System-Dynamics-Diagramme als “Physical Modeling” - eine Frage der Kausalität

Peter Junglas[1]
[1] FHWT Vechta/Diepholz/Oldenburg
peter@peter-junglas.de

System-Dynamics-Diagramme gelten als sehr einfaches Modellierungswerkzeug, das sich leicht mit traditionellen Methoden nachbilden lässt. Wie einige Beispiele zeigen, ist die Situation jedoch komplizierter: Die Kausalität - also die Frage, welche Anschlüsse eines Blocks Ein- oder Ausgänge sind - ist in manchen Fällen vom Zustand des Systems abhängig. Die Entwicklung der hier vorgestellten Modelica-Bibliothek erforderte daher einige Vorüberlegungen, die im weiteren ausführlich dargelegt werden.

1 Einleitung

Die System-Dynamics-Methode ist ein Modellierungsverfahren, das vor allem in nicht-technischen Gebieten wie der Ökonomie oder der Ökologie verwendet wird [1]. Kommerzielle Programme zur Erstellung und Simulation von System-Dynamics-Diagrammen sind weit verbreitet (z. B. Stella von isee systems [2]), es gibt aber auch eine frei verfügbare Implementation, die auf Modelica basiert [3].

Angesichts der grundsätzlich sehr einfachen Struktur der Diagramme könnte man annehmen, dass sie sich leicht durch geeignete Blöcke etwa in Simulink implementieren lassen. Es stellt sich aber heraus, dass in einigen Situationen die Signalflussmethode nicht gut zur Nachbildung geeignet ist. Der Grund dafür ist, dass die Kausalität der Blöcke - also die Frage, welche Anschlüsse Eingänge bzw. Ausgänge sind - nicht immer durch das Diagramm festgelegt ist, sondern sich abhängig vom Zustand des betrachteten Systems ändern kann. Für die Erstellung einer System-Dynamics-Bibliothek eignen sich daher auf “Physical Modeling” basierende Methoden besser, bei denen die Kausalität eines Blocks sich erst dynamisch im Kontext des gesamten Systems ergibt.

Im Folgenden wird das grundlegende Problem an einfachen Beispielen erläutert und eine in Modelica programmierte System-Dynamics-Bibliothek vorgestellt. Sie wurde für das Lehrbuch [4] erstellt und kann von der Homepage des Autors frei heruntergeladen werden [5]. Die bereits vorhandene Modelica-Implementierung von Cellier et al. [3] beachtet die hier betrachteten Effekte nicht, da sie vor allem im Hinblick auf die bekannten Weltmodelle erstellt worden ist, bei denen keine Kausalitätprobleme auftreten.

2 Grundlegende System-Dynamics-Diagramme

System-Dynamics-Diagramme setzen sich i. W. aus nur drei verschiedenen, aber sehr allgemeinen Grundbausteinen zusammen: Die Zustandsgrößen werden als Speicherelemente (**Reservoirs**) dargestellt, die ihren Wert durch Zufluss bzw. Abfluss ändern. Ventilbausteine (**Flows**) bestimmen die Stärke der Flüsse, sie verlaufen zwischen Reservoiren oder externen Quellen und Senken. Zur Berechnung können sie auf die Werte von Zwischengrößen zurückgreifen, die mit **Convertern** bestimmt werden. Abb. 1 zeigt diese Blöcke im Zusammenhang.

Als Beispiel werde ein einfaches Modell zur Bevölkerungsentwicklung betrachtet: Die Bevölkerungsgröße N ändert sich durch die Zahl G der Geburten pro Zeit und die Zahl T der Todesfälle pro Zeit. Für G wird eine konstante Rate g angenommen, während T zu-

