z_1, z_3, z_5) & = 0 \end{array} \qquad \mathbf{S}_1 = \begin{array}{c} \begin{matrix} z_1 & z_2 & z_3 & z_4 & z_5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

Die Gleichungsstruktur wird durch die Inzidenzmatrix $\mathbf{S}_1$ wiedergegeben, die anzeigt, ob die k-te Variable ($= k$-te Spalte) in der i-ten Gleichung ($= i$-te Zeile) auftritt oder nicht. Durch Permutation der Gleichungen und Variablen, bzw. durch Permutation von Zeilen und Spalten von $\mathbf{S}_1$, kann dieses Gleichungssystem auf BLT-Form transformiert werden:

$$\begin{array}{ll} f_2(\underline{z_2}) & = 0 \\ f_4(\underline{z_1}, z_2) & = 0 \\ f_3(z_2, \underline{z_3, z_5}) & = 0 \\ f_5(z_1, \underline{z_3, z_5}) & = 0 \\ f_1(z_3, \underline{z_4}) & = 0 \end{array} \qquad \mathbf{S}_2 = \begin{array}{c} \begin{matrix} z_2 & z_1 & z_3 & z_5 & z_4 \end{matrix} \\ \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 0 & 0 & 0 \\ 1 & 0 & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 1 & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 1 & 0 & \mathbf{1} \end{bmatrix} \end{array}$$

Auf Grund der unteren Block-Dreiecksform von $\mathbf{S}_2$ können die Funktionen rekursiv nach den unterstrichenen Variablen gelöst werden, die den Diagonalelementen von $\mathbf{S}_2$ entsprechen. Damit wird zuerst z_2 aus f_2 und z_1 aus f_4 berechnet. f_3, f_5 müssen simultan nach z_3, z_5 aufgelöst werden. Schließlich wird z_4 aus f_1 berechnet. Wenn die aufzulösenden Variablen *linear* in den jeweiligen Gleichungen auftreten (z.B. z_1 linear in f_4), können diese Gleichungen *explizit* gelöst werden, so daß f_3, f_5 den impliziten und f_2, f_4, f_1 den expliziten Teil von (4.3) bilden.

Block-Dreiecksformen mit *minimalen* Dimensionen der Diagonalblöcke werden als *BLT-Form* bezeichnet. D.h. es ist *nicht* möglich durch Permutationen von Variablen und Gleichungen auf algebraische Gleichungssysteme geringerer Dimension zu transformieren. Die BLT-Form kann mit einem effizienten Algorithmus der Ordnung $O(nm)$ berechnet werden, wobei n die Zahl der Gleichungen und m die Zahl der „Einsen“ in der Inzidenzmatrix ist, siehe z.B. [28]. In vielen praktischen Fällen ist der Algorithmus $O(n)$, d.h. linear in der Zahl der Gleichungen, und kann auf einem PC in wenigen Sekunden Systeme mit $n = 10000...50000$ transformieren. Im folgenden wird dieser Algorithmus näher erläutert.

Das Zuordnungsproblem

In einem *ersten* Schritt wird ermittelt, nach welcher Variablen eine Gleichung aufgelöst werden muß (z.B. f_1 nach z_4 im obigen Beispiel). Dies kann als Zuordnungsproblem angesehen werden, welches mit dem folgenden rekursiven Algorithmus (nach [34]) gelöst wird:

```
Algorithmus 4.1 (Zuordnungsproblem)

assign(j) := 0, j=1,2,..,n
for <alle Gleichungen i=1,2,..,n>
   vMark(j) := false, j=1,2,..,n;
   eMark(j) := false, j=1,2,..,n;
   if not pathFound(i), "singulär";
end for
```

Es werden drei globale Felder benutzt: **assign**$(j) = i$ gibt an, daß Gleichung i nach der Variablen j aufzulösen ist. Wenn $i = 0$, wurde für Variable j noch keine zugeordnete Gleichung ermittelt. Die Bool'schen Felder **vMark** und **eMark** werden benutzt, um zu markieren, welche Variablen (vMark) und welche Gleichungen (eMark) schon untersucht wurden. Die Zuordnung für *eine* Gleichung wird mit der zentralen Hilfsfunktion **pathFound** ermittelt:

```
Algorithmus 4.2 (Zuordnung einer Gleichung)

function success = pathFound(i)
  eMark(i) = true;
  if <assign(j)=0 für eine Variable j
      von Gleichung i> then
     success   := true;
     assign(j) := i;
  else
     success := false;
     for <jede Variable j von Gleichung i
          mit vMark(j) = false>
        vMark(j) := true;
        success  := pathFound( assign(j) );
        if success then
           assign(j) := i;
           return
        end if
     end for
  end if
end
```

Wenn z.B. die ersten 4 Gleichungen des Beispiels schon untersucht wurden, gibt es die folgenden Zuordnungen auf der linken Seite von (4.6) (durch [] gekennzeichnet):

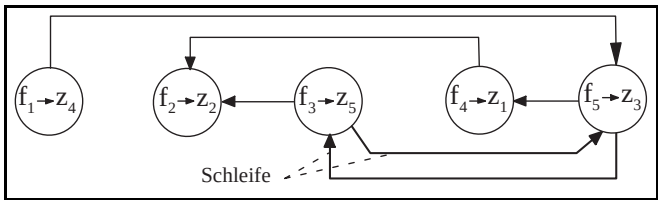


Bild 11: Gerichteter Graph des Beispiels.

$$\begin{array}{lll} f_1([z_3], z_4) & = 0 & \\ f_2([z_2]) & = 0 & \\ f_3(z_2, z_3, [z_5]) & = 0 & \Rightarrow \\ f_4([z_1], z_2) & = 0 & \\ f_5(z_1, z_3, z_5) & = 0 & \end{array} \quad \begin{array}{lll} f_1(z_3, [z_4]) & = 0 & \\ f_2([z_2]) & = 0 & \\ f_3(z_2, z_3, [z_5]) & = 0 & \\ f_4([z_1], z_2) & = 0 & \\ f_5(z_1, [z_3], z_5) & = 0 & \end{array} \tag{4.6}$$

Jetzt soll die Zuordnung der verbleibenden Gleichung f_5 bestimmt werden. Alle Variablen von f_5 haben schon eine Zuordnung, so daß der erste **`if`**-Zweig von der Funktion **pathFound** fehlschlägt. Im **`else`**-Zweig wird jetzt versucht, die bisher gefundene Zuordnung so zu ändern, daß eine der Variablen von f_5 ihre Zuordnung verliert: Die erste Variable z_1 von f_5 ist f_4 zugeordnet. Der Versuch z_2 als Zuordnung bei f_4 zu benutzen schlägt fehl, da f_2 nur die Zuordnung z_2 zuläßt. Die zweite Variable z_3 von f_5 ist f_1 zugeordnet. Hier kann statt z_3 auch z_4 als Zuordnung benutzt werden, so daß z_3 als Zuordnung von f_5 benutzt werden kann. Als Resultat erhält man die rechte Seite von (4.6).

Wenn die Funktion **pathFound** auf oberster Ebene (in Algorithmus 4.1) als Rückgabewert **false** liefert, gibt es keine Zuordnung zu Gleichung i, und die durch **eMark** markierten Gleichungen sind *strukturell singulär* (Beweis siehe [34], S. 217). D.h. gleichgültig wie die markierten Funktionen aufgebaut sind, bilden diese ein Gleichungssystem, welches keinen vollen Zeilenrang besitzt, so daß die Voraussetzung (4.1) verletzt ist und die DAE durch algebraische Umformungen nicht in Zustandsform transformiert werden kann. Durch Ausgabe der markierten Gleichungen kann dem Anwender mitgeteilt werden, welche Modellteile zu der Singularität führen.

Schleifen eines gerichteten Graphen

Nachdem nun *jeder* Gleichung eindeutig *eine* Variable zugeordnet ist, kann das Gleichungssystem als *gerichteter* Graph dargestellt werden. Hierbei repräsentiert jeder Knoten des Graphen eine Gleichung f_i mit der zugeordneten Variablen z_j wobei $i =$ **assign**(j), d.h. die Variable z_j wird an diesem Knoten mittels der Gleichung f_i berechnet. Eine Kante $(z_j \rightarrow z_k)$ gibt an, daß die Variable z_k von der Gleichung f_i benötigt wird, um die Variable z_j zu berechnen, siehe Bild 11, welches den Graph zu der Zuordnung (4.6) darstellt. Die Aufgabe besteht jetzt darin, die *Schleifen* (= „strong components“) des so konstruierten gerichteten Graphen zu ermitteln. Diese entsprechen den algebraischen Gleichungssystemen minimaler Dimension, bzw. den Diagonalblöcken der BLT-Form, da die Variablen in den Knoten einer Schleife nicht rekursiv berechnet werden können, sondern voneinander abhängen. Diese Aufgabe wird am effizientesten mit dem Algorithmus von Tarjan [35] gelöst. Nachdem die Schleifen identifiziert sind, liegt ein azyklischer Graph vor (mit den Schleifen als „Superknoten“), der rekursiv ausgewertet werden kann. Damit ist die Auswertungs-Reihenfolge der Gleichungen festgelegt und die BLT-Form bestimmt.

4.3 Tearing

Durch die Transformation auf BLT-Form werden die algebraischen Schleifen minimaler Dimension effizient ermittelt. Für viele physikalische Systeme sind diese Schleifen jedoch immer noch unnötig groß. Die Dimension der Schleifen kann durch intelligente *Variablen-Substitution* weiter verringert werden. Diese Verfahrensart ist unter unterschiedlichen Namen in vielen Fachgebieten bekannt und wurde wohl zum ersten Mal von Kron [32] dargestellt. Das Verfahren wird hier als *Tearing* bezeichnet. Es gibt viele Varianten. Die einfache Grundidee wird an einem Beispiel erläutert:

$$\begin{aligned} z_1 &= f_1(z_5) \\ z_2 &= f_2(z_1) \\ z_3 &= f_3(z_1, z_2) \\ z_4 &= f_4(z_2, z_3) \\ z_5 &= f_5(z_4, z_1) \end{aligned}$$

Dieses Gleichungssystem kann durch BLT-Transformation nicht in kleinere, voneinander unabhängige Gleichungssysteme aufgespalten werden. D.h. das Ausgangssystem ist schon in der BLT-Form. Wenn jedoch z_1 an allen Stellen durch f_1, z_2 durch f_2, z_3 durch f_3 und z_4 durch f_4 ersetzt wird, kann auf ein System mit *einer* Gleichung in der Unbekannten z_5 transformiert werden:

$$\begin{aligned} z_5 &= f_5(f_4(f_2(f_1(z_5)), f_3(\ldots)), f_1(z_5)) \\ z_1 &:= f_1(z_5) \\ z_2 &:= f_2(z_1) \\ z_3 &:= f_3(z_1, z_2) \\ z_4 &:= f_4(z_2, z_3) \end{aligned}$$

Die Variable z_5 wird aus der ersten Gleichung bestimmt und danach die restlichen Variablen durch eine einfache Vorwärtsrekursion. Diese *direkte* Vorgehensweise hat den entscheidenden Nachteil, daß z.B. die Funktionen f_1 5-Mal und f_2 3-Mal ausgewertet werden. Das Tearing Verfahren führt zu einer effizienteren Lösung, in dem die Residuen-Gleichung $0 = \bar{f}_5(z_5)$ rekursiv berechnet wird (man beachte, daß z_5 Eingangsargument der Funktion f_5 ist und damit eine bekannte Größe):

$$\begin{aligned} z_1 &:= f_1(z_5) \\ z_2 &:= f_2(z_1) \\ z_3 &:= f_3(z_1, z_2) \\ z_4 &:= f_4(z_2, z_3) \\ \bar{f}_5 &:= f_5(z_4, z_1) - z_5 \end{aligned}$$

Auf diese Weise wird auch auf ein Gleichungssystem mit *einer* Gleichung in der Unbekannten z_5 transformiert, jedoch wird jede Funktion f_i nur *einmal* ausge-

wertet. Wenn $\bar{f}_5$ linear von z_5 abhängt, kann auf ein *lineares* System transformiert werden.

Eine Schwierigkeit besteht jetzt darin, die Residuen-Gleichungen (hier: f_5) und die Tearing-Variablen (hier: z_5) so zu bestimmen, daß die Dimension des Zielsystems möglichst klein wird. In [27] wird gezeigt, daß diese Aufgabe zu einem nicht-polynomialen Algorithmus führt, im wesentlichen also nur durch Ausprobieren aller Varianten ermittelt werden kann. Dies ist auf Grund der exponentiell anwachsenden Zahl von Möglichkeiten nicht praktikabel. Deswegen gibt es nur heuristische Vefahren. Weiterhin müssen die Residuen-Gleichungen und Tearing-Variablen so bestimmt werden, daß sich der *numerische* Rang des Systems nicht ändert, siehe das folgende Beispiel:

$$\begin{aligned} z_1 &= \sin(\omega t) \\ z_1 \cdot z_2 &= \sin(z_3) \\ 0 &= z_2^2 - z_3 \end{aligned}$$

Durch BLT-Transformation wird festgestellt, daß mit der ersten Gleichung zuerst z_1 berechnet wird und daß z_2, z_3 simultan aus den letzten beiden Gleichungen berechnet werden müssen. Mittels des Tearing-Verfahrens könnte man die 2. Gleichung nach z_2 auflösen und in die dritte_Gleichung einsetzen, also das Gleichungssystem $0 = f_3(z_3)$ lösen mit:

$$\begin{aligned} z_2 &:= \sin(z_3)/z_1 \\ \bar{f}_3 &:= z_2^2 - z_3 \end{aligned}$$

Während der Simulation kann jedoch z_1 verschwinden, wobei dann durch Null dividiert wird Um solche Fälle auszuschließen führen die üblichen Tearing-Algorithmen zur Sicherstellung der Regularität eine Pivotisierung durch, siehe z.B. [28, 33]. Dann ist es aber nicht mehr möglich die Tearing-Transformation *einmal* im voraus symbolisch durchzuführen, sondern diese muß bei *jeder* Modellauswertung in der Simulation neu bestimmt werden.

In [31] wird zum ersten Mal gezeigt, wie das Tearing-Verfahren in der objektorientierten Modellierung für eine *symbolische* Transformation unter bestimmten Voraussetzungen eingesetzt werden kann. In [27] wird die Eigenschaft ausgenutzt, daß in der objektorientierten Modellierung die *Details* der Gleichungen bekannt sind. Dann kann durch eine geeignete Kandidatenauswahl *garantiert* werden, daß die Transformation *immer* regulär ist. Hierbei werden zur Substitution nur solche Variablen ausgewählt, die linear in einer Gleichung auftreten, wobei der Vorfaktor eine nicht-verschwindende Konstante ist. Im obigen Beispiel würde deswegen nur z_3 in Gleichung 3 als Kandidat für eine Tearing-Variable in Frage kommen, so daß auf das Gleichungssystem $0 = \bar{f}_2(z_2)$ mit:

$$\begin{aligned} z_3 &:= z_2^2 \\ \bar{f}_2 &:= z_1 \cdot z_2 - \sin(z_3) \end{aligned}$$

transformiert wird, welches denselben numerischen Rang wie das Ausgangssystem hat. Diese Vorgehensweise hat den entscheidenden Vorteil, daß die Transformation im voraus erfolgen kann und der Suchprozeß zum Auffinden von Tearing-Variablen und Residuen-Gleichungen nicht in jedem Integrationsschritt wiederholt werden muß. Basierend auf dieser Grundidee, wird in [27] ein einfaches heuristisches Verfahren angegeben, wie Tearing-Variablen und Residuen-Gleichungen bestimmt werden können. Dieses Verfahren wurde von H. Elmqvist und M. Otter für das objektorientierte Modellierungssystem Dymola [29], Version 3.1, weiter verbessert. Die Effektivität kann an folgendem Beispiel demonstriert werden: Eine Kette von 30 Körpern, die durch je um 90^o versetzte Drehgelenke miteinander verbunden sind, führt bei einer Modellierung als Objektdiagramm auf eine DAE mit ca. 3600 Gleichungen. Durch BLT-Transformation wird eine rekursiv auflösbare Gleichungsstruktur ermittelt, die eine lineare algebraische Schleife mit 1200 Gleichungen enthält. Diese Schleife wird mit dem skizzierten Tearing-Algorithmus auf die von der Mechanik her bekannte Minimal-Dimension 30 reduziert. Die komplette Transformation benötigt auf einem 200 MHz PC rund 2 Sekunden.

4.4 Zusammenfassung

Ein Objektdiagramm kann sehr einfach in eine DAE überführt werden. Diese kann durch BLT-Transformation und Tearing in eine sortierte DAE transformiert werden, in der möglichst viele Zustandsableitungen und algebraische Variable explizit berechenbar sind. Die sortierte DAE kann dann mit Standard-Integrationsverfahren, eventuell unter Verwendung von numerischen Sparse-Matrix Methoden, gelöst werden.

Literatur

[27] *Carpanzano, E.* und *Girelli, R.:* The Tearing Problem: Definition, Algorithm and Application to Generate Efficient Computational Code from DAE Systems. Proceedings of 2nd Mathmod Vienna, IMACS Symposium on Mathematical Modelling, Wien, 1997.

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

[29] Dymola. Homepage: http://www.dynasim.se

[30] *Elmqvist, H.:* A Structured Model Language for Large Continuous Systems. PhD dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Schweden, 1978.

[31] *Elmqvist, H.* und *Otter, M.:* Methods for Tearing Systems of Equations in Object-Oriented Modeling. Proceedings ESM'94 European Simulation Multiconference, S. 326–332, Barcelona, Spanien, 1994.

[32] *Kron, G.:* Diakoptics – The piecewise Solution of Large-Scale Systems. MacDonald & Co., London, 1962.

[33] *Mah, R. S. H.:* Chemical Process Structures and Information Flows. Butterworths Verlag, 1990.

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

[35] *Tarjan, R. E.:* Depth First Search and Linear Graph Algorithms. SIAM Journal of Comp., Nr. 1, S. 146–160, 1972.

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 5

Martin Otter, Oberpfaffenhofen und Bernhard Bachmann, Baden

Dr.-Ing. Martin Otter ist wissenschaftlicher Mitarbeiter beim DLR Institut für Robotik und Systemdynamik, leitet dort das Teilprogramm "Robotsystemdynamik" und ist Lehrbeauftragter an der TU München. Hauptarbeitsgebiete: Steuerung und Regelung von Industrierobotern, Antriebsstrangmodellierung für Echtzeitanwendungen, objektorientierte Modellierung.

Adresse: Deutsches Zentrum für Luft- und Raumfahrt, Institut für Robotik und Systemdynamik, Postfach 1116, D-82230 Weßling, Tel.: 08153/28-2473, Email: Martin.Otter@DLR.de

2

Dr. phil., Dipl. math. Bernhard Bachmann ist Fachassistent für Simulationstechniken bei ABB Power Automation Ltd. und verantwortlich für die Test- und Entwicklungsumgebung neuer Schutzalgorithmen. Hauptarbeitsgebiete: Simulation von Elektroenergiesystemen, numerische Behandlung dynamischer Systeme, Neuronale Netzwerke, objektorientierte Modellierung.

