OBJECT–ORIENTED MODELING OF POWER–ELECTRONIC CIRCUITS USING DYMOLA

Hilding Elmqvist
Dynasim AB
Research Park Ideon
S–223 70 Lund
Sweden
Elmqvist@Dynasim.SE

François E. Cellier
Dept. of Electr. & Comp. Engr.
The University of Arizona
Tucson, Arizona 85721
U.S.A.
Cellier@ECE.Arizona.Edu

Martin Otter
Inst. für Robotik & Systemdynamik
DLR Oberpfaffenhofen
D–82230 Wessling
Germany
DF43@Master.DF.OP.DLR.DE

ABSTRACT

In this paper, a new approach to the object–oriented modeling of power–electronic circuits is demonstrated. It enables the user to specify power–electronic circuits conveniently in an easy–to–use modular fashion, yet generate simulation code that is efficient in its use, not requiring the introduction of artificial fast time–constants as was the case with many of the earlier proposed methodologies. Dymola enables the user to specify models for individual circuit elements in a highly modular, compact, and object–oriented fashion. Circuit models invoke these component models, and connect them in a topological manner, just as a Spice program would. The Dymola compiler automatically translates these circuit models into monolithic descriptions at the level of the simulation language resolving discontinuous circuit elements, such as switches, into appropriate event descriptions. For a typical AC–DC converter circuit controlled by GTO thyristors, it is shown that a speedup factor of about 20 can be achieved with respect to an equivalent PSpice model.

INTRODUCTION

Power–electronic circuits form a central part of most control systems. They are at the heart of the *actuators* that translate information signals, i.e., the output signals of the controllers, into power signals, i.e., the signals that drive the actual plant to the desired operating point. The energy for these power signals is taken out of the electric net, i.e., actuators are power converters that convert a fixed power signal (usually either ac or dc) into a variable power signal under the control of an information signal. A typical actuator is shown schematically in Fig.1. The modulation (or regulation) of power signals almost invariably employs high–frequency switching devices, thus a continuous system simulation language that is not equipped to deal with *discontinuities* in the trajectories to be integrated across cannot be used to simulate power–electronic circuits.

A general–purpose continuous system simulation language, such as ACSL (Mitchell & Gauthier 1991), is principally capable of providing efficient simulations of electronic switching circuitry, but the tool is hardly ever used for such purpose, since it is a very difficult task to convert the topological description of a complex electronic switching circuit diagram into a state–space model with schedule statements and discrete sections for discontinuity handling as would be required by ACSL.

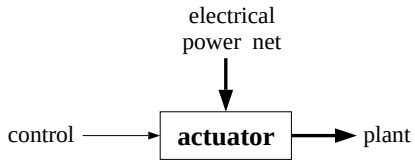

Figure 1: Schema of a typical actuator.

Most circuit designers today use dialects of Spice (Nagel 1975) to simulate switching circuitry. However, while Spice provides for a convenient input description language, the underlying software is hardly suited for simulating switching circuitry. If the switching devices are modeled in a detailed fashion using physical models of the components that implement the switching characteristics, such as power MOS or BJT devices, the simulation will run correctly since the model doesn't contain any discontinuities at all, but the simulation engine will have to employ extremely small step sizes in order to follow the fast transients that respresent the true switching behavior of the switching devices. This is appropriate for simulating one, or two, or maybe 10 switching characteristics to investigate the transient behavior of the switching device itself, e.g., to check whether any sort of ringing takes place after the device has switched, but it is hardly suitable to simulate the hundreds or even thousands of switching events during analysis of a cycloconverter. For such purposes, a coarser grain simulation is needed.

Some Spice dialects offer a switch element, in which the switch is represented by a variable resistor. If the switch is *closed*, the resistance assumes a small value, and when it is *open*, the resistance is large. However, the integration routines within Spice still have problems when the resistance suddenly changes. The discontinuity forces the step size to

be decreased. The reasons for these difficulties are explained for example in (Cellier *et al.* 1993).