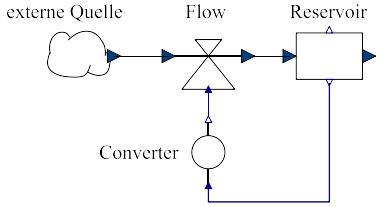

Abbildung 1: System-Dynamics-Basisblöcke

sätzlich eine feste Kapazitätsgrenze N_k enthält:

$$G = gN$$
$$T = tN \quad \text{mit} \quad t = \frac{t_0}{1 - N/N_k}$$

Im fertigen Modell (s. Abb. 2) werden die Parameter g, t_0 und N_k jeweils in Convertern bereitgestellt, ein weiterer Converter berechnet daraus die Rate t. Die Flows schließlich multiplizieren einfach ihre beiden Eingänge, um die Flüsse G bzw. T zu erhalten. Das Diagramm zeigt nur die grundsätzlichen Zusammenhänge zwischen den Größen, die konkreten Formeln sind implizit in den Blöcken enthalten.

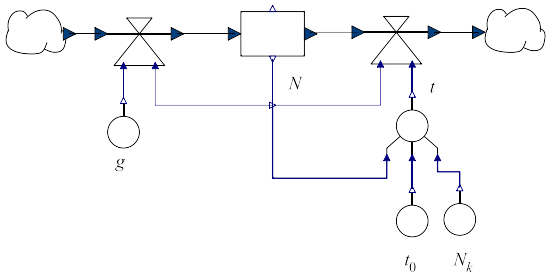

Abbildung 2: Modell `bevoelkerung`

Die Kausalität der Bausteine ist hier ganz klar: Converter haben mehrere Eingänge und einen Ausgang, die untereinander oder mit entsprechenden Eingängen der Flows verbunden werden. Ein Flow berechnet daraus den Wert des Flusses und gibt ihn als Ausgang an die angeschlossenen Reservoire weiter. Dabei darf man sich durch die Pfeile im Diagramm nicht verwirren lassen: Diese zeigen nur die (positive) Richtung der Flüsse an, die Signalflüsse dagegen verlaufen immer von den Flows zu den Reservoiren. Diese subtrahieren schließlich ihre beiden Eingangswerte und bestimmen durch einfache Integration den Wert der Zustandsgröße, der zusätzlich als Ausgangswert bereitgestellt wird. Dieses Prinzip wurde bei der Modelica-Bibliothek von [3] verwendet und lässt sich leicht auch in Simulink implementieren.

3 Modelle mit variabler Kausalität

3.1 Reservoir mit Sättigung

Ein Reservoir-Baustein kann optional einen Minimal- und einen Maximalwert angeben. Im Beispielmodell `abfluss` (s. Abb. 3) hat das erste Reservoir S_1 einen Minimalwert von 0 und einen Startwert von 4, der abgehende Fluss ist konstant 0.5. Das Ergebnis der Simulation zeigt Abb. 4: Aufgrund des konstanten Abflusses nimmt der Wert von S_1 linear ab, bis bei t = 8 der Minimalwert 0 erreicht ist, und bleibt dann konstant. Das nachgeschaltete Reservoir S_2 verhält sich gegenläufig, insbesondere bleibt auch sein Wert ab t = 8 konstant.

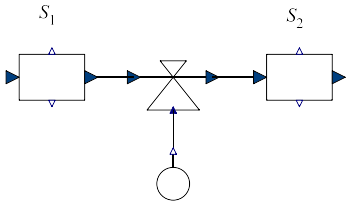

Abbildung 3: Modell `abfluss`

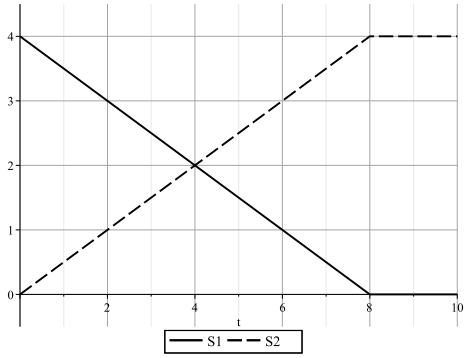

Abbildung 4: Ergebnis von `abfluss`

Auf den ersten Blick scheint es, als ließe sich dieses Modell leicht in Simulink nachbilden, indem man bei den Integratoren `Saturation limits` einführt. Aber

das Problem liegt tiefer: Zwar lässt sich der Wert von S_1 leicht auf 0 beschränken, aber das folgende Reservoir S_2 steigt trotzdem immer weiter an. Man muss vielmehr dafür sorgen, dass ab t = 8 der Fluss aus S_1 und damit auch in S_2 verschwindet. Es liegt hier ein Kausalitätsproblem vor: Bis t = 8 bestimmt der Flow-Baustein die Größe des Flusses, danach aber legt S_1 den Fluss auf 0 fest.