Adresse: ABB Power Automation Ltd., Abteilung für Schutz- und Stationsleittechnik, Haselstrasse 16/122, CH-5401 Baden/Switzerland, Tel.: +41 56 205 76 79, Email: Bernhard.Bachmann@chpau.mail.abb.com

5 Singuläre Systeme

5.1 Einführung

In Kapitel 1-4 wurden die Grundlagen der *objektorientierten Modellierung* physikalischer Systeme skizziert und insbesondere gezeigt, wie ein Objekdiagramm in eine DAE[1] der Form

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) \tag{5.1}$$

abgebildet wird. Hierbei sind **x**(t) Variablen, deren erste Ableitungen $\dot{\mathbf{x}}$ in den Modellgleichungen auftreten, und **y**(t) sind rein algebraische Größen. Weiterhin wurden Algorithmen erläutert um die Ausgangs-DAE (5.1) in die effizienter auswertbare Zustandsform

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{y} \end{bmatrix} = \mathbf{f}_1(\mathbf{x}, t) \tag{5.2}$$

oder in die *sortierte* DAE-Form (siehe Teil 4) zu transformieren. Dies ist genau dann möglich, wenn die Jacobi-Matrix von (5.1) bezüglich $\dot{\mathbf{x}}(t)$ und $\mathbf{y}(t)$ zu jedem Zeitpunkt *t regulär* ist. Diese in den früheren Kapiteln verwendete Voraussetzung wird jetzt fallengelassen. D.h. jetzt werden Systeme (5.1) untersucht, bei denen diese Jacobi-Matrix *singulär* ist:

$$\left| \frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}} \vdots \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right| = 0 \tag{5.3}$$

[1] DAE steht für Differential-Algebraic Equations.

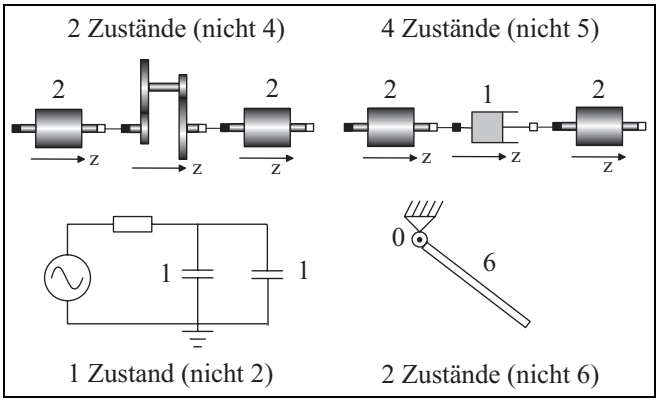


Bild 12: Beispiele für singuläre Systeme.

Im folgenden werden DAEs (5.1) mit der Eigenschaft (5.3) als *singuläre Systeme* bezeichnet.

Die Eigenschaft (5.3) bedeutet anschaulich, daß die beschreibenden Variablen **x** voneinander abhängig sind[2]. Damit kann eine singuläre DAE zwar nicht in die Zustandsform (5.2) transformiert werden, jedoch ist eine Transformation in die Zustandsform (5.4)

$$\begin{bmatrix} \dot{\mathbf{x}}^s \\ \mathbf{x}^d \\ \mathbf{y} \end{bmatrix} = \mathbf{f}_2(\mathbf{x}^s, t) \quad \text{mit} \quad \mathbf{x} = \mathbf{P} \begin{bmatrix} \mathbf{x}^s \\ \mathbf{x}^d \end{bmatrix} \tag{5.4}$$

(zumindest lokal numerisch) möglich, wobei der Zustandsvektor $\mathbf{x}^s$ aus den Elementen des "unabhängigen" Teils von **x** besteht, $\mathbf{x}^d$ die restlichen Elemente von **x** enthält, und **P** die zugeordnete Permutationsmatrix ist. Der Vektor $\mathbf{x}^d$ wird im folgenden als Dummy-Zustandsvektor bezeichnet. Man beachte, daß $\dot{\mathbf{x}}^d$ in (5.4) nicht mehr auftritt, sondern eliminiert wurde. Im allgemeinen können Zustände $\mathbf{x}^s$ nur für einen bestimmten Zeitbereich verwendet werden, so daß während einer Simulation eventuell zwischen unterschiedlichen Zustandsvektoren zu schalten ist.

In Bild 12 sind Beispiele für singuläre Systeme angegeben, wobei angenommen wird, daß jede Komponente durch ein *lokales* Objekt beschrieben wird. Im linken unteren Teil von Bild 12 gibt es einen elektrischen Schaltkreis, bei dem 2 Kapazitäten parallel geschaltet sind. Jede Kapazität besitzt den Spannungsabfall über

[2] Bei singulären DAEs können auch die algebraischen Variablen **y** untereinander, sowie von **x**, abhängig sein, d.h. es gibt redundante oder sich widersprechende Gleichungen. Dann besitzt ein Anfangswertproblem der DAE aber keine *eindeutige* Lösung mehr, so daß dieser Sonderfall (vorerst) nicht weiter betrachtet wird.

die beiden Klemmen als (lokale) Zustandsgröße (siehe auch Teil 1). Da die anderen Elemente keine Zustände besitzen, sollte auf eine Zustandsform mit 2 Zuständen transformiert werden können. Auf Grund der Parallelschaltung sind die Spannungsabfälle der beiden Kapazitäten jedoch gleich, so daß der Schaltkreis in Wirklichkeit nur 1 Zustand besitzt.

Im oberen Teil von Bild 12 gibt es zwei Antriebsstränge wobei im linken Teil die beiden trägheitsbehafteten Wellen durch ein ideales Getriebe und im rechten Teil durch einen einfachen Dämpfer verbunden sind. Jede Welle hat mit dem Drehwinkel und der Winkelgeschwindigkeit 2 Zustände. Der Dämpfer besitzt die Relativdrehzahl als Zustand (siehe Tabelle 9). Demnach sollte der linke Antriebsstrang 4 und der rechte 5 Zustände besitzen. Auf Grund der Verschaltungsart besitzt der linke Antriebsstrang jedoch nur 2 und der rechte nur 4 Zustände.

Schließlich wird im unteren rechten Teil von Bild 12 ein Einfachpendel gezeigt. Wird das Pendel durch ein ebenes Starrkörpermodell beschrieben, hat die Pendelstange 6 Zustände (2 Translationen und 1 Rotation je auf Positions- und Geschwindigkeitsebene) und das ideale Drehgelenk keinen Zustand. Das Gesamtsystem hat jedoch nur 2 und nicht 6 Zustände, da das Drehgelenk die Freiheitsgrade der Stange einschränkt.

Alle Beispiele von Bild 12 führen auf eine singuläre Jacobi-Matrix (5.3). Bei der objektorientierten Modellierung tritt dieses Phänomen recht häufig auf, da jede Komponente durch lokale Gleichungen beschrieben wird und durch das Zusammenschalten der Komponenten leicht Zwangsbedingungen zwischen den *lokalen* Zuständen der Objekte entstehen können. Ein objektorientiertes Modellierungssystem sollte solche Modelle deswegen automatisch *effizient* abhandeln können.

Die Behandlung von singulären Systemen ist ein aktueller Gegenstand der Forschung. Ein allgemein akzeptierter Standardzugang zeichnet sich noch nicht ab. Insbesondere gibt es kein Verfahren um das Anfangswertproblem einer *beliebigen* DAE zu lösen. Im vorliegenden Teil 5 und im Teil 6 dieser Serie wird die auf dem Pantelides Algorithmus [46] basierende Dummy Derivative Methode [38, 43] beschrieben. Hierbei werden singuläre Systeme durch Differentation von Gleichungen der DAE so aufbereitet, daß auf die Zustandsform (5.4) transformiert werden kann. Mit der Dummy Derivative Methode kann die zur Zeit wohl größte Klasse von singulären DAEs behandelt werden und das nicht-triviale Problem der Bestimmung von *konsistenten* Anfangsbedingungen[3], zumindest im Prinzip, gelöst werden. Das Verfahren wird in unterschiedlichen Varianten z.B. in den Programmen ABACUSS [36], Dymola [39] und Omola [45] eingesetzt.

Alternativen sind die *direkten* numerischen Lösungsverfahren, siehe z.B. [37, 40], bei denen die DAE (5.1) durch unmittelbares Anwenden numerischer Integrationsverfahren gelöst wird. Diese Methoden haben sich z.B. bei der Lösung von großen elektronischen Schaltkreisen bewährt. Für ein allgemeines objektorientiertes

[3] Dieser Punkt wird in Kapitel 5.4 erläutert.

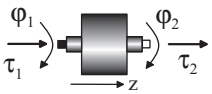
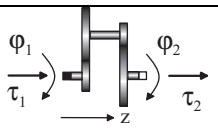
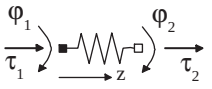
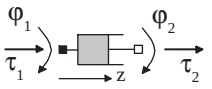
Drehträgheit		$\varphi_1 = \varphi_2$ $\varphi_1 = \omega_1$ $J \cdot \dot{\omega}_1 = \tau_1 + \tau_2$
Ideales Getriebe		$\varphi_1 = i \cdot \varphi_2$ $0 = i \cdot \tau_1 + \tau_2$
Elastizität		$0 = \tau_1 + \tau_2$ $\tau_2 = c \cdot (\varphi_2 - \varphi_1)$
Dämpfung		$0 = \tau_1 + \tau_2$ $\Delta\varphi = \varphi_2 - \varphi_1$ $\tau_2 = d \cdot \Delta\dot{\varphi}$

Tabelle 9: Objektgleichungen 1D mechanischer Komponenten.

Modellierungssystem scheinen direkte Verfahren jedoch weniger gut geeignet zu sein, da (a) nur spezielle Klassen von singulären DAEs behandelt werden können[4], (b) vorausgesetzt wird, daß *konsistente* Anfangsbedingungen für (5.1) vorliegen, was im allgemeinen nicht der Fall ist, und (c) direkte Verfahren auf Grund der iterativen Lösungsstrategie für Echtzeit-Anwendungen schwierig einzusetzen sind.

5.2 Der Index einer DAE

Zur Charakterisierung von DAEs beim Einsatz von *direkten numerischen* Lösungsverfahren ist der *Index* einer DAE gebräuchlich. Für nichtlineare DAEs (5.1) gibt es eine ganze Reihe *unterschiedlicher* Index-Definitionen, insbesondere:

- Der *differentielle Index*, siehe z.B. [37], gibt an, wie oft die DAE (5.1) differenziert werden muß, um auf
$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{y}} \end{bmatrix} = \mathbf{f}_3(\mathbf{x}, \mathbf{y}, t) \tag{5.5}$$
transformieren zu können. Eine DAE hat den differentiellen Index j, wenn j-mal zu differenzieren ist.
- Beim *Störungsindex*, siehe z.B. [40], wird die Differenz zwischen der exakten Lösung von (5.1) und der exakten Lösung der leicht gestörten DAE
$$\varepsilon(t) = \mathbf{f}(\hat{\mathbf{x}}, \hat{\mathbf{x}}, \hat{\mathbf{y}}, t) \tag{5.6}$$
herangezogen, um näherungsweise durch $\varepsilon(t)$ Fehler in der approximierten, numerischen Lösung zu beschreiben. Wenn diese Differenz eine Funktion der j-ten Ableitung von $\varepsilon(t)$, d.h. von $d^j\varepsilon/dt^j$, ist, hat die DAE den Störungsindex $j+1$.
- Beim *Traktabilitäts Index*, siehe z.B. [42, 44], wird die Index-Definition für lineare DAEs mit konstanten Koeffizienten auf die um den aktuellen Zeitpunkt linearisierte DAE angewandt.

Abgesehen von bestimmten Klassen von DAEs, wie z.B. linearen DAEs mit konstanten Koeffizienten, führen diese Index-Definitionen im allgemeinen zu unterschiedlichen Ergebnissen. Eine Charakterisierung der Form "mit

[4] Zum Beispiel haben die meisten direkten Verfahren Schwierigkeiten Systeme wie den Antriebsstrang im linken oberen Teil und das Pendel im rechten unteren Teil von Bild 12 zu lösen, wenn eine objektorientierte Modellierung verwendet wird.

dem Integrator XYZ können DAEs bis zum Index 2 gelöst werden“, ist deswegen nur bedingt aussagekräftig, da (a) nicht klar ist auf welche Index-Definition sich diese Formulierung bezieht, (b) es für einen Anwender in der Regel nicht-trivial ist den Index einer vorliegenden DAE zu ermitteln und (c) der Integrator in der Regel nicht in der Lage ist festzustellen, ob eine DAE in die lösbare Problemklasse fällt.

Es gibt keinen einfachen Zusammenhang zwischen dem Index einer DAE und einer singulären Jacobi-Matrix (5.3). Zum Beispiel haben die Gleichungen (5.7) und (5.8) jeweils einen differentiellen Index, Störungsindex und Traktabilitäts Index von eins.

$$\dot{x}_1 + \dot{x}_2 = -x_1 \quad (5.7a) \qquad \dot{x} + y = -x \quad (5.8a)$$
$$x_2 = 0 \quad (5.7b) \qquad y = 0 \quad (5.8b)$$

Das System (5.8) hat jedoch eine reguläre Jacobi-Matrix, kann also mit den Methoden von Teil 4 behandelt werden, während das System (5.7) singulär ist. Zusammengefaßt kann festgehalten werden, daß die Jacobi-Matrix (5.3) die Aussage erteilt, ob eine DAE in die Zustandsform mit $\mathbf{x}$ als Zustandsvektor transformiert werden kann, und daß der Index die Schwierigkeiten einer *numerischen* Lösung charakterisiert.

5.3 Ein einführendes Beispiel

Die Dummy Derivative Methode soll zuerst an einem einfachen Beispiel demonstriert werden. Hierzu wird eine kleine Bibliothek für 1D rotatorische, mechanische Komponenten erstellt, siehe Tabelle 9. Als Schnittstellenvariablen werden, wie in Kapitel 3 erläutert, der Drehwinkel φ und das Zwangsmoment τ an der Schnittstelle bezüglich des lokalen Schnittstellen-Koordinatensystems verwendet. Diese Bibliothek wird zur Modellierung des Antriebsstrangs von Bild 13, bestehend aus Antriebsmoment `u`, Welle `W1`, Getriebe `G`, Welle `W2`, Elastizität `F` und Welle `W3` benutzt. Wie in Kapitel 1 gezeigt, erstellt ein objektorientiertes Modellierungssystem aus dem Objektdiagramm von Bild 13 unter zuhilfenahme der Bibliothek von Tabelle 9 zuerst das folgende Gesamtgleichungssystem:

W1	$\dot{\varphi}_1 = \omega_1$	(5.9a)
	$J_1 \cdot \dot{\omega}_1 = u - \tau_1$	(5.9b)
G	$\varphi_1 = i \cdot \varphi_2$	(5.9c)
	$0 = i \cdot \tau_1 - \tau_2$	(5.9d)
W2	$\dot{\varphi}_2 = \omega_2$	(5.9e)
	$J_2 \cdot \dot{\omega}_2 = \tau_2 - \tau_3$	(5.9f)
F	$\tau_3 = -c \cdot (\varphi_3 - \varphi_2)$	(5.9g)
W3	$\dot{\varphi}_3 = \omega_3$	(5.9h)
	$J_3 \cdot \dot{\omega}_3 = \tau_3$	(5.9i)

Aus Gründen der Übersichtlichkeit sind in dem Gleichungssystem alle Variablen durchnummeriert (z.B. φ_3 statt W3.φ_1) und die trivialen Verbindungsgleichungen sind schon substituiert. Die Gleichungen (5.9) bilden eine DAE (5.1) mit $\mathbf{x}_1 = [\varphi_1\ \omega_1\ \varphi_2\ \omega_2\ \varphi_3\ \omega_3]^T$ und $\mathbf{y}_1 = [\tau_1\ \tau_2\ \tau_3]^T$. Wie in Kapitel 4 erläutert, kann die Transformation in die Zustandsform (5.2) durchgeführt werden, in dem bei *bekannt* angenommenem Zustandsvektor $\mathbf{x}_1$,

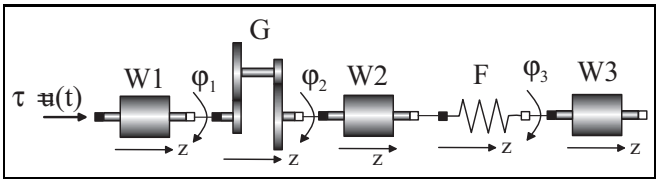


Bild 13: Objektdiagramm eines Antriebsstrangs.

aus den 9 Gleichungen (5.9) die 9 Unbekannten $\dot{\mathbf{x}}_1, \mathbf{y}_1$ ermittelt werden (genau das ist die Aussage von (5.2)).

Anschaulich ist klar, daß das nicht möglich ist, da die Kopplung der Wellen `W1` und `W2` über das ideale Getriebe `G` die Zahl der Freiheitsgrade reduziert, und damit die (lokalen) Zustände der beiden Wellen voneinander abhängig sind. Mathematisch kann dies daran gesehen werden, daß Gleichung (5.9c) keine der Unbekannten $\dot{\mathbf{x}}, \mathbf{y}$ enthält, sondern eine algebraische Beziehung zwischen den als bekannt angenommenen Zustandsgrößen φ_1 und φ_2 darstellt. Solche Gleichungen werden auch als *Zwangsgleichungen* bezeichnet. Damit ist die Jacobi-Matrix (5.3) *singulär* und die DAE (5.9) kann nicht in eine äquivalente Zustandsform mit $\mathbf{x}_1$ als Zustandsvektor transformiert werden.

Aus (5.9c) folgt, daß die beiden Größen φ_1 und φ_2 voneinander abhängig sind, und damit nur eine der beiden Variablen ein Zustand sein kann. Willkürlich wird angenommen, daß φ_1 eine (neue) *Unbekannte* und kein Zustand mehr ist und aus (5.9c) berechnet wird. Dann ist auch $\dot{\varphi}_1$ keine Ableitung eines Zustands mehr, sondern nur noch eine algebraische Größe (= Dummy Derivative), die zur Verdeutlichung nicht mehr mit einem Punkt, sondern mit einem Apostroph, φ_1', gekennzeichnet wird.