Modern integration routines handle discontinuities and events more efficiently and accurately. They require more information about the model though. Information about discontinuities is provided by means of *indicator functions*. Whenever an indicator function crosses through zero, this indicates to the integration algorithm that a discrete event is about to occur. Instead of decreasing the step size, the integration routine interpolates the indicator function in order to locate the zero crossing with sufficient accuracy. The time of the event is thus determined in an efficient way. After the discrete action associated with the event, such as a sudden change of a resistance, has been simulated, the integration algorithm is restarted.

Dymola offers an attractive alternative to Spice for simulating switching circuitry. Dymola is an object–oriented modeling language (Elmqvist 1978, Dynasim 1994) with powerful discontinuity handling capabilities. Discontinuous models are described in an object–oriented fashion using *if* expressions and *when* statements as explained in detail in (Elmqvist *et al.* 1993). These expressions/statements are translated into event descriptions at the level of the underlying simulation software. To this end, both the discrete event actions and their associated indicator functions for efficient event detection are automatically being generated by the Dymola compiler.

Dymola employs highly sophisticated and efficient symbolic formulae manipulation algorithms that are able to translate a topological description of a higher index model down to state–space form, e.g. to an ACSL simulation program or to Fortran code (Cellier and Elmqvist 1993). The distance in conceptual abstraction between the modeling surface (i.e., the Dymola program) and the representation at the level of the simulation program (e.g. an ACSL program) that is being bridged by the Dymola compiler is much larger than with most other modeling tools. Dymola combines the best of two worlds. It provides the user with an object–oriented modeling surface that is as convenient to use as that of Spice, yet a simulation run–time engine that employs the latest in simulation technology.

Omola (Andersson 1990) is another object–oriented modeling tool, in many ways quite similar to Dymola. Also Omola is capable of reducing higher index models (Mattsson and Söderlind 1993). However, it always translates them into an index 1 differential–algebraic equation (DAE) description, and never into an index 0 ordinary differential equation (ODE) model. Dymola offers additional flexibility here, since the user can select between an ODE and a DAE representation at the level of the simulation language. Omola is also able to handle discrete events (Andersson 1992), but does so with explicit event descriptions, i.e., through lower–level constructs. Dymola offers a more convenient user interface for describing discontinuous models and is therefore preferred in the context of modeling switching circuitry.

Yet, when modeling switching circuitry, and especially when ideal switching elements are used, a lot can go wrong that the user needs to be aware of, and the modeling elements of Dymola can thus not be used carelessly. It is the purpose of this paper to show what can go wrong and what the user needs to know in order to successfully employ Dymola for modeling power–electronic switching circuits.

CIRCUIT MODELING IN DYMOLA

Let us start by modeling the simple circuit of Fig.2 in Dymola. All four components used in the model are two–pin

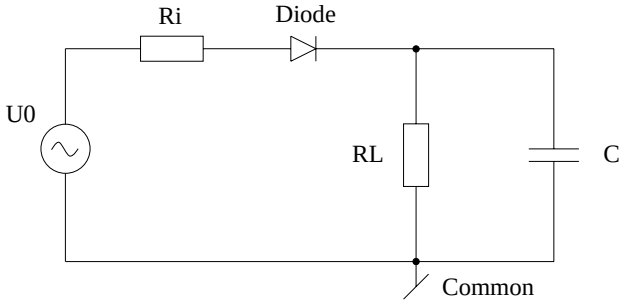

Figure 2: Half–wave rectifier circuit.

(one–port) devices that can be described in Dymola as:

```
model class TwoPin
  cut WireA(Va/i), WireB(Vb/-i)
  main cut PlugC[WireA,WireB]
  main path PathAB<WireA-WireB>
  local u
    u = Va - Vb
end
```

The *cut* declaration describes the interface of the two–pin device with its environment. Each *wire* carries two variables, a potential V (an *across* variable), and a current i (a *through* variable). The *path* declaration describes a logical connection from one wire to the other, used for connecting two–pin devices into a circuit. The local variable u describes the voltage drop across the device.

The resistor can be modeled as a specialization of the two–pin device:

```
model class (TwoPin) Resistor
  parameter R
    u = R*i
end
```

It inherits all the declarations and equations of the generic two–pin device, and extends them by the more specific declarations and equations that make the device behave like a resistor. Note that, in Dymola, equations such as "$u = R * i$"

are not assignment statements but "real" equations. Ohm's law could have been equally well defined as "$u - R * i = 0$".