3.2 Modellierung einer Fertigungsmaschine mit `Oven`

Besonders deutlich wird das Problem beim Block `Oven`, einem diskreten Modell für eine generische Fertigungsmaschine. Er hat die drei Parameter `initialLoad`, `capacity` und `cookingTime` und verhält sich wie ein einfaches Backblech: Er wird zunächst gemäß seines Eingangsflusses beladen, bis die Kapazität erreicht wurde. Anschließend beginnt die Verarbeitungszeit, an deren Ende der gesamte Inhalt als Ausgangsfluss erscheint. Das grundsätzliche Verhalten des Blocks zeigt das Modell `oven1` (s. Abb. 5) mit den Parametern `initialLoad = 0`, `capacity = 3` und `cookingTime = 2`. Der Eingangsfluss ist auf den konstanten Wert 2 eingestellt, der Ausgangsfluss auf den Wert 1.

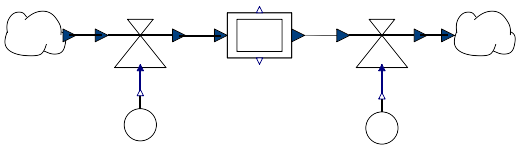

Abbildung 5: Modell `oven1`

Der Plot in Abb. 6 zeigt oben den Eingangs- und Ausgangsfluss, unten den Beladungszustand des `Oven`. Zur Zeit t = 2 gelangen zunächst zwei Einheiten, dann nur noch eine in den `Oven`. Bei t = 3 ist er voll beladen, die Verarbeitungszeit beginnt. Danach, bei t = 5, werden drei Einheiten herausgegeben, gleichzeitig kommen schon die ersten zwei neuen Teile an. Das genaue Timing, insbesondere die Überlappung, sind natürlich diskutabel. Als Vorbild diente das Verhalten des entsprechenden Blocks im Programm Stella.

Die Größe des Eingangsflusses hängt hier in komplizierter Weise vom vorgeschalteten Flow-Baustein und vom Zustand des `Ovens` ab: Während der Beladungsphase gibt der Flow den Wert vor, bei Erreichen der Kapazität arbeiten Zustand des Ovens und Flow-Wert zusammen, während der Bearbeitungsphase setzt der Oven den Eingangsfluss auf Null. Der Ausgangsflow wird dagegen komplett vom Zustand des `Oven` bestimmt: Während der Beladungs- und Bearbeitungsphase ist er Null und steigt nur während des Ausladens auf den Wert `capacity`. Der im nachgeschalteten Flow-Baustein vorgegebene Wert wird völlig ignoriert.

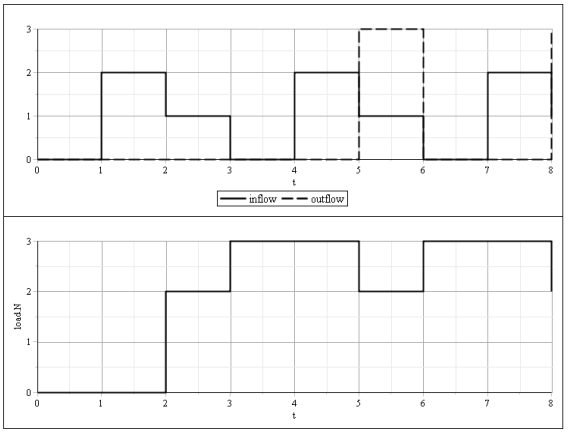

Abbildung 6: Ergebnis von `oven1`

Noch komplizierter wird es, wenn man das Modell `oven1` um Reservoir-Blöcke S_1 und S_2 an Ein- bzw. Ausgang erweitert: Wenn S_1 leer läuft – und nach unten auf Null beschränkt ist –, ergibt sich sein Ausgangsfluss durch seinen Inhalt, die (Maximal-)Größe des Flusses und den Beladungszustand des Ovens. Ergebnisse eines solchen Simulationslaufs zeigt Abb. 7.