Da *eine* neue *Unbekannte* (= φ_1) eingeführt worden ist, muß gleichzeitig auch *eine* neue *Gleichung* hinzugenommen werden. Dies ist natürlicherweise die Ableitung von (5.9c), d.h.

$$\varphi_1' = i \cdot \dot{\varphi}_2 \quad (5.9c')$$

Die so konstruierte DAE besitzt 10 Gleichungen mit 10 Unbekannten. Die Jacobi-Matrix ist jedoch immer noch singulär, da

$$\varphi_1' = \omega_1 \quad (5.9a)$$
$$\dot{\varphi}_2 = \omega_2 \quad (5.9e)$$
$$\varphi_1' = i \cdot \dot{\varphi}_2 \quad (5.9c')$$

3 Gleichungen in den 2 Unbekannten φ_1' und $\dot{\varphi}_2$ ist. Es besteht also auch eine algebraische Beziehung zwischen den als bekannt angenommenen Zustandsgrößen ω_1, ω_2, da diese auch in den 3 Gleichungen auftreten. Wird festgelegt, daß ω_1 eine *Unbekannte* und kein Zustand mehr ist, so können die 3 Gleichungen regularisiert werden, da dann $\varphi_1', \dot{\varphi}_2, \omega_1$ aus diesen Gleichungen berechnet werden können. Wie zuvor müssen die Zwangsgleichungen differenziert werden:

$$\varphi_1'' = \omega_1' \quad (5.9a')$$
$$\varphi_2'' = \dot{\omega}_2 \quad (5.9e')$$
$$\varphi_1'' = i \cdot \varphi_2'' \quad (5.9c'')$$

Durch das Differenzieren ergeben sich 3 neue Gleichungen. Da ω_1 kein Zustand mehr ist, und da durch das Differenzieren die neuen Variablen φ_1'', φ_2'' auftre-

ten, gibt es jedoch auch *3 neue* Unbekannte. Damit entsteht eine DAE mit 13 Gleichungen in 13 Unbekannten. Diese DAE mit den Vektoren $\mathbf{x}_2 = [\varphi_2\ \omega_2\ \varphi_3\ \omega_3]^T$ und $\mathbf{y}_2 = [\tau_1\ \tau_2\ \tau_3\ \varphi_1\ \varphi_1'\ \varphi_1''\ \varphi_2''\ \omega_1\ \omega_1']^T$ hat nun eine Jacobi-Matrix die bezüglich $\dot{\mathbf{x}}_2, \mathbf{y}_2$ regulär ist. Durch BLT-Transformation, Tearing (siehe Kapitel 4) und Elimination trivialer Gleichungen der Form $a = b$ wird diese erweiterte DAE in die Zustandsform (5.4) mit $\mathbf{x}^s = \mathbf{x}_2$ transformiert:

$$\begin{aligned}
\varphi_1 &:= i \cdot \varphi_2 \\
\omega_1 &:= i \cdot \omega_2 \\
\tau_3 &:= -c \cdot (\varphi_3 - \varphi_2) \\
\dot{\omega}_2 &:= (i \cdot u - \tau_3)/(i^2 \cdot J_1 + J_2) \\
\dot{\omega}_3 &:= \tau_3 / J_3 \\
\dot{\varphi}_2 &:= \omega_2 \\
\dot{\varphi}_3 &:= \omega_3
\end{aligned}$$

Man beachte, daß die Auswahl der zu differenzierenden Gleichungen unabhängig davon ist, welche Zustände aus $\mathbf{x}_1$ ausgewählt werden, da

- die Zwangsgleichungen, welche eine algebraische Beziehung zwischen lokalen Zuständen beschreiben, immer differenziert werden, unabhängig davon welche Dummy-Zustände ausgewählt werden, und da
- die Wahl der Dummy-Zustände keine Auswirkung auf das restliche Gleichungssystem hat, da diese mittels der Zwangsgleichungen berechnet werden, also in den übrigen Gleichungen der DAE bekannte Größen sind, wie schon zuvor.

Damit können die Zustände bzw. Dummy-Zustände auch erst beim Vorliegen aller zu differenzierenden Gleichungen ausgewählt werden. Es wird sich noch zeigen, daß eine solche spätere Festlegung aus *numerischen* Gründen im allgemeinen notwendig ist.

5.4 Konsistente Anfangsbedingungen

Nach Vorgabe von Anfangsbedingungen $\mathbf{x}_2 = \mathbf{x}_2(t_0)$ hat die erweiterte DAE eine eindeutige Lösung, da sich die DAE in Zustandsform, mit $\mathbf{x}_2$ als Zustandsvektor, transformieren läßt. Wenn eine DAE (5.1) mit einem *direkten numerischen* Verfahren gelöst wird, muß die DAE auch zum Anfangszeitpunkt erfüllt sein. Dies bedeutet, daß der Anwender Anfangsbedingungen $\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0)$ so vorgeben muß, damit

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0), t_0) \tag{5.10}$$

Übertragen auf das obige Beispiel bedeutet dies, daß bei einer direkten numerischen Lösung von (5.9) Anfangsbedingungen $\dot{\mathbf{x}}_1(t_0), \mathbf{x}_1(t_0), \mathbf{y}_1(t_0)$ so vorgegeben werden müssen, daß (5.9) zum Anfangszeitpunkt erfüllt ist. Zum Beispiel wird (5.9) mit den Anfangsbedingungen

$$\begin{aligned}
\mathbf{x}_1(t_0) &= [0\ 1\ 0\ 0\ 0\ 0]^T \\
\dot{\mathbf{x}}_1(t_0) &= [1\ 0\ 0\ 0\ 0\ 0]^T \\
\mathbf{y}_1(t_0) &= [u(t_0)\ 0\ 0]^T
\end{aligned}$$

erfüllt. Sowohl anschaulich, als auch wegen (5.9*a*), (5.9*e*), (5.9*c'*), ist aber klar, daß es physikalisch unmöglich ist, daß $\omega_1(t_0) = 1$ und $\omega_2(t_0) = 0$ ist, d.h. daß die Welle 1 sich bewegt während Welle 2 aus dem Stillstand startet. Es kann also für diese *inkonsistenten* Anfangsbedingungen keine Lösung der DAE (5.9) geben. Pantelides hat in [46] gezeigt, daß die Anfangsbedingungen nicht nur die Ausgangs-DAE (5.1) erfüllen müssen, sondern auch diejenigen differenzierten Gleichungen von (5.1), die benötigt werden um die DAE in eine äquivalente Zustandsform (5.4) zu transformieren. Ist dies der Fall, wird von *konsistenten* Anfangsbedingungen gesprochen.

Mit anderen Worten, auch wenn ein Verfahren benutzt wird, mit dem die DAE (5.1) direkt numerisch gelöst wird, müssen im allgemeinen trotzdem die Zwangsgleichungen differenziert werden, da diese zum Anfangszeitpunkt erfüllt sein müssen! In [41] wird gezeigt, daß für *stationäre* Anfangsbedingungen (= die Zeit-Ableitungen aller Variablen verschwinden identisch zum Anfangszeitpunkt) von autonomen DAEs (= die DAE hängt nicht *explizit* von der Zeit ab) die differenzierten Zwangsgleichungen *automatisch erfüllt* sind. Für diesen wichtigen Sonderfall werden differenzierte Zwangsgleichungen beim Einsatz von direkten Verfahren deswegen *nicht* benötigt. Problematisch wird es jedoch auch hier, wenn unstetige oder strukturvariable DAEs behandelt werden sollen (siehe Kapitel 6 und 7), da dann die DAE an einer Unstetigkeit neu initialisiert werden muß, und konsistente, *nicht-stationäre* Anfangswerte benötigt werden.

Literatur

[36] *ABACUSS.* http://yoric.mit.edu/abacuss/abacuss.html.

[37] *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.

[38] *Cellier, F.E.* und *Elmqvist, H.:* Automated formula manipulation supports object-oriented continuous-system modeling. IEEE Control Systems Magazine, Vol. 13, S. 28–38, 1993.

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

[40] *Hairer, E.* und *Wanner, G.:* Solving Ordinary Differential Equations II, Stiff and Differential Algebraic Problems. Springer-Verlag, Berlin, 2. Auflage, 1996.

[41] *Kröner, A., Marquardt, W.* und *Gilles, E.D.:* Getting around Consistent Initialization of DAE Systems? Computers chem. Engineering, Vol 21, S. 145–158, 1997.

[42] *März, R.:* Numerical methods for differential-algebraic equations. Acta Numerica, S. 141–198, 1992.

[43] *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.

[44] *Matz, K.* und *Clauß, C.:* Simulation Support by Index Computation. Proc. 15th IMACS World Congress, Berlin, 24.-29. August, 1997.

[45] *Omola:* http://www.control.lth.se/~cace/omsim.html.

[46] *Pantelides, C.C:* The consistent initialization of differential-algebraic systems. SIAM Journal of Scientific and Statistical Computing, S. 213–231, 1988.

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 6

Herrn Prof. Dr.-Ing. Georg Grübel zum 60. Geburtstag gewidmet

Martin Otter, Oberpfaffenhofen und Bernhard Bachmann, Baden

(Fortsetzung von Kapitel 5 aus Teil 5)

5.5 Strukturell inkonsistente DAEs

In Kapitel 5.3 wurde am Beispiel eines Antriebsstranges gezeigt, wie durch Differentiation gewisser Gleichungen des Systems und durch Auswahl von Dummy-Zuständen eine singuläre DAE auf Zustandsform transformiert werden kann. Diese Vorgehensweise versagt jedoch z.B. bei folgendem System:

$$\dot{x} = f_1(x, y_1) \tag{5.12a}$$
$$0 = f_2(y_2) \tag{5.12b}$$
$$0 = f_3(y_2) \tag{5.12c}$$

Dies ist eine DAE mit singulärer Jacobi-Matrix, da die beiden letzten Gleichungen nur von der einen Unbekannten y_2 abhängen. Im Gegensatz zu Kapitel 5.3 ist es jetzt aber nicht möglich durch Auswahl geeigneter Dummy-Zustände eine reguläre Jacobi-Matrix zu erhalten, da (5.12b,5.12c) nicht von x abhängen. Damit kann (5.12) mit der erläuterten Methodik nicht auf Zustandsform transformiert werden. Anschaulich ist klar, daß dies auch generell nicht möglich ist, da die Gleichungen (5.12b,5.12c) entweder zueinander kompatibel sind (dann gibt es unendlich viele Lösungen), oder zueinander im Widerspruch stehen (dann gibt es keine Lösung), d.h. die DAE besitzt *keine eindeutige Lösung*. Diese Eigenschaft ist unabhängig davon, wie die beiden Funktionen f_2, f_3 letztendlich aufgebaut sind. In [51] werden solche DAEs als *strukturell inkonsistent* bezeichnet.

Es stellt sich die Frage, wie DAEs mit dieser Eigenschaft automatisch erkannt werden können, da z.B. durch einfache Modellierungsfehler des Anwenders solche DAEs enstehen können. Hierzu werde angenommen, daß die DAE

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) \tag{5.13}$$

mit einem impliziten Integrationsverfahren gelöst werden soll, d.h. daß $\dot{x}_i(t_j)$ als Funktion von $x_i(t_j)$ und (bekannten) Werten von x_i zu früheren Zeitpunkten angegeben wird:

$$\dot{x}_i = h_i(x_i) \tag{5.14}$$

Einsetzen von (5.14) in (5.13) ergibt:

$$\mathbf{0} = \mathbf{f}(\mathbf{h}(\mathbf{x}), \mathbf{x}, \mathbf{y}, t) \tag{5.15}$$

(5.15) ist ein nichtlineares Gleichungssystem in den unbekannten Variablen $\mathbf{x}, \mathbf{y}$ und hat auf Grund des Satzes über implizite Funktionen genau dann keine eindeutige Lösung, wenn die Jacobi-Matrix bezüglich $\mathbf{x}, \mathbf{y}$ singulär ist. Mit dem in Kapitel 4 erläuterten Algorithmus 4.1 zur Lösung des Zuordnungsproblems kann festgestellt werden, ob (5.15) strukturell singulär ist, d.h. ob dieses Gleichungssystem eine singuläre Jacobi-Matrix hat, unabhängig davon wie die Funktionen f_i aufgebaut sind (bei diesem Algorithmus wird nur die jeweilige funktionale Abhängigkeit von den Unbekannten berücksichtigt). Aus diesen Überlegungen ergibt sich nun die folgende Eigenschaft:

Eine DAE (5.13) wird als strukturell inkonsistent bezeichnet, wenn

$$\mathbf{0} = \mathbf{f}(\mathbf{x}, \mathbf{x}, \mathbf{y}, t), \quad \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}} \vdots \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right] \text{ strukt. singulär} \tag{5.16}$$

d.h. wenn alle Ableitungen $\dot{\mathbf{x}}$ in der DAE durch den Vektor $\mathbf{x}$ ersetzt werden und das hieraus resultierende Gleichungssystem mit Algorithmus 4.1 bezüglich $\mathbf{x}$ und $\mathbf{y}$ als strukturell singulär charakterisiert wird. Man beachte, daß die genaue funktionale Abhängigkeit von $\dot{x}_i$ und x_i in (5.14) nicht bekannt sein muß, da nur strukturelle Eigenschaften untersucht werden, so daß es in (5.13) genügt, $\dot{x}_i$ durch x_i (und nicht durch $h_i(x_i)$) zu ersetzen.

Auf Grund der Herleitung ist klar, daß strukturell inkonsistente DAEs nicht mit einem impliziten Integrationsverfahren gelöst werden können. Es kann gezeigt werden, daß die DAE dann auch keine eindeutige Lösung besitzt. Die Bedeutung von (5.16) liegt vor allem darin, daß der im nächsten Abschnitt erläuterte Pantelides Algorithmus genau dann konvergiert, wenn die DAE nicht strukturell inkonsistent ist (Beweis siehe [51], S. 219–221). Vor Anwendung des Pantelides Algorithmus muß demnach (5.16) überprüft werden.

5.6 Der Pantelides Algorithmus

Der Pantelides Algorithmus [51] wird benutzt um diejenigen *zusätzlichen* Gleichungen zu ermitteln, die für eine Transformation in die Zustandsform benötigt werden. Dies sind gleichzeitig auch diejenigen Gleichungen, die von konsistenten Anfangsbedingungen (neben der Ausgangs-DAE) erfüllt sein müssen, siehe Kapitel 5.4. Die Grundidee wurde an Hand eines Beispiels in Kapitel 5.3 skizziert und wird jetzt in allgemeinerer Form erläutert:

1. Es wird eine möglichst kleine Untermenge von Gleichungen aus (5.13) gesucht, bei der die *Zahl der Gleichungen größer ist als die Zahl der Unbekannten*, d.h. die Jacobi-Matrix dieser Untermenge hat keinen vollen Rang. Gleichungen dieser Art werden als Zwangsgleichungen bezeichnet. Als Unbekannte $\mathbf{z}$ werden die höchsten Ableitungen der beschreibenden Variablen betrachtet. Zu Beginn sind dies $\dot{\mathbf{x}}$ und $\mathbf{y}$.
2. Die im ersten Schritt gefundenen Zwangsgleichungen werden differenziert und zur DAE hinzugenommen. Eventuell durch das Differenzieren *neu* auftretende Unbekannte $\dot{\mathbf{z}}_{neu}$ werden zur Menge der unbekannten Variablen $\mathbf{z}$ hinzugenommen. In den Folgeschritten (siehe 1.) werden nur die höchsten Ableitungen, wie $\dot{\mathbf{z}}_{neu}$, als Unbekannte angesehen, da $\mathbf{z}_{neu}$ aus den Zwangsgleichungen berechnet wird und die Zwangsgleichungen deswegen in den nachfolgenden Untersuchungen nicht mehr betrachtet werden müssen. Weiterhin müssen aus den Zwangsgleichungen soviele Dummy-Zustände aus den auftretenden Elementen von $\mathbf{x}$ ausgewählt werden, bis die Gleichungen vollen Rang besitzen. Da diese Selektion aber keinen Einfluß auf die Wahl der zu differenzierenden Gleichungen hat, wird dieser Schritt auf einen späteren Zeitpunkt verschoben.
3. Die gesamte DAE, jedoch ohne die im 2. Schritt ermittelten Zwangsgleichungen, wird wieder bezüglich der höchsten auftretenden Ableitungen analysiert, d.h. es wird mit Schritt 1 fortgefahren. Der Algorithmus bricht ab, wenn keine Untermenge von Gleichungen mehr gefunden wird, bei der die Zahl der Gleichungen größer ist als die Zahl der Unbekannten. Dieser Abbruch findet nach einer endlichen Zahl von Schritten statt, wenn die Ausgangs-DAE die Eigenschaft (5.16) nicht besitzt.

Zur formalen Beschreibung des Algorithmuses ist es notwendig, (5.13) anders darzustellen. Mit $\mathbf{v} = [\mathbf{x}; \dot{\mathbf{x}}; \mathbf{y}]$, der *Variablen-Assoziierungsliste* V und der *Gleichungs-Assoziierungsliste* F

$$\begin{aligned} V(j) &= i, && \text{wenn } dv_j/dt = v_i \\ &= 0, && \text{sonst} \\ F(j) &= i, && \text{wenn } df_j/dt = f_i \\ &= 0, && \text{sonst} \end{aligned}$$

kann eine singuläre DAE (5.13) geschrieben werden als

$$\mathbf{0} = \mathbf{f}(\mathbf{v},t), \quad \left|\frac{\partial f_i}{\partial v_j}\right| = 0, \quad V(j) = 0 \qquad (5.17)$$

d.h. die DAE ist bezüglich ihrer höchsten Ableitungen singulär. Durch Differenzieren neu hinzukommende Gleichungen bzw. neu auftretende Unbekannte, werden in die entsprechenden Assozierungslisten eingetragen. Zur Identifikation der kleinsten strukturell singulären Untermengen wird der in Kapitel 4 beschriebene Algorithmus 4.2 (Funktion **pathFound**) bezüglich der *höchsten* auftretenden *Ableitungen* benutzt, d.h. bezüglich der Variablen v_j bei denen $V(j) = 0$ ist. Zusammengefaßt erhält man den folgenden Algorithmus:

Algorithmus 5.1 (Pantelides Algorithmus)

```
<Initialisiere Var.-Assoziierungsliste V>
assign(j) := 0, j=1,2,..,nv;
F(j)      := 0, j=1,2,..,nf;

for <Ausgangsgleichungen k = 1,2,...,nf>
  i := k;
  loop
    // Finde Zuordnung zu Gleichung i bzgl.
    // der höchsten Ableitung von v.
       vMark(j) := false, j=1,2,..,nv;
       eMark(j) := false, j=1,2,..,nf;
       if pathFound(i), exit loop;

    // Markierte Gleichungen sind singulär.
    // Differenziere Gleichungen und Var.
       for <Var. j, mit vMark(j) = true>
         nv := nv + 1;  V(j) := nv;
       end for
       for <Gl. j, mit eMark(j) = true>
         nf := nf + 1;  F(j) := nf;
         <füge Var.liste zu Gl. nf hinzu>
       end for

    // Kopiere Zuordnung zu diff. Gleichungen
       for <Var. j, mit vMark(j) = true>
         assign(V(j)) := F(assign(j));
       end for

    // Untersuche differenzierte Gleichungen
       i := F(i);
  end loop
end for
```

Wie schon in Kapitel 4 erläutert, gibt **assign**$(j) = i$ an, daß Gleichung i nach der Variablen j aufzulösen ist. Die Bool'schen Felder **vMark** und **eMark** werden benutzt um zu markieren, welche Variablen (vMark) und welche Gleichungen (eMark) schon untersucht wurden. Nach Beendigung von Algorithmus 5.1 enthält die Gleichungs-Assoziierungsliste F alle Gleichungen, die differenziert werden müssen um das System in Zustandsform transformieren zu können. Der Pantelides Algorithmus ist effizient mit einer maximalen Zahl von $O(n \cdot m)$ Operationen, wobei n die Zahl der Gleichungen und m die Summe der in allen Gleichungen auftretenden Variablen des erzeugten *Endgleichungssystems* ist.