The capacitor and the voltage source can be described as:

```
model class (TwoPin) Capacitor
  parameter C
    C * der(u) = i
end

model class (TwoPin) VSource
  terminal V0
    u = V0
end
```

Also the *ideal switch* element can be described directly as a two–pin device:

```
model class (TwoPin) IdealSwitch
  terminal OpenSwitch
    0 = if OpenSwitch then i else u
end
```

The switch equation states that either i or u are zero at all times, depending on whether the switch is open (*OpenSwitch=true*) or closed (*OpenSwitch=false*), respectively. Dymola is able to cope with such a variable structure equation as shown in (Elmqvist *et al.* 1993). Using this ideal switch component, it is now possible to model power–electronic devices such as ideal diodes:

```
model class (IdealSwitch) IdealDiode
  new(OpenSwitch) = u <= 0 and not i > 0
end
```

The circuit of Fig.2 can now be described as follows:

```
model Rectifier
  submodel (VSource)   U0
  submodel (Resistor)  Ri(R = 10), RL(R = 50)
  submodel (Capacitor) C(C = 0.001)
  submodel (IdealDiode) Diode
  submodel Common

  output    y
  parameter f = 50

  connect Common - ((U0 - Ri - Diode)//C//RL)

  U0.V0 = sin(2 * 3.14159 * f * Time)
  y     = C.u
end
```

where $-$ denotes a connection in series, and // indicates a connection in parallel.

How this model is translated to efficient simulation code is described in detail in (Elmqvist *et al.* 1993). Note that the ideal switch element is sometimes difficult to use, since several things can go wrong, as will be explained in due course.

A POWER CONVERTER WITH IDEAL SWITCHES

Let us discuss one of the basic configurations of power converters (Ramshaw 1993). Its circuit diagram is shown in Fig.3. The switches can be realized in many different

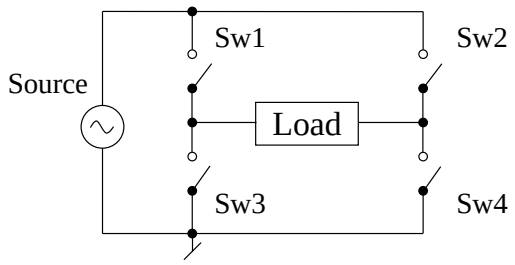

Figure 3: Circuit diagram of a power converter.

ways. Depending on the control of the switches, different types of power regulation are obtained. The normal mode of operation of this circuit is that either $Sw1$ and $Sw4$ are closed, whereas $Sw2$ and $Sw3$ are open, or vice–versa. In these two situation, the source voltage is applied to the load in one of two polarities.

What happens if *all four switches are open* at the same time? In this case, the load gets disconnected from the ground altogether. In electrical circuit modeling, there is always a need for one arbitrary assumption. Usually, this assumption will be that the potential of the ground is zero. This is not a truth statement, but simply a convention. When the load gets disconnected from the ground, the model "universe" suddenly decays into two separate "universes," and the modeler now needs to supply two arbitrary equations, one for each universe. Usually, this problem is resolved by attaching the load to the rest of the circuit (e.g. the ground) through an (arbitrarily selected) large resistor. Of course, if a non–ideal switch element modeled by means of variable resistors is used, this happens automatically, and the user need not to worry. In Dymola, such a resistive switch could be described as follows:

```
model class (TwoPin) VarResistor
  terminal R
    u = R * i
end

model class (VarResistor) ResSwitch
  terminal OpenSwitch
  parameter ROpen = 1.0E+6, RClose = 1.0E-6
    R = if OpenSwitch then ROpen else RClose
end
```

What happens if *either Sw1 and Sw3 or Sw2 and Sw4 are opened* at the same time? If the load contains an inductor, its current will not immediately cease to flow, but instead, an arc will appear. A common remidy for this type of problem is the introduction of a large shunt resistor placed in parallel with the inductor, or a freewheeling diode when converting to dc. The use of resistive switch elements will take care of this problem as well.