Und sollte das Ausgangsreservoir nach oben beschränkt sein, bricht die Simulation zusammen: Der `Oven` will seinen Inhalt abgeben, aber das Ausgangslager hat nicht mehr genug Platz. Natürlich ist das eine Situation, die man nicht nur im Modell vermeiden will.

3.3 Modellierung von `Oven` in Simulink

Dass die Kausalität der Verbindungen sich mit dem Zustand ändern kann, heißt natürlich nicht, dass sich ein solches Modell mit der Signalflussmethode, also etwa in Simulink, nicht erstellen ließe. Allerdings

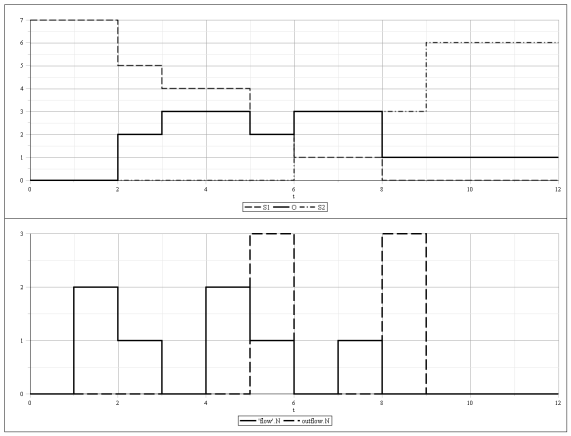

Abbildung 7: Ergebnis von `oven2`

muss man die Kausalitätsprobleme hier selbst lösen. Dazu erhält der Flow-Block neben seinem Eingang für den gewünschten Flow-Wert zwei weitere Eingänge, die die maximal möglichen Flüsse der beiden benachbarten Reservoire bekommen. Durch einfache Bildung des Minimums wird daraus der tatsächliche Fluss bestimmt. Auch der Block `Oven` lässt sich mit etwas Mühe mit Bausteinen der Standardbibliothek aufbauen, z. B. indem man seinen Modelica-Code nachbildet. Er hat einen Eingang für den Eingangsfluss sowie zwei Ausgänge für den Ausgangsfluss und den aktuellen Beladungszustand. Dazu kommt ein weiterer Ausgang, der den momentan möglichen maximalen Eingangsfluss angibt. Daraus lässt sich leicht das Modell `oven1` erstellen (s. Abb. 8), das die obigen Werte reproduziert.

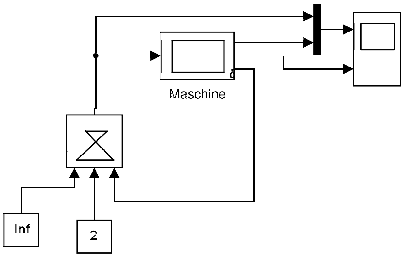

Abbildung 8: Modell `oven1` in Simulink

Das Modell lässt sich leicht um zwei Standard-Reservoire erweitern, wobei das Eingangslager in analoger Weise seinen Restbestand an den nachgeschalteten Flow meldet. Allerdings tritt hier ein Problem am Ausgang des `Oven`-Blocks auf: Er gibt seinen Ausgangsfluss in voller Größe ab, auch wenn der Flow einen kleineren Wert vorgesehen hat. Um dies nachzubilden, enthält der Flow-Baustein als Parameter die Option, bei der Berechnung des Flusses seinen “Wunschwert” einfach zu ignorieren. Das fertige Modell (s. Abb. 9) reproduziert wieder die Ergebnisse von oben.