Die vom Pantelides Algorithmus ermittelte Gleichungs-Assoziierungsliste F legt das folgende System von Gleichungen fest:

$$0 = f_i(\mathbf{v},t), \quad \frac{\partial f_i}{\partial v_j} \text{ strukturell regulär} \qquad (5.18a)$$

$$0 = f_k(\mathbf{v},t), \quad \frac{\partial f_k}{\partial v_j} = 0 \qquad (5.18b)$$

$$F(i) = 0, \; F(k) > 0, \; V(j) = 0 \qquad (5.18c)$$

(5.18c) definiert die in (5.18a), (5.18b) verwendeten Indices i, j, k. (5.18a) sind die n Gleichungen mit den höchsten Ableitungen in F, wobei n die Dimension der Ausgangs-DAE (5.13) ist. Dieses Gleichungssystem ist strukturell regulär bezüglich der höchsten Ableitungen in $\mathbf{v}$, d.h. jede Gleichung f_i kann nach der eindeutig zugeordneten Variablen v_j aufgelöst werden. Die restlichen Gleichungen, f_k, stellen die Zwangsbedingungen zwischen den potentiellen Zuständen $\mathbf{x}$ dar, die zu einer Festlegung der Dummy-Zustände benützt werden.

In den Zwangsgleichungen treten die höchsten Ableitungen, v_j, nicht auf.

Der Pantelides Algorithmus beruht auf dem *hinreichenden* Kriterium, daß die Jacobi-Matrix eines Satzes von Gleichungen einen Rangabfall besitzt, wenn die Zahl der Gleichungen größer ist als die Zahl der Unbekannten. Es ist damit möglich, daß der Pantelides Algorithmus nicht alle zu differenzierenden Gleichungen findet, wie im folgenden konstruierten Beispiel:

$$\dot{x}_1 = y_1 \tag{5.19a}$$
$$\dot{x}_2 = y_2 \tag{5.19b}$$
$$0 = y_1 + y_2 + 2x_1 \tag{5.19c}$$
$$0 = y_1 + y_2 + x_1 + 2x_2 \tag{5.19d}$$

Um (5.19) in Zustandsform zu transformieren, müsste zuerst erkannt werden, daß (5.19d) äquivalent ist zu:

$$0 = x_1 - 2x_2 \tag{5.20}$$

Die Anwendung vom Pantelides Algorithmus auf (5.19a–5.19c,5.20) führt dann auf die gewünschten Gleichungen. Man beachte, daß eine kleine Änderung in irgendeiner der Vorfaktoren von y_1, y_2 in (5.19c,5.19d) diese DAE in eine reguläre DAE verwandelt, die mit den Methoden von Kapitel 4 auf Zustandsform transformiert werden kann!

5.7 Die Bestimmung der Dummy-Zustände

Um die weitere Vorgehensweise zu motivieren, wird das folgende Beispiel betrachtet, wobei $\mathbf{x} = [x_1 x_2 x_3]^T$, $\mathbf{y} = [y]$ und $u(t)$ eine gegebene Eingangsfunktion ist:

$$0 = y + x_1 \tag{5.21a}$$
$$0 = y + x_2 \tag{5.21b}$$
$$0 = y + x_3 \cdot u \tag{5.21c}$$
$$0 = \dot{x}_1 + \dot{x}_2 + \dot{x}_3 \tag{5.21d}$$

Der Pantelides Algorithmus stellt fest, daß (5.21a,5.21b) singulär sind, da dies 2 Gleichungen für die eine Unbekannte y ist, d.h. es gibt eine algebraische Beziehung zwischen den (als bekannt angenommenen) Zustandsgrößen x_1 und x_2. Willkürlich wird x_1 als *neue* Unbekannte, d.h. als Dummy-Zustand, eingeführt. Damit werden x_1 und y aus diesen beiden Gleichungen berechnet. In den weiteren Iterationen von Algorithmus 5.1 ist y eine bekannte Größe, da durch das Differenzieren die *neue* höchste Ableitung $\dot{y}$ auftritt. Dann ist (5.21c) singulär, da dies eine Gleichung ist, die nur *bekannte* Größen enthält. Hieraus ergibt sich, daß x_3 kein Zustand mehr sein kann, sondern aus (5.21c) berechnet werden muß. Dies ist der *kritische* Punkt: Bei der Umformung nach x_3 muß durch $u(t)$ dividiert werden. Verschwindet $u(t)$ während der Simulation, wird durch Null geteilt. Werden die drei ersten Gleichungen dagegen *zusammen* analysiert stellt man fest, daß es besser ist, wenn x_3 Zustandsgröße ist und x_1, x_2, y aus diesen Gleichungen berechnet werden, da dann die Zwangsgleichungen während der Simulation regulär bleiben!

Mattsson und Söderlind haben in [50] ein Verfahren angegeben, welches die erläuterte Schwierigkeit elegant vermeidet, und im folgenden diskutiert wird. Das Verfahren in [50] setzt voraus, daß die Jacobi-Matrix von (5.18*a*) zu jedem Zeitpunkt der Simulation *regulär* ist:

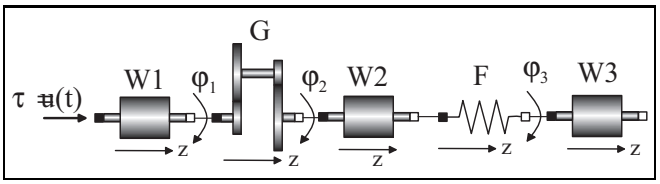


Bild 11: Objektdiagramm eines Antriebsstrangs.

$$\left| \frac{\partial f_i(\mathbf{v}, t)}{\partial v_j} \right| \neq 0, \quad F(i) = 0, \; V(j) = 0 \tag{5.22}$$

Unter dieser (in vielen Fällen zutreffenden) Annahme wird garantiert, daß auf Zustandsform transformiert werden kann. Man beachte, daß mit der bis jetzt benutzten Methodik keine solche Garantie gegeben werden kann, selbst wenn (5.22) zutrifft.

Aus Platzgründen wird der einleuchtende Algorithmus nicht allgemein erläutert, sondern gleich direkt an dem einführenden Beispiel eines Antriebsstrangs aus Kapitel 5.3 vorgeführt, siehe Bild 11. Auf Grund der Herleitung ist klar, daß der Pantelides Algorithmus für die DAE (5.9) genau die schon ermittelten Gleichungen (5.9a',c',c",e') ableitet. Das reguläre System (5.18*a*) ergibt sich dann zu:

W1	$\ddot{\varphi}_1 = \dot{\omega}_1$	(5.23a')
	$J_1 \cdot \dot{\omega}_1 = u - \tau_1$	(5.23b)
G	$\ddot{\varphi}_1 = i \cdot \ddot{\varphi}_2$	(5.23c'')
	$0 = i \cdot \tau_1 - \tau_2$	(5.23d)
W2	$\ddot{\varphi}_2 = \dot{\omega}_2$	(5.23e')
	$J_2 \cdot \dot{\omega}_2 = \tau_2 - \tau_3$	(5.23f)
F	$\tau_3 = -c \cdot (\varphi_3 - \varphi_2)$	(5.23g)
W3	$\dot{\varphi}_3 = \omega_3$	(5.23h)
	$J_3 \cdot \dot{\omega}_3 = \tau_3$	(5.23i)

D.h. das System besteht aus allen Gleichungen der Ausgangs-DAE, wobei die einzelnen Gleichungen jedoch teilweise ein- oder zweimal differenziert wurden. Für diese DAE wird eine BLT-Transformation (siehe Kapitel 4) bezüglich der höchsten auftretenden Ableitungen von $\mathbf{v}$, also bezüglich $\{\ddot{\varphi}_1, \ddot{\varphi}_2, \dot{\varphi}_3, \dot{\omega}_1, \dot{\omega}_2, \dot{\omega}_3, \tau_1, \tau_2, \tau_3\}$, durchgeführt. Wegen (5.18*a*) wird garantiert, daß das möglich ist. Dies führt auf eine BLT-Form mit 4 Diagonalblöcken B1,B2,B3,B4, die rekursiv gelöst wird:

B1	$[\dot{\varphi}_3] = \omega_3$	(5.24h)
B2	$[\tau_3] = -c \cdot (\varphi_3 - \varphi_2)$	(5.24g)
B3	$J_3 \cdot [\dot{\omega}_3] = \tau_3$	(5.24i)
B4	$J_1 \cdot \dot{\omega}_1 = u - [\tau_1]$	(5.24b)
	$J_2 \cdot [\dot{\omega}_2] = \tau_2 - \tau_3$	(5.24f)
	$0 = i \cdot \tau_1 - [\tau_2]$	(5.24d)
	$\ddot{\varphi}_1 = [\dot{\omega}_1]$	(5.24a')
	$[\ddot{\varphi}_2] = \dot{\omega}_2$	(5.24e')
	$[\ddot{\varphi}_1] = i \cdot \ddot{\varphi}_2$	(5.24c'')

In eckigen Klammern werden diejenigen Variablen markiert, nach denen aufgelöst werden muß, siehe Algorithmus 4.1. Der Block B4 besteht aus einem linearen Gleichungssystem. Unabhängig von der weiteren Vorgehensweise werden die höchsten Ableitungen aus (5.24) berechnet. Hierbei kann mit Tearing (siehe Kapitel 4) das Gleichungssystem von Block B4 noch auf ein System der Ordnung 1 reduziert werden.

In einem zweiten Schritt werden diejenigen Gleichungssysteme der BLT-Form untersucht, die *differenzierte* Gleichungen enthalten. In diesem Beispiel ist das nur Block B4. Dieser Block wird benutzt um die unbekannten Zustandsableitungen $\ddot{\varphi}_1, \ddot{\varphi}_2, \dot{\omega}_1, \dot{\omega}_2$, sowie die algebraischen Variablen τ_1, τ_2 zu berechnen. Die in Block B4 auftretenden Funktionsableitungen, d.h. (5.24a'), (5.24e'), (5.24c''), werden einmal integriert und führen auf die schon vom Pantelides Algorithmus ermittelten Zwangsgleichungen

$$\dot{\varphi}_1 = \omega_1 \qquad (5.25a)$$
$$\dot{\varphi}_2 = \omega_2 \qquad (5.25e)$$
$$\dot{\varphi}_1 = i \cdot \dot{\varphi}_2 \qquad (5.25c')$$

einer Untermenge von (5.18b). Damit sind die potentiellen Zustände $\dot{\varphi}_1, \dot{\varphi}_2, \omega_1, \omega_2$ nicht unabhängig voneinander. Durch Hinzufügen dieser 3 Gleichungen müssen wiederum 3 neue Unbekannte eingeführt werden, so daß 3 der 4 Größen Dummy-Zustände sind. Nach Voraussetzung ist (5.24) regulär, so daß (5.25) *zeilenregulär* sein muß, da die Ableitung von (5.25) Teil von (5.24) ist. Durch Vorgabe einer der Zustände können demnach die 3 anderen Dummy-Zustände aus (5.25) immer berechnet werden.

Die Bestimmung der Dummy-Zustände ist im allgemeinen nur numerisch möglich. Hierzu wird die Jacobi-Matrix der Zwangsbedingungen aufgestellt. Diese ergibt sich einfach aus der Ableitung der (im allgemeinen nichtlinearen) Zwangsbedingungen (5.25), d.h. aus den schon vorhandenen Gleichungen (5.24a'), (5.24e'), (5.24c''). Durch Wahl von n linear unabhängigen *Spalten* der n-zeiligen Jacobi-Matrix werden die zugehörigen Variablen als Dummy-Zustände festgelegt. Dies kann z.B. durch eine *QR-Zerlegung* mit *Spaltenpivotisierung* erfolgen, einem guten Kompromiß zwischen Effizienz und numerischer Robustheit. Während der Simulation muß im allgemeinen zwischen unterschiedlichen Dummy-Zuständen geschaltet werden. Details für eine numerische Umschaltstrategie sind in [48] zu finden.

Vorab können Dummy-Zustände, zumindest teilweise, mittels des in Kapitel 4 skizzierten Tearing-Verfahrens ermittelt werden. Bei der automatischen Wahl der Tearing-Variablen werden jeweils Gleichungen und Unbekannte aus einem Gleichungssystem so eliminiert, daß dadurch keine Änderung des Rangs auftritt. Angewandt auf (5.25) führt das dazu, daß $\dot{\varphi}_2$ Zustand bleibt und die anderen Größen, $\dot{\varphi}_1, \omega_1, \omega_2$ aus (5.25) berechnet werden.

Die erläuterte Vorgehensweise wird auf alle Diagonalblöcke der BLT-Form *rekursiv* solange angewendet, bis keine Zwangsgleichungen mehr vorhanden sind. Im obigen Beispiel gibt es auf Grund von (5.25c') eine weitere Zwangsbedingung

$$\varphi_1 = i \cdot \varphi_2 \qquad (5.26c)$$

so daß die potentiellen Zustände φ_1, φ_2 nicht unabhängig voneinander sind. Wiederum ist garantiert, daß (5.26c) zeilenregulär ist, da die Ableitung von (5.26c) Teil von (5.25) ist. Da $\dot{\varphi}_2$ Zustand ist, muß auch φ_2 Zustand sein und φ_1 wird aus (5.26c) berechnet.

Zusammengefaßt wird damit die singuläre DAE (5.9) in eine reguläre DAE überführt, die entweder numerisch in die Zustandsform transformiert, oder aber auch direkt numerisch gelöst werden kann. Im speziellen Beispiel ist eine Transformation in die Zustandsform sogar (einmal im voraus) symbolisch möglich. Der Pantelides Algorithmus zusammen mit der Dummy Derivative Methode hat den Vorteil, daß eine große Klasse von singulären DAEs (mit einem beliebigen Index) völlig automatisch behandelt werden kann. Der Nachteil besteht darin, daß der Pantelides Algorithmus in seltenen Fällen nicht immer alle zu differenzierenden Gleichungen findet.

Das Gleichungssystem (5.18) könnte prinzipiell direkt mit einem numerischen Integrationsverfahren gelöst werden. Hierzu müssten nur existierende Verfahren für DAEs mit spezieller Struktur, insbesondere *Projektionsverfahren* [49, 52], entsprechend verallgemeinert werden. Hierbei wird die reguläre DAE (5.18a) mit einem Standard-Integrationsverfahren gelöst und das Ergebnis nach jedem Integrationsschritt auf die durch (5.18b) beschriebenen Zwangsgleichungen projiziert. Da eine solche Verallgemeinerung jedoch noch nicht durchgeführt wurde, kann zur Zeit nur die Dummy-Derivative Methode verwendet werden.

Danksagung

Für hilfreiche Hinweise möchten wir Sven Erik Mattsson, Hilding Elmqvist, Martin Arnold und den beiden Gutachtern danken.

Literatur

[47] *Cellier, F.E.* und *Elmqvist, H.:* Automated formula manipulation supports object-oriented continuous-system modeling. IEEE Control Systems Magazine, Vol. 13, S. 28–38, 1993.

[48] *Feehery, W.* und *Barton, P.I.:* A Differentiation-Based Approach to Dynamic Simulation and Optimization with High-Index Differential-Algebraic Equations. Computational Differentiation, M. Berz, C. Bischof, G. Corliss und A. Griewank (Editors), SIAM, 1996.

[49] *Hairer, E.* und *Wanner, G.:* Solving Ordinary Differential Equations II, Stiff and Differential Algebraic Problems. Springer-Verlag, Berlin, 2. Auflage, 1996.

[50] *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.

[51] *Pantelides, C.C:* The consistent initialization of differential-algebraic systems. SIAM Journal of Scientific and Statistical Computing, S. 213–231, 1988.

[52] *Schulz, V.H., Bock, H.G.,* und *Steinbach, M.C.:* Exploiting invariants in the numerical solution of multipoint boundary value problems for DAE. SIAM Journal on Scientific Computing, Vol. 19, No. 2, pp. 440-467, 1998.

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 7

Martin Otter, Oberpfaffenhofen, Hilding Elmqvist, Lund und Sven Erik Mattsson, Lund

Dr.-Ing. Martin Otter ist wissenschaftlicher Mitarbeiter beim DLR und Lehrbeauftragter an der TU München. Hauptarbeitsgebiete: Steuerung und Regelung von Industrierobotern, Antriebsstrangmodellierung für Echtzeitanwendungen, objektorientierte Modellierung, Modelica Sprachentwurf.

Adresse: Deutsches Zentrum für Luft- und Raumfahrt, Institut für Robotik und Systemdynamik, Postfach 1116, D-82230 Weßling, Tel.: 08153/28-2473, Email: Martin.Otter@DLR.de

2

Dr. Hilding Elmqvist ist President der Firma Dynasim AB, Chairman der Modelica Design Group und Entwickler der Dymola und Simnon Sprachen. Hauptarbeitsgebiete: Wiederverwendbarkeit von Modellen, Modellierungsmethodik, Modelica Sprachentwurf.

Adresse: Dynasim AB, Research Park Ideon, S-22370 Lund, Schweden, Tel.: 0046/46/182500, E-mail: Elmqvist@Dynasim.se

3

Dr. Sven Erik Mattsson ist Senior Scientist bei der Firma Dynasim AB und Mitentwickler von Omola und Omsim. Hauptarbeitsgebiete: DAE Formalismus, symbolische Gleichungsverarbeitung, hybride Modellierung, Modelica Implementierung.

Adresse: Dynasim AB, Research Park Ideon, S-22370 Lund, Schweden, Tel.: 0046/46/182503, E-mail: SvenErik@Dynasim.se

6 Modelica — Hybride Systeme

In Kapitel 1-5 wurden die Grundlagen der objektorientierten Modellierung *kontinuierlicher* physikalischer Systeme erläutert und an einfachen Systemen demonstriert. Insbesondere wurde in Kapitel 2 exemplarisch die objektorientierte Modellierungssprache *Modelica* eingeführt und zur Modellierung von *kontinuierlichen* Systemen eingesetzt. Im vorliegenden Kapitel werden weitergehende Eigenschaften von Modelica erläutert, um unstetige und strukturvariable Systeme modellieren und simulieren zu können, wie z.B. Abtastsysteme, Begrenzer, Hysterese, ideale Dioden und Thyristoren, Coulomb Reibung, Lose oder Stöße. Hierzu werden zusätzliche Sprachelemente zur Modellierung von *ereignis-diskreten* Komponenten benötigt. Das Kapitel basiert auf [61, 60].