What happens if *either Sw1 and Sw3 or Sw2 and Sw4 are closed* at the same time? In this case, the source gets short-circuited, and an arbitrarily large current will flow. One way to overcome this problem is to model the source more realistically by adding an internal source resistance. Of course, the use of non–ideal switch elements will take care of the problem as well (not necessarily in a very realistic fashion, since voltage sources today on the market certainly don't come with internal resistance values of $2 \cdot 10^{-6}\ \Omega$).

Let us assume, the voltage source has been modeled with an internal source resistance, so that no unrealistically large currents can ever flow. What happens if *all four switches are closed* at the same time? Now, there are two wires connected in parallel, and it is not clear how the current that is drawn out of the shortcircuited source will be split between them. Again, the use of resistive switches will take care of the problem. The small resistances in the parallel branches serve as a current divider, which is exactly what physics will do in the real case as well.

What happens if the modeler does not take these situations properly into account? This kind of circuit exhibits a *variable structure*, i.e., the causality changes as a function of the switch position. When Dymola analyses the structure of the model equations, a set of equations is isolated that need to be solved simultaneously. The coefficients of these equations are time–varying. The opening and closing of switches corresponds to toggling certain coefficient values between zero and one (Elmqvist *et al.* 1993). If the above switch settings take place and proper remedies are not introduced in the model, the set of equations becomes singular, and it is no longer possible to solve the system of equations, i.e., the simulation has to be aborted.

To summarize, the usage of *ideal* switch elements requires special attention by the modeler. However, the problems associated with ideal switch elements can be easily circumvented by using a slightly more realistic switch model in which the switch is represented by a big yet not infinitely big resistance when it is open, and by a small yet not vanishingly small resistance when it is closed.

NON-IDEAL SWITCHES AND THYRISTORS

A *general switch* element with open and close resistances could be described as:

```
model class (TwoPin) GenSwitch
  terminal OpenSwitch
  parameter GOpen = 0.0, RClose = 0.0
    0 = if OpenSwitch then i - GOpen * u
                      else u - RClose * i
end
```

which behaves like the resistive switch when *GOpen* and *RClose* are different from zero, and degenerates to the ideal switch when the default values of both *RClose* and *GOpen* are in effect.

Using such a general switch, a Gate Turn–Off (GTO) thyristor (Ramshaw 1993) could be described as (cf. the ideal diode):

```
model class (GenSwitch) GTOThyristor
  main cut control(Gate)
    new(OpenSwitch) = u <= 0 and not i > 0 - >
                      or not Gate
end
```

Contrary to Spice, which offers a fixed set of basic circuit component models, new model types can be added to Dymola easily and in a compact and convenient fashion.

AC–DC CONVERTER WITH PULSE–WIDTH MODULATION CONTROL

As a more complete example, an AC–DC converter with pulse–width modulation (PWM) control will be studied. A Spice implementation of the same circuit is presented in (Rashid 1993). Its circuit diagram is shown in Fig.4.

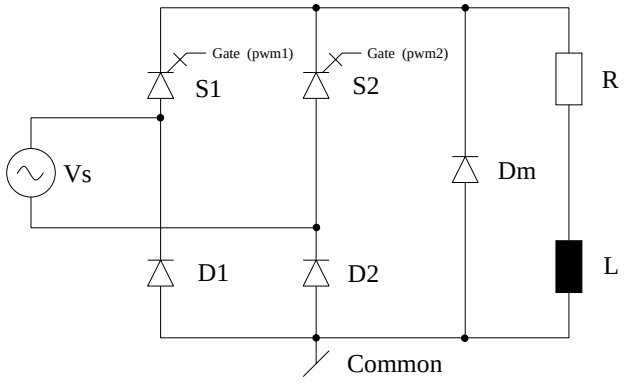

Figure 4: Circuit diagram of AC–DC converter.

It becomes evident from looking at Fig.4 that this AC–DC converter is just a special case of the generic converter scheme shown in Fig.3. Two gate turn–off thyristors S1 and S2 are used for control. The diodes D1 and D2 complete the bridge to obtain full–wave rectification. Since the load is inductive, a freewheeling diode (Dm) is needed in addition.

Different schemes for controlling the GTOs can be used. One scheme, called *sinusoidal pulse–width modulation* (Rashid 1993), will be studied. The pulses controlling the GTOs are determined by the intersection of a sinusoidal and a saw–tooth signal (Fig.5). When using Spice for simulation (Rashid 1993), the generation of these control pulses was done by modeling a "precision rectifier and comparator" including three operational amplifiers, nine resistors, and other circuit elements.