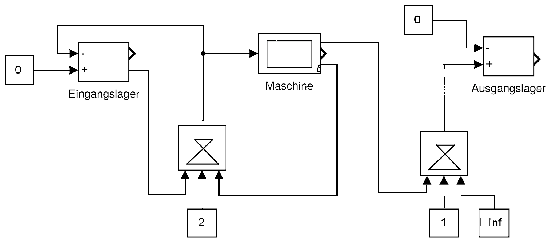

Abbildung 9: Modell `oven2` in Simulink

Trotz dieser Erfolge erscheinen die Simulink-Modelle sehr adhoc, sie lassen die Einfachheit und Flexibilität der ursprünglichen System-Dynamics-Diagramme vermissen. Außerdem entsteht aufgrund der starren Regeln für die Anordnung von Ein- und Ausgängen am Block ein ziemlicher “Drahtverhau” an Signalleitungen. Eine etwas einfachere, aber weniger flexible Implementierung der hier betrachteten Beispiele wird in [4] beschrieben.

4 Modelica als Basis einer System-Dynamics-Bibliothek

4.1 Bibliotheken in Modelica

Die Frage der Kausalität stellt sich in auf Modelica basierenden Physical-Modeling-Systemen in ganz anderer Weise [6]: Statt festzulegen, welche Größen als Eingangswerte verwendet werden, um damit andere als Ausgangswerte zu berechnen, werden für jeden Block nur die verknüpfenden Gleichungen angegeben, ohne dass sie nach bestimmten Größen aufgelöst sind. Zusätzliche Gleichungen ergeben sich bei Verbindungen der Blöcke. Dazu unterscheidet man zwei Typen von Größen: Flow-Variable addieren sich an Verbindungspunkten zu Null – sie entsprechen oft Zeitableitungen von Erhaltungsgrößen –, Potential-Variable haben an einer Verbindung einen konstanten Wert.

Wie sich das aus diesen Gleichungen schließlich ergebende System – häufig ein DAE-System – simulieren lässt, ist ein schwieriges Problem, das zum Glück inzwischen gut verstanden ist [7]. Die dazu entwickelten Algorithmen wurden in Modelica-basierten Simulationsprogrammen wie Dymola oder MapleSim implementiert.

Die in Modelica erreichte hohe Flexibilität hat allerdings vor allem bei der Entwicklung von Block-Bibliotheken ihren Preis: Da nicht festgelegt wird, welche Größe wo berechnet wird, ist nicht automatisch gewährleistet, dass die Verknüpfung beliebiger Blöcke zu einem geschlossenen System führt, bei dem die Zahl der Gleichungen und Variablen gleich groß sind. Abhilfe schafft das in [8] vorgestellte Verfahren: Man definiert an den Verbindungspunkten (**Connectoren**) jeweils gleich viele Variable vom Flow- und Potential-Typ und gibt jedem Block so viele Gleichungen, wie er externe Flow-Variable hat. Darüber hinaus kann man noch "gewöhnliche" Signalleitungen definieren, die als Ein- oder Ausgänge gekennzeichnet sind. Hier benötigt jede Ausgangsgröße natürlich eine entsprechende Gleichung im Block.

4.2 Konzeption des Connectors

Diese Überlegungen sollen nun beim Aufbau einer System-Dynamics-Bibliothek in Modelica angewendet werden. Startpunkt ist dabei die Definition eines geeigneten Connectors `MassPort`, der die Größe eines Flows als Modelica-Flow-Variable `dm` enthält, entsprechend der integrierten Zustandsgröße `m`. Zwei Fragen müssen nun geklärt werden:

1. Welche Größe kommt als `dm` begleitende Potential-Variable in Frage?
2. Wie werden die Gleichungen auf die Blöcke, insbesondere auf Reservoirs und Flows, aufgeteilt?