Für rein kontinuierliche Modellteile sind sich die objektorientierten Modellierungssprachen, wie Dymola, gPROMS, Modelica, Omola etc. recht ähnlich, da alle auf demselben Prinzip beruhen, Komponenten durch algebraische Gleichungen und Differentialgleichungen zu beschreiben. Für ereignis-diskrete Systeme gibt es jedoch keine allgemein akzeptierte Standardbeschreibungsform. Stattdessen liegt eine Vielzahl unterschiedlicher Formalismen vor, die meist auf ein bestimmtes Anwendungsfeld zugeschnitten sind, wie z.B. *endliche Automaten*, *Petri-Netze*, *Statecharts*, *Sequential Function Charts*, *Differenzengleichungen*, oder *prozessorientierte Sprachen*. Es ist deswegen nicht verwunderlich, daß sich die objektorientierten Modellierungssprachen bei der Beschreibung von diskreten Komponenten stark unterscheiden und es keine ”optimale“ Beschreibungsform geben kann, die alle Anwendungsgebiete abdeckt. Im Prinzip kann jeder diskrete Formalismus mit einer kontinuierlichen Systembeschreibung kombiniert werden, vorausgesetzt eine geeignete Synchronisierung der beiden Beschreibungsformen ist möglich.

In Modelica wird die für Echtzeitsteuerungen und die Verifikation diskreter Systeme entwickelte Technik der *synchronen Sprachen* [58, 55, 59, 57] eingesetzt, da sich diese gleichungsbasierte Beschreibung diskreter Systeme sehr elegant und einfach mit der kontinuierlichen Systembeschreibung kombinieren läßt. In [56] wurde diese Vorgehensweise zum ersten Mal auf hybride Systeme angewandt und in [60, 61] weiter ausgebaut.

6.1 Synchrone Systembeschreibung

Ein hybrides Modelica Modell besteht aus Differentialgleichungen, algebraischen und diskreten Gleichungen. Diskrete Gleichungen besitzen hierbei Boole’sche, Integer oder abgetastete Real Variable als Unbekannte. Ein typisches Beispiel ist in Bild 12 zu sehen, bei dem eine kontinuierliche Strecke

$$\dot{\mathbf{x}}_p = \mathbf{f}(\mathbf{x}_p, \mathbf{u}) \tag{6.1a}$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}_p) \tag{6.1b}$$

durch einen digitalen, linearen Regler

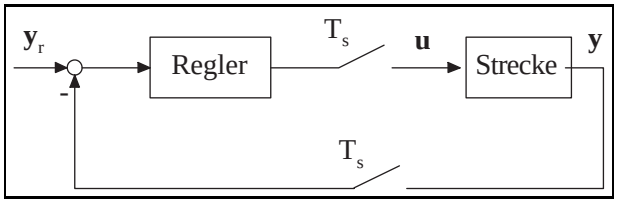


Bild 12: Abtastsystem.

$$\mathbf{x}_c(t_i) = \mathbf{A}\mathbf{x}_c(t_i - T_s) + \mathbf{B}\left(\mathbf{y}_r(t_i) - \mathbf{y}(t_i)\right) \quad (6.2a)$$
$$\mathbf{u}(t_i) = \mathbf{C}\mathbf{x}_c(t_i - T_s) + \mathbf{D}\left(\mathbf{y}_r(t_i) - \mathbf{y}(t_i)\right) \quad (6.2b)$$

geregelt wird. Hierbei ist T_s die Abtastzeit, $\mathbf{u}(t)$ das Stellsignal, $\mathbf{x}_p(t)$ der Zustandsvektor der Strecke, $\mathbf{y}(t)$ der Meßvektor, $\mathbf{x}_c(t_i)$ der Zustandsvektor des digitalen Reglers und $\mathbf{y}_r(t_i)$ das Reglersollsignal. An den Abtastzeitpunkten $t_i = i \cdot T_s,\ i = 0,1,2,\ldots$ werden die Meßsignale abgetastet, die Reglergleichungen ausgewertet und insbesondere die Steuergröße **u** berechnet, die bis zum nächsten Abtastzeitpunkt konstant gehalten wird, d.h. es wird ein Halteglied 0-ter Ordnung verwendet. In Modelica kann dieses System einfach als eine Verschaltung geeigneter Ein-/Ausgangsblöcke beschrieben werden (siehe Teil 2 dieser Serie). Zur Vereinfachung der Diskussion wird jedoch die folgende Gesamtbeschreibung in *einer* Modell-Klasse benutzt:

```
model SampledSystem
  parameter Real Ts=0.1 "Abtastzeit";
  parameter Real A[:, size(A,1)],
                 B[size(A,1), :],
                 C[:, size(A,2)],
                 D[size(C,1), size(B,2)];
  constant  Integer nx = 5;
  input  Real yr[size(B,2)]  "Sollsignal";
  output Real y [size(B,2)]  "Meßsignal";
         Real u [size(C,1)]  "Steuersignal";
         Real xc[size(A,1)]  "Reglerzustand";
         Real xp[nx]         "Streckenzust.";
equation
  der(xp) = f(xp, u);     // Strecke
      y   = g(xp);
  when sample(0,Ts) then  // Regler
     xc = A*pre(xc) + B*(yr-y);
     u  = C*pre(xc) + D*(yr-y);
  end when;
end SampledSystem;
```

Dieses Modelica-Modell besteht aus den kontinuierlichen Gleichungen der Strecke und den diskreten Gleichungen des Reglers innerhalb der **when** Anweisung. Wie in Teil 2 schon erläutert, charakterisiert **der**(xp) die Zeitableitung von xp. Während der kontinuierlichen Integration sind die Gleichungen innerhalb des **when** Rumpfes de-aktiviert. Wenn die Bedingung der **when** Anweisung wahr wird, wird ein *Ereignis* ausgelöst, die Integration angehalten und die Gleichungen im **when** Rumpf werden an diesem Ereignispunkt aktiviert. Der **sample**(...) Operator löst an den Abtastzeitpunkten i*Ts ($i = 0,1,2,\ldots$) Ereignisse aus und ist nur an diesen Zeitpunkten **true**. Die Werte von Variablen werden konstant gehalten, bis diese explizit neu berechnet werden. Zum Beispiel wird u nur zu den Abtastzeitpunkten berechnet. Zu anderen Zeitpunkten hat u den Wert der beim letzten Ereignis (= Abtastzeitpunkt) berechnet wurde.

Im digitalen Regler werden die Werte des Reglerzustands xc sowohl beim aktuellen Abtastzeitpunkt xc(t_i) als auch vom vorherigen Abtastzeitpunkt xc($t_i - T_s$) benötigt. Hierzu wird im obigen Modelica-Modell der **pre**(...) Operator benutzt. Formal wird der *linke Grenzwert* $x(t^-)$ einer Variablen x zum Zeitpunkt t durch **pre**(x) charakterisiert, während x den *rechten Grenzwert* $x(t^+)$ repräsentiert. Da xc nur an Abtastzeitpunkten unstetig ist, ist der linke Grenzwert xc(t_i^-) zum Abtastzeitpunkt t_i identisch zum rechten Grenzwert xc($t_i^+ - T_s$) beim vorherigen Abtastzeitpunkt, so daß **pre**(xc) diesen Wert charakterisiert.

Es stellt sich die Frage, in welcher Reihenfolge die Gleichungen des obigen Modells ausgewertet werden. In Modelica wird das *Synchronitätsprinzip* der synchronen Sprachen verwendet:

> *Zu jedem Zeitpunkt drücken die aktivierten Gleichungen Relationen zwischen Variablen aus, die gleichzeitig erfüllt sein müssen.*

Demnach müssen während der kontinuierlichen Integration die Gleichungen der Strecke und an Abtastzeitpunkten die Gleichungen der Strecke und die Gleichungen des Reglers *gleichzeitig* erfüllt sein, bilden also jeweils ein gemeinsames algebraisches Gleichungssystem, welches nach den unbekannten Variablen aufzulösen ist. Um solche Modelle *effizient* zu lösen, werden alle (kontinuierlichen und diskreten) Gleichungen mit Hilfe der in Teil 4 erläuterten BLT-Transformation unter der Annahme *sortiert*, daß *alle* Gleichungen *aktiv* sind. Mit anderen Worten, die Auswertungsreihenfolge der Gleichungen wird über eine Datenflußanalyse (automatisch) ermittelt. Zum Beispiel führt eine BLT-Transformation beim obigen Modelica Modell auf die folgenden Zuweisungen, die in der angegebenen Reihenfolge auszuwerten sind:

```
// bekannte Variablen: yr, xp, pre(xc)
y := g(xp);
when sample(0,Ts) then
   xc := A*pre(xc) + B*(yr-y);
   u  := C*pre(xc) + D*(yr-y);
end when;
der(xp) := f(xp, u);
```

Man beachte, daß diese Auswertungsreihenfolge sowohl korrekt ist, wenn nur die kontinuierlichen Gleichungen aktiv sind, als auch an einem Abtastzeitpunkt, wenn zusätzlich die diskreten Gleichungen des Reglers aktiv werden.

Das Synchronitätsprinzip hat eine Reihe von Konsequenzen: Zum einen müssen die aktiven diskreten und kontinuierlichen Gleichungen *gemeinsam* und *gleichzeitig* erfüllt sein, Zum anderen muß die Zahl der aktiven Gleichungen und die Zahl der unbekannten Variablen in diesen Gleichungen zu jedem Zeitpunkt *identisch* sein, damit die Unbekannten *eindeutig* berechnet werden können. Diese Forderung ist im folgenden Beispiel nicht erfüllt:

```
  Boolean close; // inkorrektes Modell
equation
  when h1 < 3 then
     close = true;
  end when
  when h2 > 1 then
     close = false;
  end when
```

In diesem Modell soll ein Ventil bei Vorliegen bestimmter Sensordaten entweder geöffnet (close = **true**) oder geschlossen (close = **false**) werden. Wenn nun beide when-Bedingungen (h1 < 3, h2 > 1) zufälligerweise oder auch beabsichtigt *gleichzeitig* wahr

werden, gibt es zwei miteinander in Konflikt stehende Gleichungen für `close` und es ist nicht definiert welche Gleichung verwendet werden sollte. Formal gibt es zwei Gleichungen für eine Unbekannte (= `close`), so daß es keine eindeutige Lösung geben kann.

In Modelica kann das obige Model einfach in eine korrekte Form überführt werden, in dem die in Konflikt stehenden *Gleichungen* in eine **`algorithm`** Sektion überführt und die Gleichungen in Zuweisungen umgewandelt werden:

```
    Boolean close;
algorithm
    when h1 < 3 then
        close := true;
    end when;
    when h2 > 1 then
        close := false;
    end when;
```

Alle Zuweisungen innerhalb *derselben* **`algorithm`** Sektion werden als eine Menge von *n* Gleichungen betrachtet, wobei *n* die Zahl der *unterschiedlichen* Variablen ist, die auf der linken Seite der Zuweisungen auftreten (z.B. entspricht die obige **`algorithm`** Sektion *einer* Gleichung für die Unbekannte `close`). Die so definierten Gleichungen einer **`algorithm`** Sektion werden als ein zusammenhängender Modellteil betrachtet, der immer als Ganzes mit den restlichen Gleichungen und anderen **`algorithm`** Sektionen sortiert wird. Innerhalb einer **`algorithm`** Sektion werden die Zuweisungen in der aufgeführten Reihenfolge ausgeführt. Aus diesem Grunde hat die zweite **`when`** Anweisung eine höhere Priorität und es gibt keine Konflikte mehr. Man beachte jedoch, daß eine zusätzliche Gleichung für `close` außerhalb der obigen **`algorithm`** Sektion wiederum zu einem Fehler führen würde, da es dann wieder Mehrdeutigkeiten bei der Berechnung von `close` geben würde. Mit anderen Worten, Mehrdeutigkeiten müssen vom Modellierer an Hand der physikalisch-technischen Gegebenheiten explizit aufgelöst werden.

Das in Modelica verwendete Synchronitätsprinzip zur Beschreibung von gemischt zeit-kontinuierlich und ereignis-diskreten Systemen hat den Vorteil, daß die "Synchronisierung" zwischen den kontinuierlichen und diskreten Modellteilen automatisch durch die Gleichungssortierung erfolgt, und daß ein korrektes Modelica Modell immer ein deterministisches Verhalten ohne Konflikte besitzt. Durch die einheitliche mathematische Darstellung können viele Probleme schon statisch an Hand des Quellcodes festgestellt werden.

Zum Beispiel entspricht dem "Deadlock" einer prozeßorientierten Sprache, einer algebraischen Schleife zwischen den Gleichungen von unterschiedlichen **`when`** Anweisungen (dies wird bei der Sortierung festgestellt). Wenn die **`when`** Anweisungen dieselben Schaltbedingungen besitzen, kann das Problem automatisch durch Lösen des Gleichungssystems aufgelöst werden. Ansonsten liegt ein Fehler vor, da es keine eindeutige Lösung gibt, wenn die Schaltbedingungen zu unterschiedlichen Zeitpunkten wahr werden.

Der Nachteil dieser Vorgehensweise besteht darin, daß es in einigen Anwendungen schwierig sein kann, ein diskretes System in einen Satz von synchronen, diskreten Gleichungen zu überführen. Weiterhin ist die Art der modellierbaren diskreten Systeme eingeschränkt. Zwar können in Modelica diskrete Formalismen, wie endliche Automaten, priorisierte Petri-Netze [62] und Statecharts mit einer speziellen Semantik, direkt realisiert werden; nicht beschreiben lassen sich jedoch z.B. allgemeine Petri-Netze (da diese ein nicht-deterministisches Verhalten besitzen) oder allgemeine prozess-orientierte Vorgänge (da durch das Erzeugen und Löschen von Prozessen im Quellcode nicht bekannt ist, welche Variablen während der Simulation auftreten). Nur durch die Verwendung von externen Funktionen, können solche diskreten Formalismen mit Modelica kombiniert werden.

6.2 Unstetige Systeme

Während der Integration der kontinuierlichen Systemteile müssen die Modellgleichungen stetig und differenzierbar sein, da alle numerischen Integrationsverfahren auf dieser Annahme basieren. Diese Voraussetzung wird leicht durch die Verwendung von **`if`**-Anweisungen verletzt. Zum Beispiel wird ein einfacher Zweipunktregler, siehe Bild 13, mit der Eingangsgröße *u* und der Ausgangsgröße *y* durch das folgende Modelica-Modell beschrieben:

```
model TwoPoint
   parameter Real y0=1;
   input     Real u;
   output    Real y;
equation
   y = if u > 0 then y0 else -y0;
end TwoPoint
```

Bei `u=0` würde es in den Modellgleichungen während der Integration eine Unstetigkeit geben, wenn die **`if`**-Anweisung wörtlich (wie in einer Programmiersprache) interpretiert werden würde. Potentiell können unstetige bzw. nicht-differenzierbare Punkte während der Integration auftreten, wenn eine *Relation* (wie z.B. $x_1 > x_2$) ihren Wert ändert, da dann z.B. der Zweig in einer **`if`**-Anweisung gewechselt wird. Eine solche Situation kann numerisch sicher durch ein *Zustandsereignis* (engl.: state event) behandelt werden. Hierbei wird der Zeitpunkt an dem sich der Wert einer Relation ändert innerhalb vorgegebener Grenzen relativ genau ermittelt, die Integration wird an diesem Zeitpunkt angehalten, es wird in den neuen Zweig der **`if`**-Anweisung gewechselt, und die Integration wird neu gestartet. Diese Technik wurde von Cellier [53] entwickelt. Details einer entsprechenden numerischen Realisierung sind auch in [54] zu finden.

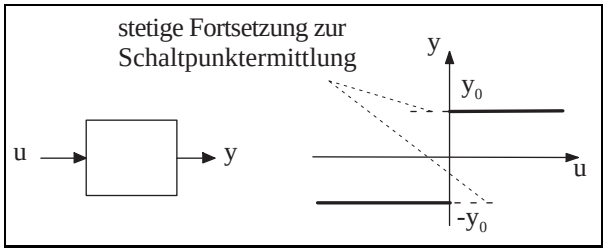


Bild 13: Unstetige Komponente.

Im allgemeinen ist es nicht möglich durch die Analyse des Quelltextes eines Modelica-Modells zu ermitteln, ob eine spezifische Relation zu einer Unstetigkeit führt oder nicht. Deswegen wird in Modelica standardmäßig angenommen, daß die Werte-Änderung einer Relation zu einem unstetigen oder nicht-differenzierbaren Punkt im Modell führt, die durch ein entsprechendes (automatisch ausgelöstes) Zustandsereignis numerisch korrekt abgehandelt wird. Das obige Modell eines Zweipunktreglers führt deswegen *nicht* zu einer Unstetigkeit während der Integration. Stattdessen wird an der Stelle `u = 0` ein Zustandsereignis ausgelöst an dem die Integration angehalten wird, bevor der **`if`**-Zweig gewechselt wird. Man beachte, daß während der Ermittlung des genauen Umschaltzeitpunktes der Relation (`u` $>$ `0`) der **`if`**-Zweig noch nicht geändert wird, so daß `y = y0` auch bei kleinen negativen Werten von `u` kurzfristig gilt, siehe Bild 13.

Der Modellierer kann eine wörtliche Interpretation einer Relation durch Verwendung des **`noEvent`**(...) Operators erzwingen, z.B. **`noEvent`**`(u > 0)`, d.h. dann wird kein Zustandsereignis ausgelöst. Dies ist sinnvoll, wenn der Wertewechsel von Relationen nicht zu unstetigen oder nicht-differenzierbaren Punkten führt. Selbst wenn solche Punkte vorhanden sind, kann der Effekt so klein sein, daß die Integration davon kaum beeinflußt wird, wenn einfach über diese Stelle hinweg integriert wird. Schließlich gibt es Fälle, in denen eine wörtliche Interpretation einer **`if`**-Anweisung zwingend gefordert wird, wenn z.B. der Definitionsbereich einer Funktion verlassen wird. Dies tritt im folgenden Beispiel auf, bei dem das Argument der Funktion `sqrt` (= Berechnung der Wurzel des Arguments) nicht negativ sein darf, da es ansonsten keine reelle Lösung gibt:

```
y = if u >= 0 then sqrt(u) else 0;
```

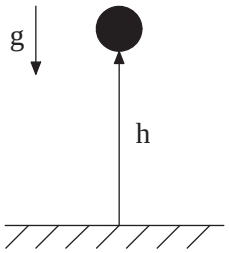
Dieses Modelica-Modell führt während der Simulation zu einem Fehler, da bei der Iteration zur Ermittlung des genauen Umschaltzeitpunktes bei `u = 0` der **`if`**-Zweig erst gewechselt wird, wenn dieser Umschaltzeitpunkt genau genug ermittelt ist. Bei der Ermittlung des Schaltpunktes werden aber auch (kleine) negative Werte von `u` auftreten, da nur dann festgestellt werden kann, daß die Relation `u >= 0` ihren Wert ändern wird. Dies bedeutet, daß die Funktion `sqrt(u)` während der Integation mit einem (kleinen) negativen Argumentwert aufgerufen wird, was zu einem Laufzeitfehler führt. Diese Anweisung muß deswegen folgendermaßen geschrieben werden:

```
y=if noEvent(u >= 0) then sqrt(u) else 0;
```

In diesem Fall wird also bei `u = 0` kein Zustandsereignis generiert und es ist garantiert, daß `sqrt(u)` nur mit nicht-negativen Argumenten aufgerufen wird.