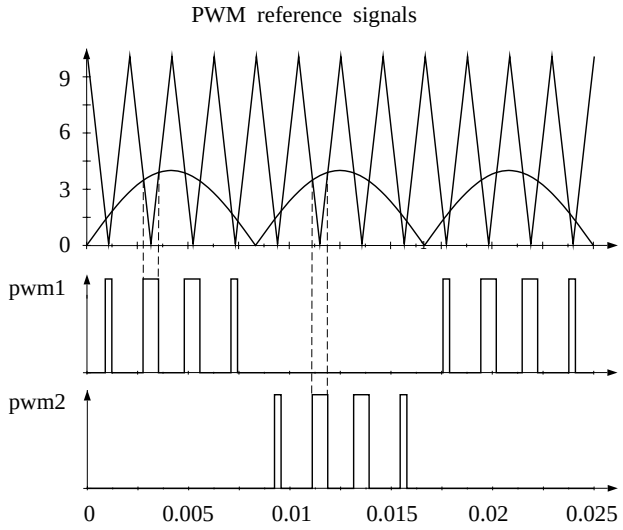

Figure 5: Sinusoidal pulse–width modulation.

In Dymola, this control scheme can be realized using the following much simpler behavioral description:

```
model class PWMcontrol
  main cut control(Gate)
  parameter p = 1, f, M
  constant PI = 3.14159265

  local Vr = 1, Vc, fr, square, rate

    { Generate saw-tooth reference signal. }
    fr = 2*p*f
    square = if sin(2*PI*fr*Time) > 0 then +1
                                      else -1
    rate = 2*fr
    der(Vr) = -rate*square

    { Generate carrier signal. }
    Vc = M*sin(2*PI*f*Time)

    { Determine output. }
    Gate = not (Vr < Vc)
end
```

The complete AC–DC converter can be described as follows:

```
model Converter

  submodel (Vsource)      Vs
  submodel (Resistor)     R(R = 2.5)
  submodel (Inductor)     L(L = 6.5E - 3)
  submodel (Diode)        D1(GOpen = 1E-6),  - >
                          D2(RClosed = 1E-6),  - >
                          Dm(GOpen = 1E-6)
  submodel (GTOThyristor) S1(RClosed = 1E-6),  - >
                          S2(RClosed = 1E-6)
  submodel Common

  submodel (PWMcontrol)   PWM1(p = 4, f = f, M = 0.4)
  submodel (PWMcontrol)   PWM2(p = 4, f = f, M = -0.4)

  parameter f = 60
  constant PI = 3.14159265

    connect Common - D2 - \Vs - S1
    connect S1 - ((R - L) // \Dm) - Common
    connect (D1 - S1)//(D2 - S2)
    connect PWM1 at S1
    connect PWM2 at S2

    Vs.V0 = 169.7*sin(2*PI*f*Time)
end
```

The results of the simulation are shown in Fig.6. The simulation time using a second–order variable step Runge–Kutta algorithm in ACSL on a Compaq 486, 66 MHz machine was 1.92 seconds. Simulating the same circuit in PSpice on the same computer required 44 seconds, i.e., the ACSL code ge-

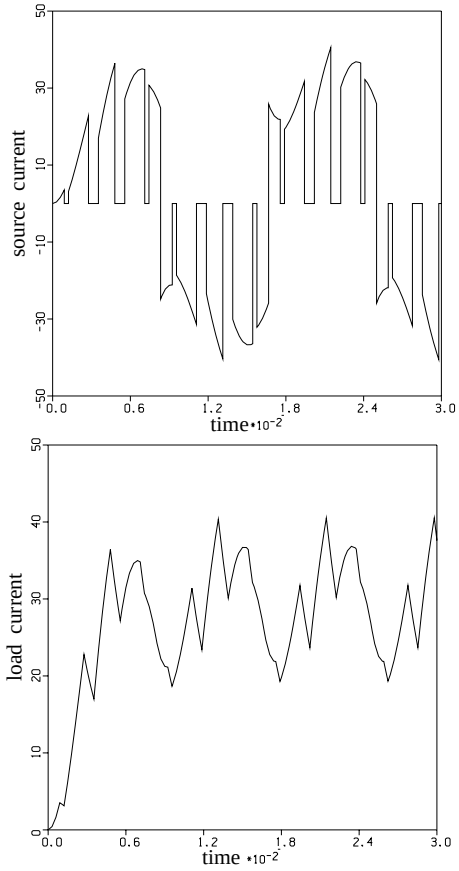

Figure 6: Source and load currents.

nerated by Dymola ran about 20 times more faster than the PSpice code.