Motiviert von der in Abschnitt 2 vorgestellten Grundsystematik von System-Dynamics-Modellen soll die Integration des Flusses im Reservoir geschehen, das dazu folgende Gleichung enthält:

```
der(m) = inflow.dm + outflow.dm;
```

Dabei ist, wie in Modelica üblich, eine Flow-Variable positiv, wenn der Fluss in den Block einströmt.

Ein Flow-Block dagegen berechnet den Wert von `dm` mit Hilfe seiner Eingangsgrößen. Dabei muss er aber, wie in Abschnitt 3 deutlich wurde, auch die Zustände der angeschlossenen Reservoire berücksichtigen. Daher schicken diese die benötigten Informationen als reelle Variable `data`, die die Rolle der Potential-Variablen übernimmt. Angesichts der oben vorgestellten Beispiele gibt es drei Möglichkeiten, wie der Wert von `data` verwendet werden kann:

1. gar nicht, das Reservoir übernimmt den vom Flow vorgegebenen Wert,
2. der Flow wird auf `data` gesetzt,
3. der Flow wird durch `data` begrenzt.

Fall 1 entspricht der "Standard-Situation", Fall 2 z. B. dem Ausgang des `Oven`, Fall 3 tritt bei Reservoiren mit Sättigung oder bei der Beladung eines `Oven` auf.

Dieses Verhalten ließe sich implementieren, indem man im 1. Fall `data = 0` setzt und die beiden anderen Fälle durch das Vorzeichen von `data` unterscheidet. Dies verhindert aber die Möglichkeit, in zukünftigen Erweiterungen neue Verwendungsmöglichkeiten von `data` vorzusehen. Außerdem ist der Test auf Null bei reellen Variablen generell eine schlechte Idee. Daher wird der Connector um eine ganzzahlige Variable `info` erweitert, mit der ein Reservoir den entsprechenden Fall anzeigt. Diese Größe hat eine fest vorgegebene Kausalität: Sie wird im Reservoir berechnet und im Flow verwendet. Es müssen also zwei Varianten des Connectors verwendet werden, bei denen `info` als `output` bzw. `input` gekennzeichnet ist:

```
connector MassPortR
  "mass port of reservoirs"
  flow Real dm;
  Real data;
  output Integer info;
end MassPortR;

connector MassPortF
  "mass port of flows"
  flow Real dm;
  Real data;
  input Integer info;
end MassPortF;
```

4.3 Aufbau der System-Dynamics-Bibliothek

Nach diesen Vorüberlegungen lässt sich nun eine System-Dynamics-Bibliothek ohne Probleme aufbauen. Sie enthält die vier Unterpakete:

- `Interfaces`
- `Reservoirs`
- `Converters`
- `Flows`

In `Interfaces` befinden sich die Definitionen der Connectoren, Basisklassen und Hilfsfunktionen. Sie enthält insbesondere die Funktion `constrainedRate`, die den von einem Fluss vorgegebenen Wert mit den `data`- und `info`-Werten zweier `MassPorts` kombiniert, um den tatsächlichen Fluss zu bestimmen. Diese Berechnung wird über die Basisklasse `GenericFlow` in alle Flow-Bausteine integriert.

`Reservoirs` enthält als Standard-Reservoire die Blöcke `Stock` und `SaturatedStock` sowie `CloudSource` und `CloudSink` für externe Quellen bzw Senken. Zum besprochenen Block `Oven` kommt noch `Conveyor` hinzu, der ein einfaches Modell für ein Transportband darstellt. Beide sind zeitdiskret, sie arbeiten mit vorgegebener Samplerate. Ergänzend gibt es auch diskrete Versionen `StockD` und `SaturatedStockD`.