6.3 Neuinitialisierung kontinuierlicher Zustände

Modelica unterstützt eine Reihe von weiteren hybriden Operatoren, wie z.B. **`initial`**() und **`terminal`**() um den ersten und letzten Aufruf der Modellgleichungen zu ermitteln. Insbesondere können an Ereignispunkten kontinuierliche Zustände `x` (= **`der`**`(x)` tritt ebenfalls im




```
model BouncingBall
  parameter Real e = 0.5;
  parameter Real g = 9.81;
            Real h, v;
equation
  der(h) = v;
  der(v) = -g;
  when h <= 0 then
     reinit(v, -e*pre(v));
  end when;
end BouncingBall;
```

Bild 14: Springender Ball.

Modell auf) mit dem **`reinit`**`(x,expr)` Operator neu initialisiert werden. Hierbei wird eine neue Gleichung der Form `x = expr;` eingeführt, wobei `x` der neue Wert des Zustands (= rechter Grenzwert $x(t^+)$ der Variablen) und `expr` der Ausdruck zur Berechnung dieses Neuwertes ist. Weiterhin wird implizit definiert, daß `x` während der Sortierung als unbekannt anzusehen ist um das Synchronitätsprinzip nicht zu verletzen (ansonsten werden alle Zustände bei der Sortierung als bekannt angesehen, siehe Teil 4). Als Beispiel ist in Bild 14 das Modelica Modell eines springenden Balls angegeben, bei dem der Aufprall auf den Boden als Impuls modelliert wird.

Literatur

[53] *Cellier, F.E.:* Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools. Diss. ETH No 6483, ETH Zürich, Schweiz, 1979.

[54] *Eich-Soellner, E.* und *Führer, C.:* Numerical Methods in Multibody Dynamics. Teubner, 1998.

[55] *Elmqvist, H.:* An Object and Data-Flow based Visual Language for Process Control. ISA/92-Canada Conference & Exhibit, Instrument Society of America, Toronto, April 1992.

[56] *Elmqvist, H., Cellier, F.E.* und *Otter, M.:* Object–Oriented Modeling of Hybrid Systems. Proceedings ESS'93, European Simulation Symposium, S. xxxi-xli, Delft, Niederlande, Okt. 1993.

[57] *Gautier, T., Le Guernic, P.* und *Maffeis, O.:* For a New Real-Time Methodology. Publication Interne No. 870, Institut de Recherche en Informatique et Systemes Aleatoires, Campus de Beaulieu, 35042 Rennes Cedex, France, 1994.

[58] *Halbwachs, N., Caspi, P., Raymond, P.* und *Pilaud, D.*. The synchronous data flow programming language LUSTRE. Proc. of the IEEE, 79(9), S. 1305–1321, Sept. 1991.

[59] *Halbwachs, N.:* Synchronous Programming of Reactive Systems. Kluwer, 1993.

[60] *Elmqvist, H., Bachmann, B., Boudaud, F., Broenink, J., Brück, D., Ernst, T., Franke, R., Fritzson, P., Jeandel, A., Grozman, P., Juslin, L., Kagedahl, D., Klose, M., Loubere, N., Mattsson, S.E., Mosterman, P., Nilsson, H., Otter, M., Sahlin, P., Schneider, A., Tummescheit, H.* und *Vangheluwe, H.*: Modelica^{TM} – A Unified Object-Oriented Language for Physical Systems Modeling, Version 1.2, 1999. Modelica homepage: http://www.modelica.org/.

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

[62] *Mosterman, P. J., Otter, M.* und *Elmqvist, H.:* Modeling Petri Nets as Local Constraint Equations for Hybrid Systems using Modelica. SCSC'98, Reno, Nevada, 1998.

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 8

Martin Otter, Oberpfaffenhofen, Hilding Elmqvist, Lund und Sven Erik Mattsson, Lund

(Fortsetzung von Kapitel 6 aus Teil 7)

6.4 Strukturvariable Systeme

In diesem Abschnitt werden ideale Schaltelemente untersucht, die auf strukturvariable Systeme führen.

Ideale elektrische Schaltelemente

Wenn genügend genau modelliert wird, treten in der Regel bei physikalischen Systemen keine Unstetigkeiten auf. Diese ergeben sich auf Grund von *vereinfachten Annahmen*, um einerseits die zur Simulation des Modells benötigte *Rechenzeit* und andererseits den *Identifikationsaufwand* für die Modellparameter zu reduzieren. Als typisches Beispiel ist in Bild 15 eine Diode zu sehen, wobei i den Strom durch die Diode und u den Spannungsabfall zwischen den Klemmen charakterisiert. Die reale Kennlinie ist im linken Teil von Bild 15 dargestellt. Diese kann im Arbeitsbereich der Diode durch die ideale Kennlinie im rechten Teil von Bild 15 approximiert werden, wenn das genaue Schaltverhalten gegenüber dem Effekt des Durchschaltens vernachlässigt werden kann. Hierbei wird die Simulation typischerweise um ein bis zwei Größenordnungen schneller.

Die reale Dioden-Kennlinie im linken Teil von Bild 15 kann problemlos modelliert werden, da nur der Strom i als Funktion des Spannungsabfalls u in analytischer Form, oder mittels einer tabellierten Kennlinie, angegeben werden muß. Im Gegensatz dazu ist auf den ersten Blick nicht klar, wie die ideale Dioden-Kennlinie im rechten Teil von Bild 15 beschrieben werden kann, da der Strom bei $u = 0$ nicht mehr als Funktion des Spannungsabfalls u angegeben werden kann, d.h. eine Funktionsdarstellung der Form $i = f(u)$ ist nicht länger möglich. Eine Lösung dieses Problems ist elegant durch Verwendung einer *Parameterdarstellung* der Kennlinie in der Form $i = i(s)$, $u = u(s)$ mit einem geeigneten Kurvenparameter s erreichbar. Diese Beschreibungsart ist allgemeiner und erlaubt es eine ideale Diode *eindeutig* auf *deklarative* Weise mathematisch zu beschreiben, siehe erste Zeile in Tabelle 10.

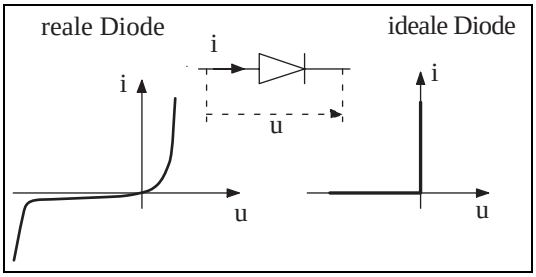


Bild 15: Reale und ideale Dioden-Kennlinie.

Der Bahnparameter s in Tabelle 10 wird so gewählt, daß der Ursprung bei $s = 0$ zu liegen kommt und s bei offener Schalterstellung dem Spannungsabfall u und bei geschlossener dem Strom i_1 entspricht. Die Bool'sche Variable *off* wird benutzt, um die beiden Stellungen des Dioden-Schalters zu kennzeichnen und wird aus dem Bahnparameter berechnet (`off = s < 0`).

Um die Konsequenzen von Kennlinien in Parameterdarstellung zu verstehen, wird das erläuterte ideale Diodenmodell in dem einfachen Gleichrichter-Schaltkreis von Bild 16 eingesetzt. Werden die Gleichungen dieses Schaltkreises aufgestellt, vereinfacht und sortiert, so ergibt sich unter Verwendung des Diodenmodells von Tabelle 10 und mit den Variablen-Bezeichnungen von Bild 16:

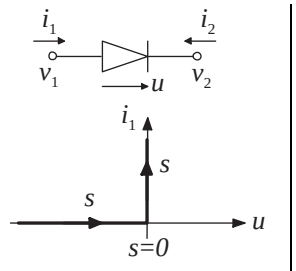
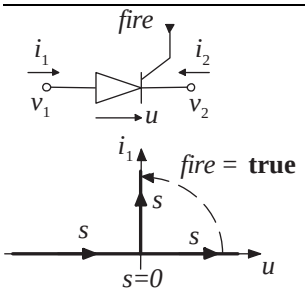
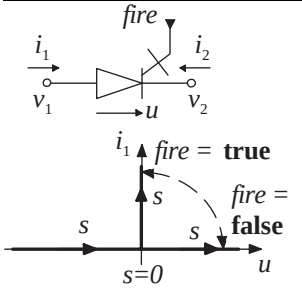
	*ideale Diode* $0 = i_1 + i_2$ $u = v_1 - v_2$ $off = s < 0$ $u =$ **if** *off* **then** s **else** 0 $i_1 =$ **if** *off* **then** 0 **else** s
	*idealer Thyristor* $0 = i_1 + i_2$ $u = v_1 - v_2$ $off = s < 0$ **or** **pre** (*off*) **and not** *fire* $u =$ **if** *off* **then** s **else** 0 $i_1 =$ **if** *off* **then** 0 **else** s
	*idealer GTO-Thyristor* $0 = i_1 + i_2$ $u = v_1 - v_2$ $off = s < 0$ **or not** *fire* $u =$ **if** *off* **then** s **else** 0 $i_1 =$ **if** *off* **then** 0 **else** s

Tabelle 10: Ideale elektrische Schalter

$$
\begin{aligned}
\text{off} &= s < 0 \\
u &= v_1 - v_2 \\
u &= \textbf{if } \text{off } \textbf{then } s \textbf{ else } 0 \\
i_0 &= \textbf{if } \text{off } \textbf{then } 0 \textbf{ else } s \\
R_1 \cdot i_0 &= v_0(t) - v_1 \\
i_2 &:= v_2/R_2 \\
i_1 &:= i_0 - i_2 \\
\frac{dv_2}{dt} &:= i_1/C
\end{aligned}
\tag{6.3}
$$

Man beachte, daß für die Sortierung die Eingangsfunktion $v_0(t)$ und der Zustand der Kapazität v_2 als bekannte Größen angesehen werden. Die ersten 5 Gleichungen sind gekoppelt und bilden ein Gleichungssystem mit den 5 Unbekannten off, s, u, v_1, i_0. Die restlichen Zuweisungen können rekursiv ausgewertet werden um letztendlich die Ableitung der Zustandsgröße, $\dot{v}_2$, zu bestimmen.

Wie im vorigen Abschnitt 6.2 erläutert, werden während der kontinuierlichen Integration die Relationen (hier: $s < 0$) nicht geändert. Damit ändert sich auch die Bool'sche Variable *off* nicht. In dieser Situation muß die erste Gleichung deswegen nicht ausgewertet werden und die nächsten 4 Gleichungen bilden ein lineares Gleichungssystem in den 4 reellen, unbekannten Variablen s, u, v_1, i_0, welches problemlos, z.B. mit dem Gauß'schen Algorithmus, gelöst werden kann.

Wenn eine der Relationen (hier: $s < 0$) ihren Wert ändert, tritt ein Ereignis ein. Auf Grund des in Abschnitt 6.1 erläuterten *Synchronitätsprinzips* müssen alle Gleichungen am *Ereignispunkt* gleichzeitig erfüllt werden. Jetzt bilden die ersten 5 Gleichungen ein Gleichungssystem in den 4 unbekannten, *reellen* Variablen s, u, v_1, i_0 und der unbekannten *Bool'schen* Variablen *off*.

Solche gemischt kontinuierlich/diskreten Gleichungssysteme können auf die folgende Weise gelöst werden: (1) Es wird eine *Annahme* über die Relationen getroffen. (2) Die diskreten Variablen werden berechnet (z.B. *off* im obigen Beispiel). (3) Die reellen Variablen werden durch Lösen eines rein reellen Gleichungssystems bestimmt. (4) Basierend auf (2) und (3) werden die Relationen berechnet. Wenn die Relationen mit der Annahme aus (1) übereinstimmen, wird die Iteration abgebrochen und das gemischte Gleichungssystem ist gelöst. Anderenfalls wird die Annahme korrigiert und die Iteration wird fortgesetzt. Sinnvolle Annahmen sind z.B.: (a) Die Werte der Relationen die im letzten Schritt berechnet wurden, wobei als Startannahme die Werte der Relationen vom letzten Ereignis verwendet werden. Durch verschiedene zusätzliche Maßnahmen kann das sichere Auffinden von konsistenten Lösungen verbessert werden. (b) Es werden systematisch alle sinnvollen Werte der Relationen durchprobiert. Im obigen Beispiel können beide Vorgehensweisen einfach angewandt werden, da nur zwei mögliche Werte der Relation auftreten können ($s < 0$ ist **true** oder **false**). Wenn jedoch n Schalter gekoppelt sind, gibt es n verschiedene Relationen und 2^n verschiedene Werte die im ungünstigsten Falle alle überprüft werden müssen.

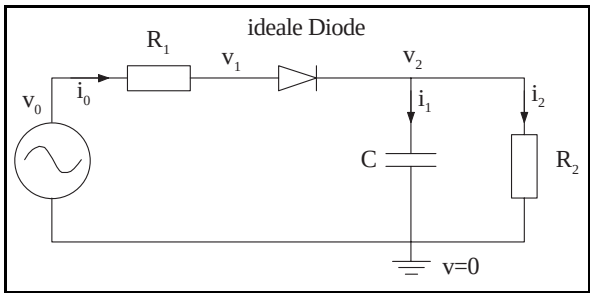


Bild 16: Einfacher Gleichrichter-Schaltkreis.

Zur weiteren Demonstration sind in Tabelle 10 noch andere elektrische Schalter, der ideale Thyristor und der ideale GTO-Thyristor, aufgeführt. Hier tritt eine weitere Schwierigkeit auf, da es drei Kennlinienäste gibt, die ins Unendliche weisen. Unter der Annahme, daß benachbarte Kennlinienpunkte auch benachbarte Werte des Bahnparameters s besitzen, ist dann eine *eindeutige* Beschreibung mit *einem* Bahnparameter nicht möglich. Das Problem wird gelöst, indem zwei der Äste dieselbe s-Parameterisierung erhalten ($s \geq 0$) und die Eindeutigkeit durch den Wert von *off* erreicht wird.

Zusammengefaßt kann festgehalten werden, daß strukturvariable Komponenten, wie ideale Dioden, durch eine Parameterdarstellung der entsprechenden Kennlinie modelliert werden können. Aus Benutzersicht ist dieses Vorgehen sehr eingängig. Ein solches Modell führt an *Ereignispunkten* auf *gemischt kontinuierlich/diskrete* Gleichungssysteme, die mit entsprechenden Algorithmen gelöst werden müssen.

Die Methode, Kennlinien durch eine Parameterdarstellung zur beschreiben, wurde in [64] vorgeschlagen. In [68] wird gezeigt, daß dieses Verfahren auf gemischt kontinuierlich/diskrete Gleichungssysteme führt und es werden Algorithmen zu deren Lösung skizziert. Dieses Vorgehen bietet Vorteile gegenüber bisher bekannten Verfahren, wie (a) die Verwendung von endlichen Automaten zur Beschreibung der Schaltstruktur, siehe z.B. [63, 65, 67], oder (b) einer Transformation in ein Komplementaritätsproblem, siehe z.B. [66, 69], da bei der Parameterdarstellung effizientere und bessere Algorithmen zur Ermittlung einer konsistenten Lösung eingesetzt werden können als bei (a), und (b) nur für sehr spezielle Problemklassen anwendbar ist (es können z.B. keine idealen Thyristoren beschrieben werden).

Reibung

Die Simulation von idealen Schaltelementen wird schwierig, wenn sich durch das Schalten die Zahl der Zustände ändert. Ein typisches Beispiel ist Coulomb-Reibung, bei der diese Situation selbst im einfachsten Fall vorliegt. Um sich auf die wesentlichen Eigenschaften zu konzentrieren, wird zuerst das vereinfachte Reibmodell in Bild 17 untersucht. Die Reibkraft f wirkt zwischen zwei Oberflächen, siehe rechter Teil von Bild 17, und ist eine lineare Funktion der relativen Geschwindigkeit v, wenn die beiden Reibflächen aufeinander gleiten. Wenn die Relativgeschwindigkeit verschwindet, haften die beiden Oberflächen aufeinander und die Reibkraft ist nicht länger eine Funktion von v. Wenn der Betrag der Reibkraft größer wird als die maximale statische Reib-

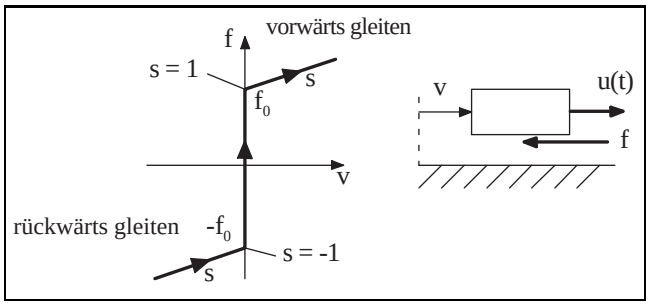


Bild 17: Vereinfachtes Coulomb Reibelement.

kraft f_0, beginnen die beiden Oberflächen wieder aufeinander zu gleiten. Die Kennlinie dieses Elementes kann ebenfalls in einer Parameterdarstellung angegeben werden und führt auf die folgenden Gleichungen:

```
forward  = s >  1;
backward = s < -1;
v=if forward  then  s - 1       else
  if backward then  s + 1       else 0;
f=if forward  then  f0+f1*(s-1) else
  if backward then -f0+f1*(s+1) else f0*s;
```

Hiermit wird das vereinfachte Reibmodell vollständig und *deklarativ* beschrieben. Leider ist zur Zeit nicht bekannt, wie eine solche Elementbeschreibung *automatisch* in eine Form überführt werden kann, die numerisch zuverlässig simulierbar ist. Die auftretenden Schwierigkeiten werden an dem reibungsbehafteten Block im rechten Teil von Bild 17 diskutiert. Dieser wird durch die folgende Gleichung beschrieben:

$$m \cdot \dot{v} = u - f \tag{6.4}$$

Hierbei ist m die Masse des Blocks und $u(t)$ ist eine gegebene äußere Kraft die an dem Block angreift. Wenn sich das Reibelement im *Vorwärts-Gleitzustand* befindet, d.h. $s > 1$, wird das Gesamtsystem durch

$$\begin{aligned} m \cdot \dot{v} &= u - f \\ v &= s - 1 \\ f &= f_0 + f_1 \cdot (s-1) \end{aligned}$$

beschrieben, und kann einfach in eine Zustandsform mit v als Zustandsgröße transformiert werden. Wenn der Block haftet, d.h. $-1 \leq s \leq 1$, wird die Gleichung $v = 0$ aktiv und v kann nicht länger ein Zustand sein. Es liegt ein singuläres System vor, das in diesem Schaltzustand mit den Methoden von Kapitel 5 behandelt werden muß. Es gibt jedoch noch ein größeres Problem: Wenn der Block haftet und s größer als eins wird, dann ist $s \leq 1$ and $v = 0$ kurz bevor das Ereignis eintritt. Am Ereignispunkt ist $s > 1$, da die Änderung dieser Relation das Zustandsereignis auslöst. Das Reibelement schaltet in den Vorwärtsgleitzustand um, wobei v ein Zustand ist, der mit dem letzten Wert $v = 0$ initialisiert wird. Da v Zustand ist, wird s über $s := v + 1$ berechnet und ergibt $s = 1$, d.h. die Relation $s > 1$ wird wieder **false** und das Reibelement schaltet zurück in den Haftzustand. Mit anderen Worten, es ist unmöglich vom Haft- in den Gleitzustand zu schalten. Diese Analyse ändert sich nicht, wenn auch numerische Fehler berücksichtigt werden.

Wie schon erwähnt, ist zur Zeit nicht bekannt wie diese Probleme *automatisch* gelöst werden können. Wir gehen deswegen pragmatisch vor und transformieren die Gleichungen des Reibmodells *manuell* in eine leichter verarbeitbare Form:

Wenn die Relativgeschwindigkeit verschwindet, kann sich das Reibelement sowohl im Haft- als auch im Gleitzustand befinden (v gibt darüber keine Aussage). Die aktuelle Konfiguration ergibt sich aus der Reibkraft und der Beschleunigung $a = \dot{v}$, siehe Bild 18, da $a = 0$ Haften und z.B. $a > 0$ Vorwärtsgleiten charakterisiert. Damit kann das Reibelement bei $v = 0$ durch die folgenden Gleichungen, einer Parameterdarstellung der Kennlinie von Bild 18, beschrieben werden.

```
startFor  = sa >  1;
startBack = sa < -1;
a = der(v);
a = if startFor  then  sa-1 else
    if startBack then  sa+1 else 0;
f = if startFor  then  f0   else
    if startBack then -f0   else f0*sa;
```

Diese Gleichungen, zusammen mit der Gleichung des Blocks (6.4), bilden wieder ein gemischt kontinuierlich/diskretes Gleichungssystem das an Ereignispunkten gelöst werden muß. Wenn vom Gleiten in den Haftzustand geschaltet wird, ist die Geschwindigkeit klein oder verschwindet beim Umschaltpunkt. Da die Ableitung der Zwangsgleichung, d.h. $\dot{v} = 0$, im Haftzustand erfüllt ist, bleibt die Relativgeschwindigkeit klein, auch wenn die Gleichung $v = 0$ nicht explizit berücksichtigt wird. Mit dieser bekannten Vorgehensweise bleibt v in jeder Konfiguration ein Zustand und die Probleme mit singulären Systemen werden vermieden.

Damit folgt, daß v klein ist, aber jedes Vorzeichen besitzen kann, wenn vom Haft- in den Gleitzustand geschaltet wird. Deswegen muß solange "gewartet" werden, bis die Geschwindigkeit z.B. wirklich positiv geworden ist, wenn in den Vorwärtsgleitzustand geschaltet werden soll (man beachte, daß auch bei exakter Rechnung eine "Wartephase" notwendig ist, da beim Beginn des Gleitens $v = 0$ ist). Da $\dot{v} > 0$ ist, wird dieser Fall nach einer kurzen Zeitspanne auftreten. Diese "Warteprozedur" kann einfach mit dem endlichen Automaten von Bild 19 beschrieben werden. Die beiden diskutierten Darstellungsformen müssen jetzt nur noch kombiniert werden. Hierzu wird definiert, daß *startFor* und *startBack* nur **true** sein können, wenn sich das Reibelement im *Stuck*-Modus $v = 0$ befindet. Zusammengefaßt ergibt sich schließlich das folgende Modell des vereinfachten

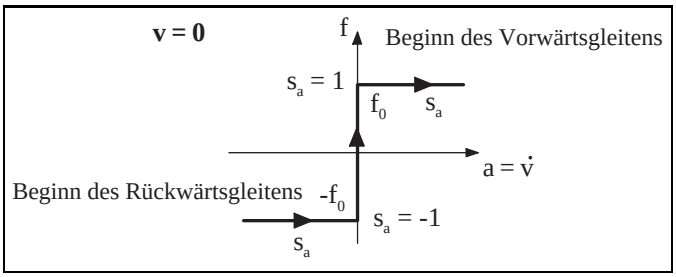


Bild 18: Reibkennlinie bei $v = 0$.

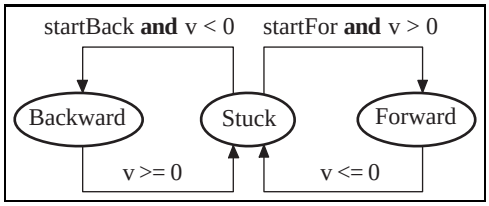


Bild 19: "Wartephase" beim Umschalten vom Haften ins Gleiten.

Coulomb Reibmodells von Bild 17:

```
// Teil vom gemischten Gleichungssystem
startFor  = pre(mode)==Stuck and sa >  1;
startBack = pre(mode)==Stuck and sa < -1;
a=der(v);
a=if pre(mode)==Forward or startFor then
      sa - 1 else
  if pre(mode)==Backward or startBack then
      sa + 1 else 0;
f=if pre(mode)==Forward or startFor then
      f0 + f1*v else
  if pre(mode)==Backward or startBack then
     -f0 + f1*v else f0*sa;

// endlicher Automat zur Konfigurationsbest.
mode=if (pre(mode)==Forward or startFor)
         and v>0 then Forward else
     if (pre(mode)==Backward or startBack)
         and v<0 then Backward else Stuck;
```

Das gemischt kontinuierlich/diskrete Gleichungssystem wird mit demjenigen Wert von `mode` gelöst, der beim Eintreten des Ereignisses vorliegt, also von `pre(mode)`. Danach berechnet sich der neue Wert von `mode` mit der letzten Gleichung, die eine direkte Übertragung des endlichen Automaten von Bild 19 darstellt.

Die erläuterte Vorgehensweise kann auch auf das allgemeinere Coulomb Reibelement von Bild 20 angewendet werden, bei der die Gleitreibkraft eine nichtlineare Funktion ist und sich die Reibkraft unstetig von f_{max} nach f_0 ändert, wenn vom Haften in den Beginn des Vorwärtsgleitens geschaltet wird. Hierzu müssen (1) die lineare Gleichung der Gleitreibkraft durch eine nichtlineare Beziehung und (2) die Gleitbedingungen durch

```
startFor  = pre(mode)==Stuck and (sa > peak
            or pre(startFor) and sa > 1);
startBack = pre(mode)==Stuck and (sa< -peak
            or pre(startBack) and sa < -1);
```

ersetzt werden. Hierbei ist `peak` = $f_{max}/f_0 \geq 1$. Alle anderen Gleichungen können unverändert vom einfachen Reibelement übernommen werden.

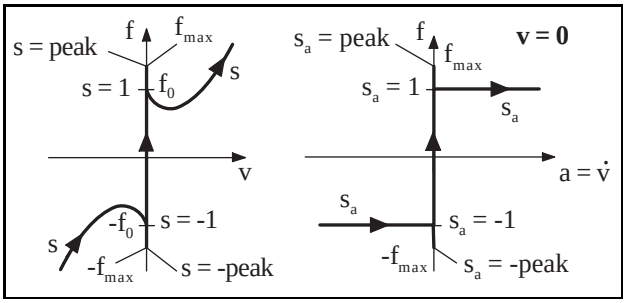


Bild 20: Coulomb Reibelement.

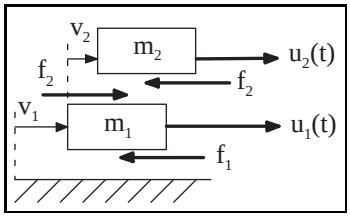


Bild 21: Zwei aufeinander gleitende Blöcke mit Reibung.

Als einfaches Beispiel zur dynamischen Kopplung von Reibelementen sind in Bild 21 zwei aufeinander gleitende Blöcke zu sehen, bei denen die Reibkräfte f_1, f_2 wirken. An Ereignispunkten muß ein gemischt kontinuierlich/diskretes Gleichungssystem der folgenden Struktur gelöst werden:

$$\begin{aligned} m_1\dot{v}_1 &= u_1 + f_2 - f_1; & \dot{v}_1 &= g_1(s_1 > 1,\ s_1 < -1,\ s_1) \\ m_2(\dot{v}_1 + \dot{v}_2) &= u_2 - f_2; & f_1 &= g_2(s_1 > 1,\ s_1 < -1,\ s_1, v_1) \\ & & \dot{v}_2 &= g_3(s_2 > 1,\ s_2 < -1,\ s_2) \\ & & f_2 &= g_4(s_2 > 1,\ s_2 < -1,\ s_2, v_2) \end{aligned}$$

Da 4 Relationen vorliegen, müssen im ungünstigsten Fall $2^4 = 16$ Annahmen überprüft werden. Von diesen sind nur $3^2 = 9$ relevant, da z.B. $s_1 > 1 =$ **true**, $s_1 < -1 =$ **true** nicht gleichzeitig auftreten können.

6.5 Ausblick

In Teil 1–8 dieser Serie wurde eine Einführung in die objektorientierte Modellierung physikalischer Systeme gegeben und an einfachen Modellen demonstriert. Im weiteren Verlauf der Serie wird die erläuterte Technik bei komplexeren Anwendungen aus unterschiedlichen Fachgebieten eingesetzt, und auf Eigenheiten bei der Modellierung in der jeweiligen Disziplin eingegangen.

Literatur

[63] *Barton, P.I.:* The Modelling and Simulation of Combined Discrete/Continuous Processes. Ph.D. Thesis, University of London, Imperial College, 1992.

[64] *Clauß, C., Haase, J., Kurth, G.* und *Schwarz, P.:* Extended Amittance Description of Nonlinear n-Poles. Archiv für Elektronik und Übertragungstechnik / International Journal of Electronics and Communications, 40, S. 91-97, 1995.

[65] *Elmqvist, H., Cellier, F.E.* und *Otter, M.:* Object–Oriented Modeling of Hybrid Systems. Proceedings ESS'93, European Simulation Symposium, S. xxxi-xli, Delft, Niederlande, Okt. 1993.

[66] *Lötstedt, P.:* Mechanical systems of rigid bodies subject to unilateral constraints. SIAM J. Appl. Math., Vol. 42, No. 2, S. 281-296, 1982.

[67] *Mosterman, P. J.* und *Biswas, G.:* A Formal Hybrid Modeling Scheme for Handling Discontinuities in Physical System Models. Proceedings of AAAI-96, S. 905-990, 2.-4. Aug., Portland, OR, 1996.

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

[69] *Pfeiffer, F.* und *Glocker, C.*: Multibody Dynamics with Unilateral Contacts. John Wiley, 1996.

Manuskripteingang: ???

Objektorientierte Modellierung Physikalischer Systeme, Teil 9

Martin Otter und Clemens Schlegel, Oberpfaffenhofen

Dr.-Ing. Martin Otter ist wissenschaftlicher Mitarbeiter beim DLR und Lehrbeauftragter an der TU München. Hauptarbeitsgebiete: Steuerung und Regelung von Industrierobotern, Antriebsstrangmodellierung für Echtzeitanwendungen, objektorientierte Modellierung, Modelica Sprachentwurf.

Adresse: Deutsches Zentrum für Luft- und Raumfahrt, Institut für Robotik und Systemdynamik, Postfach 1116, D-82230 Weßling, Tel.: 08153/28-2473, Email: Martin.Otter@DLR.de

2

Dipl.-Ing. Clemens Schlegel ist wissenschaftlicher Mitarbeiter beim DLR. Hauptarbeitsgebiete: Simulation, insbesondere Echtzeit- und Hardware-in-the-loop Simulation, sowie digitale Regelung und Signalverarbeitung

Adresse: Deutsches Zentrum für Luft- und Raumfahrt, Hauptabteilung Raumflugbetrieb und Astronautentraining, Postfach 1116, D-82230 Weßling, Tel.: 08153/28-1521, Email: Clemens.Schlegel@DLR.de

7 Modellierung von Antriebssträngen

In Teil 1–8 dieser Serie wurde eine Einführung in die objektorientierte Modellierung physikalischer Systeme gegeben und an einfachen Modellen demonstriert. Beginnend mit dem vorliegenden Teil 9, wird diese Technik bei komplexeren Anwendungen aus unterschiedlichen Fachgebieten eingesetzt, und auf Eigenheiten der jeweiligen Disziplin eingegangen. Viele der diskutierten Komponenten sind in der frei verfügbaren Basisbibliothek von Modelica [76] enthalten und sofort einsetzbar.

Antriebsstränge sind 1-dimensionale, rotatorische mechanische Systeme die im wesentlichen die mechanische Energie zwischen einem Motor und einer Arbeitsmaschine übertragen und dabei eine gewünschte Übersetzung von Drehzahl und Moment realisieren. Traditionelle Modellierungsverfahren von Antriebssträngen sind z.B. in [75, 73] beschrieben. Die objektorientierte Modellierung von Antriebssträngen [77, 74] erlaubt eine flexiblere und mächtigere Vorgehensweise und stellt gegenüber den traditionellen Verfahren einen wesentlichen Fortschritt dar, wie im folgenden beim Aufbau einer geeigneten Bibliothek für Antriebsstränge gezeigt wird.

7.1 Schnittstellen

Der erste Schritt beim Aufbau einer Komponenten-Bibliothek besteht in der Festlegung der Komponenten-Schnittstellen. Die Schnittstellen einer Antriebsstrangkomponente sind *mechanische Flansche*, siehe Zeile 1 von Tabelle 10. Als Schnittstellengrößen werden der *absolute Drehwinkel* φ und das *Schnittmoment* τ in einem Flansch verwendet. Hierbei ist φ eine Potential- und τ eine Flußvariable. In jedem Flansch wird ein fest mit dem Flansch verbundenes *Flansch-Koordinatensystem* definiert, wobei die z-Achse in Richtung der positiven Drehrichtung zeigt. Alle Vektoren von Schnittstellengrößen werden im jeweiligen Flansch-Koordinatensystem angegeben. Wenn nur ein Koordinatensystem in einer Komponente aufgeführt ist (siehe Zeile 1 von Tabelle 10), sind alle Flansch-Koordinatensysteme dieser Komponente hierzu parallel ausgerichtet. Aus diesem Grunde zeigt z.B. das Schnittmoment τ_1 vom linken Flansch in Zeile 1 von Tabelle 10 *in* den Flansch und das Schnittmoment τ_2 vom rechten Flansch *aus* dem Flansch. Eine Verbindungslinie zwischen zwei Flanschen bedeutet, daß die beiden Flansche mechanisch starr verbunden sind, und daß insbesondere die beiden *Flansch-Koordinatensysteme deckungsgleich* sind. Diese Definition und die sich hieraus ergebenden Konsequenzen wurden schon in Teil 3 erläutert.

7.2 Grundelemente

In Tabelle 10 sind die wichtigsten Grundelemente von Antriebssträngen aufgeführt, wobei die Komponenten-Gleichungen in der rechten Spalte angegeben sind:

Trägheit: Trägheitsbehaftete Welle mit zwei Flanschen. Die Gleichungen ergeben sich aus dem Drallsatz.

Feder: Elastizität einer Welle oder eines Getriebes.

Dämpfer: Lineare Dämpfung zwischen zwei Wellen oder zwischen einer Welle und dem Lagergehäuse.

Lose: Lose und Elastizität bei einem Getriebe. Die Lose wird nicht als eigenständiges Element definiert, da ansonsten durch die mögliche direkte Verschaltung mit zwei trägheitsbehafteten Wellen *Stöße* auftreten können, die mit der vorgestellten Methodik (noch) nicht *automatisch* abgehandelt werden können.

Lagergehäuse: Wird verwendet um eine Welle auf einen festen Drehwinkel zu fixieren oder um ein Kraftelement (Feder, Dämpfer, etc.) zwischen einer Welle und dem Gehäuse anzubringen.

externes Moment: Ein Eingangssignal $u(t)$ (z.B. Stellgröße eines Reglers) wird als externes Moment auf einen Flansch geschaltet.

Drehzahlsensor: Die Drehzahl eines Flansches wird in ein Ausgangssignal $y(t)$ umgewandelt. Entsprechend können Drehwinkel, Winkelbeschleunigung und Schnittmoment im Flansch in ein Ausgangssignal umgewandelt werden.

7.3 Einfache, ideale Getriebe

Im unteren Teil von Tabelle 10 sind einfache, ideale Getriebe aufgeführt. Durch entsprechende Verschaltung mit den Grundelementen (z.B. mit "Trägheit", "Elastizität" und "Lose") können realistische Getriebemodelle aufgebaut werden. Ideale Getriebe sind Übertragungselemente, bei denen die Energie verlustlos übertragen wird und die Drehzahl in einem bestimmten Verhältnis geändert wird. Zur Ableitung der entsprechenden Gleichungen müssen zuerst die Beziehungen zwischen den Flanschwinkeln auf Grund kinematischer Überlegungen ermittelt werden (siehe nächster Abschnitt). Die Beziehungen zwischen den Schnittmomenten in den Flanschen ergeben sich dann aus dem Energiesatz. Dies wird exemplarisch am Standard-Planetengetriebe (siehe Tabelle 10) gezeigt:

Da im idealen Planetengetriebe keine Energie gespeichert oder in Form von Wärme in die Umgebung abgeführt wird, muß die Summe der in die Komponente fließenden Energien verschwinden, d.h.

$$\begin{aligned} 0 &= P \ (= dE/dt) \\ &= \dot{\varphi}_s \cdot \tau_s + \dot{\varphi}_t \cdot \tau_t + \dot{\varphi}_h \cdot \tau_h \\ &= \left((1+i)\dot{\varphi}_t - i\dot{\varphi}_h\right) \cdot \tau_s + \dot{\varphi}_t \cdot \tau_t + \dot{\varphi}_h \cdot \tau_h \\ &= \left(\tau_t + (1+i)\tau_s\right) \cdot \dot{\varphi}_t + \left(\tau_h - i\tau_s\right) \cdot \dot{\varphi}_h \end{aligned} \qquad (6.1)$$

Die beiden Drehzahlen $\dot{\varphi}_t, \dot{\varphi}_h$ können unabhängig und beliebig voneinander eingestellt werden. Damit die Gesamtleistung $P(t)$ immer verschwindet, müssen deswegen die Vorfaktoren in (6.1) verschwinden. Dies führt auf die Gleichungen von Tabelle 10:

$$0 \ = \ \tau_t + (1+i) \cdot \tau_s, \qquad 0 \ = \ \tau_h - i \cdot \tau_s \qquad (6.2)$$

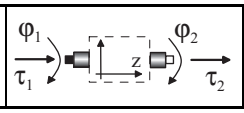
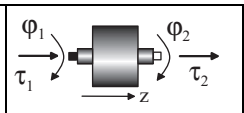
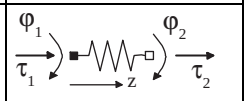
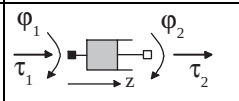
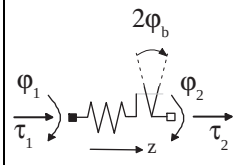
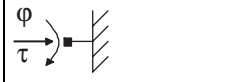
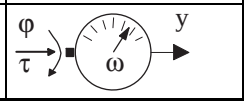
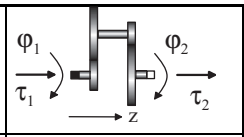
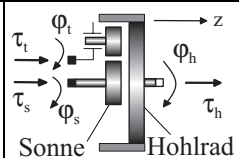
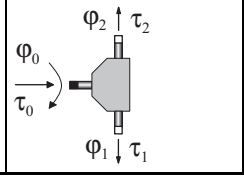
	Schnittstelle	
Schnittstelle		$\tau_1 = \tau_1 \cdot \mathbf{e}_z$ $\tau_2 = \tau_2 \cdot \mathbf{e}_z$ $\omega_1 = \dot{\varphi}_1 \cdot \mathbf{e}_z$ $\omega_2 = \dot{\varphi}_2 \cdot \mathbf{e}_z$
	Grundelemente	
Trägheit		J: Trägheitsmoment $\varphi_1 = \varphi_2$ $\dot{\varphi}_1 = \omega_1$ $J \cdot \dot{\omega}_1 = \tau_1 + \tau_2$
Elastizität		c: Federsteifigkeit $0 = \tau_1 + \tau_2$ $\tau_2 = c \cdot (\varphi_2 - \varphi_1)$
Dämpfung		$0 = \tau_1 + \tau_2$ $\Delta\varphi = \varphi_2 - \varphi_1$ $\tau_2 = d \cdot \Delta\dot{\varphi}$
Lose		$0 = \tau_1 + \tau_2$ $\Delta\varphi = \varphi_2 - \varphi_1$ τ_2 = **if** $\Delta\varphi > \varphi_b$ **then** $c \cdot (\Delta\varphi - \varphi_b)$ **else if** $\Delta\varphi < -\varphi_b$ **then** $c \cdot (\Delta\varphi + \varphi_b)$ **else** 0
Lager-gehäuse		φ_0: Gehäuse-Lagewinkel $\varphi = \varphi_0$
externes Moment		u: Eingangs*signal* $\tau = -u$
Drehzahl-Sensor		y: Ausgangs*signal* $y = \dot{\varphi}$ $\tau = 0$
	ideale Getriebe	
Stirnrad-getriebe		i: Getriebeübersetzung $\varphi_1 = i \cdot \varphi_2$ $0 = i \cdot \tau_1 + \tau_2$
Standard-Planeten-getriebe		z_s: Zähnezahl Sonnenrad z_h: Zähnezahl Hohlrad $i = z_h / z_s$ $(1+i) \cdot \varphi_t = \varphi_s + i \cdot \varphi_h$ $\tau_h = i \cdot \tau_s$ $0 = \tau_t + (1+i) \cdot \tau_s$
Differential-getriebe		$\varphi_0 = (\varphi_1 + \varphi_2)/2$ $0 = \tau_1 + \tau_0/2$ $0 = \tau_2 + \tau_0/2$

Tabelle 10: Bibliothek von Antriebsstrang-Komponenten.