Die Blöcke in `Flows` enthalten neben zwei `MassPorts` ggf. zusätzliche Signal-Eingänge zur Berechnung der Flüsse aus Hilfsgrößen. Die Blöcke in `Converters` sind ausschließlich mit Signal-Anschlüssen versehen, sie sind daher in Modelica als `block` definiert worden. In auf System-Dynamics-Diagramme spezialisierten Programmen gibt es in der Regel nur je einen `Converter`- und `Flow`-Block. Die benutzten Beziehungen werden als Parameter der Blöcke definiert und die Zahl der Eingänge entsprechend angepasst. Leider bieten die auf Modelica basierenden Programme keine solche Möglichkeit. Daher wurden für die häufigsten Verknüpfungen fertige Blöcke in den entsprechenden Unterbibliotheken bereitgestellt. Eine Besonderheit stellen `SwitchedConverter` und `TimeSwitchedConverter` dar, die in Abhängigkeit von der Eingangsgröße oder der Zeit zwischen zwei Werten umschalten, sowie der `GraphConverter`, der einen funktionalen Zusammenhang durch Interpolation von Tabellenwerten herstellt, die aus einer Datei eingelesen werden. Weitere Blöcke lassen sich durch Ableiten von Basisblöcken mit wenigen Zeilen Modelica-Code erstellen.

5 Schlussfolgerungen

Für die Entwicklung einer flexiblen System-Dynamics-Bibliothek sind die im Physical Modeling gegebenen Möglichkeiten einer variablen Kausalität zumindest sehr hilfreich. Allerdings hätte man die im Connector verwendeten Flow- und Potential-Größen auch komplett als Signalgrößen definieren können. Die Frage der Kausalität hängt also auch davon ab, was man als "eine" Signalleitung auffasst. Auf jeden Fall eignen sich die hier vorgestellten Ideen gut dazu, in der Lehre den Studierenden einen ersten Einblick in die Problematik der Kausalität zu geben.

Die konkrete Umsetzung leidet momentan an Beschränkungen der Tools, es fehlt insbesondere die Möglichkeit, einen funktionalen Zusammenhang als Parameter eines Blocks einzugeben. Auch das im Detail unterschiedliche Verhalten des `pre`-Operators in Dymola und MapleSim und des 1/z-Blocks in Simulink erschwerte die Arbeit nicht unerheblich. Hier besteht wohl noch ein gewisser Klärungs- bzw. Verbesserungsbedarf.

Zwar sind für die wichtigsten Zusammenhänge fertige Blöcke in der System-Dynamics-Bibliothek vorhanden, weitere lassen sich durch Ableiten vorhandener Blöcke mit wenigen Zeilen Modelica-Code leicht erstellen. Der typische Anwender von System-Dynamics-Software hat aber wenig bis gar keine Programmiererfahrung und ist von anderen Tools ein einfacheres Vorgehen gewöhnt. Für den Einsatz in der Lehre, speziell im Kontext verschiedener Modellierungsverfahren, ist diese Einschränkung aber nicht gravierend.

Literatur

[1] B. Hannon und M. Ruth. *Dynamic Modeling.* Springer, New York, 2. Auflage 2001.

[2] B. Richmond, S. Peterson und P. Vescuso. *An Academic User’s Guide to STELLA.* High Performance Systems, Inc., Lyme, N.H., 1987.

[3] F. E. Cellier. *World3 in Modelica: Creating System Dynamics Models in the Modelica Framework.* Proceedings of the 6th International Modelica Conference, Bielefeld, Germany, S. 393 – 400, 2008.

[4] P. Junglas. *Praxis der Simulationstechnik.* Europa-Lehrmittel, Haan-Gruiten, 2014.

[5] P. Junglas. *Homepage von “Praxis der Simulationstechnik”.* Online: http://www.peter-junglas.de/fh/publications/simulation/index.html (Aufruf 2014-06-13).

[6] P. A. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley & Sons, New York, 2004.

[7] F. E. Cellier und E. Kofman. *Continuous System Simulation.* Springer, New York, 2010.

[8] H. Olsson, M. Otter, S. E. Mattsson und H. Elmqvist. *Balanced Models in Modelica 3.0 for Increased Model Quality.* Proceedings of the 6th International Modelica Conference, Bielefeld, Germany, S. 21 –33, 2008.