7.4 Ideale Planetengetriebe

Es gibt eine Vielzahl unterschiedlicher und komplexer Planetengetriebe, so daß es unkomfortabel wäre, für jeden denkbaren Typ die Gleichungen manuell abzuleiten. Es ist jedoch möglich praktisch jedes denkbare

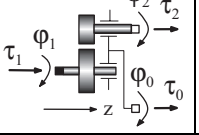
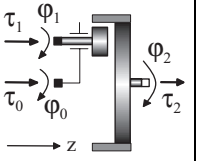
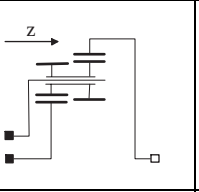
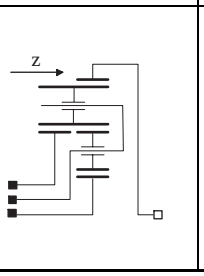
Außen-Außen Verzahnung		z_j: Zähnezahl Rad j $i_1 = z_1/(z_1+z_2)$ $i_2 = z_2/(z_1+z_2)$ $\varphi_0 = i_1 \cdot \varphi_1 + i_2 \cdot \varphi_2$ $0 = \tau_1 + i_1 \cdot \tau_0$ $0 = \tau_2 + i_2 \cdot \tau_0$
Außen-Innen Verzahnung		z_j: Zähnezahl Rad j $i_1 = z_1/(z_2-z_1)$ $i_2 = z_2/(z_2-z_1)$ $\varphi_0 = i_2 \cdot \varphi_2 - i_1 \cdot \varphi_1$ $\tau_1 = i_1 \cdot \tau_0$ $0 = i_2 \cdot \tau_0 + \tau_2$
Planeten-radsatz		
Ravigneaux-getriebe		

Tabelle 11: Bibliothek von idealen Planetengetrieben.

Planetengetriebe mit Hilfe von zwei Basiskomponenten aufzubauen (siehe die beiden ersten Zeilen in Tabelle 11): (1) Einer Außen-Außen Verzahnung und (2) einer Außen-Innen Verzahnung. Dies wird in den beiden unteren Zeilen von Tabelle11 exemplarisch demonstriert:

Beim "Planeten-Radsatz" wird das Planetenrad so aufgebaut, daß der Eingriff mit dem Sonnenrad und dem Hohlrad zu unterschiedlichen Wälzkreisen am Planetenrad führen kann, wie bei modernen Lastgetrieben üblich. Dies entspricht zwei fest gekoppelten Planetenrädern mit unterschiedlichen Radien (siehe rechte Spalte von Zeile 3 in Tabelle 11). Das Ravigneauxgetriebe wird häufig als Kern eines 3- oder 4-gängigen Automatikgetriebes in Fahrzeugen verwendet und besteht aus 2 Sonnenrädern, 2 Planetenrädern und einem Hohlrad (siehe z.B. [72]). In der rechten Spalte von Tabelle 11 wird gezeigt, wie dieses Getriebe durch geeignete Verschaltung von 3 Außen-Außen Verzahnungen und einer Außen-Innen Verzahnung modelliert werden kann.

Die Ableitung der Gleichungen der Außen-Außen Verzahnung soll kurz skizziert werden: Die benötigten Hilfsgrößen sind in Bild 12 definiert. Es wird die bekannte Formel benötigt, um die absolute Geschwindigkeit $\mathbf{v}_P$ eines Punktes P aus der absoluten Geschwindigkeit $\mathbf{v}_B$ eines Bezugspunktes B auf *demselben* Körper, der absoluten Winkelgeschwindigkeit $\boldsymbol{\omega}$ dieses Körpers und des Ortsvektors $\mathbf{r}_{BP}$ von B nach Q zu berechnen:

$$\mathbf{v}_P \;=\; \mathbf{v}_B + \boldsymbol{\omega} \times \mathbf{r}_{BP} \tag{6.3}$$

Anwenden von (6.3) auf die Punkte P_1, P_2, P_3 in Bild 12, sowie Berücksichtigung der geschlossenen Vektorkette der Ortsvektoren $\mathbf{R}_1, \mathbf{r}_1, \mathbf{r}_2, \mathbf{R}_2$ führt auf:

$$\mathbf{R}_1 + \mathbf{r}_1 = \mathbf{R}_2 + \mathbf{r}_2 \tag{6.4a}$$
$$\mathbf{v}_1 = \boldsymbol{\omega}_0 \times \mathbf{R}_1 \tag{6.4b}$$
$$\mathbf{v}_2 = \boldsymbol{\omega}_0 \times \mathbf{R}_2 \tag{6.4c}$$
$$\mathbf{v}_3 = \mathbf{v}_1 + \boldsymbol{\omega}_1 \times \mathbf{r}_1 \;=\; \mathbf{v}_2 + \boldsymbol{\omega}_2 \times \mathbf{r}_2 \tag{6.4d}$$

Einsetzen von (6.4a–6.4c) in (6.4d) ergibt:

$$\boldsymbol{\omega}_0 \times (\mathbf{r}_2 - \mathbf{r}_1) + \boldsymbol{\omega}_1 \times \mathbf{r}_1 \;=\; \boldsymbol{\omega}_2 \times \mathbf{r}_2 \tag{6.5}$$

Werden die Vektoren im Koordinatensystem vom Getrieberad 2 (siehe Bild 12) angeschrieben, d.h.

$$\begin{array}{l} \boldsymbol{\omega}_0 = \dot{\varphi}_0 \cdot \mathbf{e}_z, \quad \boldsymbol{\omega}_1 = \dot{\varphi}_1 \cdot \mathbf{e}_z, \quad \boldsymbol{\omega}_2 = \dot{\varphi}_2 \cdot \mathbf{e}_z, \\ \mathbf{r}_1 = r_1 \cdot \mathbf{e}_x, \quad \mathbf{r}_2 = r_2 \cdot (-\mathbf{e}_x) \end{array} \tag{6.6}$$

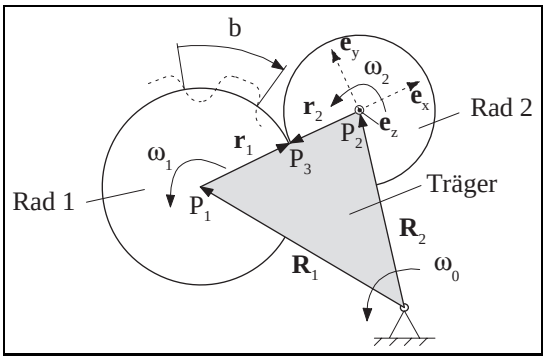


Bild 12: Bezeichnungen der Außen-Außen Verzahnung.

in (6.5) eingesetzt und einmal integriert folgt

$$(r_1 + r_2) \cdot \varphi_0 = r_1 \cdot \varphi_1 + r_2 \cdot \varphi_2 \tag{6.7}$$

Sei z_1 die Zahl der Zähne von Rad 1, z_2 die Zahl der Zähne vom Rad 2 und b die Breite eines Zahnes und einer Zahnlücke am Wälzkreis, dann gilt für den Umfang der beiden Getrieberäder:

$$2\pi r_1 = z_1 \cdot b, \quad 2\pi r_2 = z_2 \cdot b \tag{6.8}$$

Auflösen dieser beiden Gleichungen nach r_1 bzw. r_2 und einsetzen in (6.7) ergibt schließlich die gesuchte kinematische Beziehung zwischen den Drehwinkeln und den Zähnezahlen:

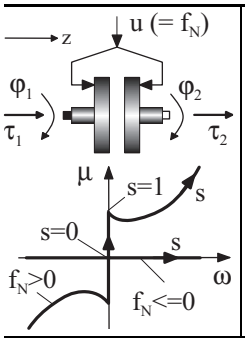
$$(z_1 + z_2) \cdot \varphi_0 = z_1 \cdot \varphi_1 + z_2 \cdot \varphi_2 \tag{6.9}$$

Wie im vorigen Abschnitt gezeigt, können dann aus dem Energiesatz die Beziehungen zwischen den Schnittmomenten τ_i abgeleitet werden. Man beachte, daß (6.9) geeigneter ist als (6.7), da die Zähnezahlen eines gegebenen Getriebes *exakt* durch Abzählen bestimmt werden können, während die Wälzkreisradien r_1, r_2 nur im Rahmen einer gewissen *Toleranz* ermittelbar sind.

7.5 Reibelemente

Bei Antriebssträngen gibt es eine ganze Reihe von Komponenten, die in unterschiedlicher Weise Coulomb-Reibung enthalten, wie z.B. Lagerreibung, Zahnflankenreibung (= Reibung zwischen Getriebezähnen; führt auf lastabhängiges Verlustmoment), Kupplung, Scheiben-, Trommel- und Band-Bremse. Ein Freilauf läßt nur eine Drehrichtung zu und führt zu vergleichbaren Modellierungsproblemen wie bei der Reibung. Eine detailliertere Beschreibung dieser Komponenten ist aus Platzgründen nicht möglich. Alle Elemente basieren jedoch auf dem im Teil 8 ausführlich erläuterten Basis-Reibelement. Exemplarisch wird am Beispiel einer Reibkupplung gezeigt, wie die Gleichungen aus dem Basis-Reibelement abgeleitet werden können:

Die translatorischen Größen (f, v) werden durch die entsprechenden rotatorischen Größen (τ, ω) ersetzt. Über einen zusätzlichen Signaleingang wird die Anpreßkraft f_N der Kupplungsscheiben zugeführt. Für $f_N \leq 0$, wird das Reibelement "abgeschaltet". Dies wird in den Gleichungen durch eine zusätzliche Bedingung **if** *off* **then** ... erreicht. Das Reibmoment τ_2 berechnet sich



$\varphi = \varphi_2 - \varphi_1$
$\omega = \dot{\varphi}$
off = $f_N \leq 0$
$\tau_2 = \mu \cdot f_N \cdot c_{geo}$
$\tau_1 = -\tau_2$
$\dot{\omega}$ = **if** *off* **then** s_a **else if pre**(*m*)
== *Forward* **then** $s_a - 1$ ...
μ = **if** *off* **then** 0 **else if pre**(*m*)
== *Forward* **then** $\mu_0 + \mu_1(\omega)$ ...
m = **if** *off* **then** *Free* **else if** ...
startFor = **pre**(*m*) == *Stuck* **and** (*sa* ...
startBack = **pre**(*m*) == *Stuck* **and** (*sa* ...

Tabelle 12: Reibkupplung ("..." bedeutet: siehe Teil 8 dieser Serie).

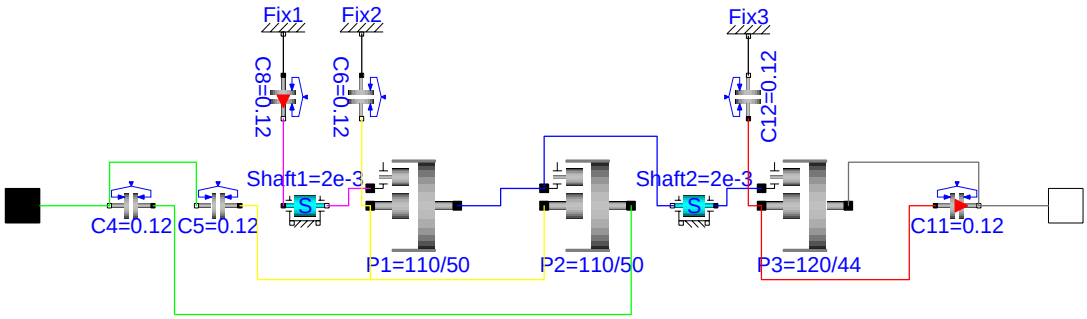


Bild 13: Objektdiagramm des Automatikgetriebes ZF 4HP22.

als Produkt der Anpreßkraft f_N, des Reibkoeffizienten μ und eines Geometriefaktors c_{geo}. Statt der Reibkraft f von Teil 8, wird hier die Kennlinie von μ als Funktion der Relativdrehzahl angegeben. Der Geometriefaktor c_{geo} berücksichtigt den Kupplungsaufbau. Besteht die Kupplung z.B. aus n Reibflächen mit jeweils einem Innenradius r_i und einem Außenradius r_a gilt unter der Annahme gleichmäßigen Tragens der Kupplungsscheiben

$$c_{geo} = n \cdot (r_i + r_a)/2 \tag{6.10}$$

Die Bedingungen für die Bool'schen Variablen *startFor* und *startBack* bleiben gleich, während bei der Mode-Variablen m noch die Abschaltung der Kupplung berücksichtigt werden muß.

7.6 Anwendungsbeispiel: Automatikgetriebe

Mit den beschriebenen Bibliothekselementen kann beispielsweise ein Simulationsmodell für die Dynamik der Schaltübergänge in einem automatischen Fahrzeuggetriebe aufgebaut werden. Dieses besteht aus Planetenradsätzen, elektrohydraulisch angesteuerten Schaltelementen wie Kupplungen und Bremsen sowie Freiläufen [72]. Um die tatsächlichen Drehmomentverhältnisse am Getriebeein- bzw. ausgang korrekt nachzubilden, muß das Modell auch die Komponenten Motor, hydrodynamischer Drehmomentwandler, Differentialgetriebe, Fahrzeuglängsdynamik (Fahrzeugträgheit und Fahrwiderstände) sowie die relevanten Funktionen der elektronischen Steuerungen von Motor und Getriebe enthalten.

Bild 14 zeigt den schematischen Aufbau des Vierganggetriebes ZF 4HP22 [72] (C_i sind Kupplungen bzw. Bremsen), Bild 13 das entsprechende Objektdiagramm (ein Bildschirmabzug des Programs Dymola [70]). Dank akausaler, objektorientierter Modellierung entspricht nicht nur jeder Baugruppe des Getriebes genau einer Komponente im Simulationsmodell sondern auch die Verbindungen der einzelnen Baugruppen sind, anders als bei einer signalorientierten Modellierung, in der Schemazeichnung und in dem Simulationsmodell identisch. Dies ermöglicht auch bei strukturvariablen Systemen (das Beispiel hat 7 Kupplungen/Bremsen C_i und damit $2^7 = 128$ mögliche Strukturen und $3^7 = 2187$ unterschiedliche Schaltzustände) die Wiederverwendung der Modellkomponenten im Simulationsmodell eines anderen Getriebes ohne jede Modifikation, was mit anderen Modellierungsmethoden nicht möglich ist. Gleichzeitig ist diese Modellierung bei Verwendung eines geeigneten Simulationsprogramms [70] so effizient, daß sie auch für Echtzeitanwendungen problemlos verwendet werden kann [78, 71].

Gang	C4	C5	C6	C7	C8	C11	C12
1	x					x	
2	x		x	x		x	
3	x	x		x		x	
4	x	x		x			x
R		x			x	x	

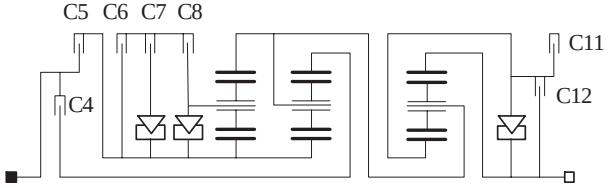


Bild 14: Getriebeschema des Automatikgetriebes ZF 4HP22.

Literatur

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

[71] *Elmqvist, H., Otter, M.* und *Schlegel, C.:* Physical Modeling with Modelica and Dymola and Realtime Simulation with Simulink and Real Time Workshop. 1997 Matlab User Conference, San Jose, U.S.A., 6.-8. Okt., 1997. (http://www.modelica.org/papers/mlconf.ps)

[72] *Förster, H.J.:* Automatische Fahrzeuggetriebe. Springer, 1991.

[73] *Rubin, Z.J., Munns, S.A.* und *Moskwa, J.J.:* The Development of Vehicular Powertrain System Modeling Methodologies: Philosophy and Implementation. 1997 SAE Congress and Exposition, Detroit, SAE paper No. 971089.

[74] *Jacobson, B.:* Modular Simulation Tool for Vehicle Propulsion concerning Energy Consumption and Emissions. 1997. http://www.mvd.chalmers.se/%7Ebeja/simproj/finalreport/webrepor.htm

[75] *Laschet, A.:* Simulation von Antriebssystemen. Springer, 1988.

[76] *Modelica package*. Homepage: http://www.modelica.org/

[77] *Otter, M.*: Object-Oriented Modeling of Drive Trains with Friction, Clutches and Brakes. Proc. ESM94 European Simulation Multiconference, Barcelona, Spanien, S. 335-340, 1994.

[78] *Schlegel, C.* und *Kessler, F.*: Modellierung und Hardware-in-the-Loop Simulation von Automatikgetrieben. "Hardware-in-the-Loop Simulation in der Fahrzeugentwicklung", Haus der Technik e.V., Essen, 1998

Manuskripteingang: ???