How can this speedup be explained? One important factor is the use of indicator functions. Abusing the integration step–size control algorithms to locate discontinuities in a simulation model is very inefficient. A second important factor relates to the integration method itself. Dymola can be asked to translate implicit DAE models into explicit ODE models. Whereas a DAE solver (as it is used in Spice) iterates on all the model equations simultaneously, Dymola isolates systems of algebraic equations, and makes each system of simultaneous equations as small as possible. Linear systems of equations can then either be solved symbolically at compile time using a version of Cramer's rule, or they can be solved at run time by generating calls to LAPACK (Anderson *et al.* 1992). Nonlinear systems are always solved numerically at run time by generating calls to a nonlinear programming routine. Whether the symbolic or the numeric solution of linear systems is preferable depends on the application, i.e., it depends on whether the explicit solution destroys the inherent sparsity of the system or not. In the AC–DC converter example, a linear system of nine equations resulted, and in this example, it was indeed considerably more efficient to solve the system of equations symbolically at compile time. This reduced the run–time computations by about a factor of three. A third important factor relates to the use of ideal GTOs and diodes. Spice employs more accurate but unnecessarily inefficient models. The results obtained by ACSL are undistinguishable from those obtained by Spice. Finally in Spice, the control signal had to be generated using circuit elements. In Dymola, a behavioral description was used. Of course, when building the circuit, something similar to the Spice circuit will be needed, but for the purpose of the simulation, this detail was entirely unessential. It is a strength of Dymola that it allows to mix abstraction levels arbitrarily and easily.

CONCLUSIONS

In this paper, a new tool for flexibly and efficiently modeling and simulating power–electronic circuits was presented. Contrary to Spice, Dymola offers direct access to all model equations at an object–oriented easy–to–use equation level. It was shown that different phenomena can be modeled conveniently at different abstraction levels in accordance with the needs of the simulation attempted. In this way, highly efficient simulation code can be generated. It is also very easy to integrate the power–electronics models with the usually electro–mechanical plant models that the power–converters are supposed to drive, i.e., Dymola can be conveniently used for modeling mechatronic systems (Otter *et al.* 1994).

REFERENCES

Anderson, E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. 1992. *LAPACK User's Guide*, SIAM, Philadelphia.

Andersson, M. 1990. *Omola – An Object-Oriented Language for Model Representation*, Licenciate Thesis TFRT–3208, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Andersson, M. 1992. "Discrete Event Modelling and Simulation in Omola." In *Proc. IEEE Symposium on Computer–Aided Control System Design* (Napa, California, March 17–19), 262–268.

Cellier, F.E. 1979. *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*, Ph.D. Dissertation, Diss ETH No 6483, ETH Zürich, CH–8092 Zürich, Switzerland.

Cellier, F.E. 1991. *Continuous System Modeling*, Springer–Verlag, New York.

Cellier, F.E. 1995. *Continuous System Simulation*, Springer–Verlag, New York.

Cellier F.E. and H. Elmqvist. 1993. "Automated Formula Manipulation in Object–Oriented Continuous–System Modeling," *IEEE Control Systems*, no. 13 (2): 28–38.

Cellier, F.E., H. Elmqvist, M. Otter, and J.H. Taylor. 1993. "Guidelines for Modeling and Simulation of Hybrid Systems." In *Proc. IFAC World Congress* (Sydney, Australia, July 18–23), vol. 8, 391–397.

Dynasim. 1994. *Dymola — User's Manual*, Dynasim AB, Research Park Ideon, Lund, Sweden.

Elmqvist, H. 1978. *A Structured Model Language for Large Continuous Systems*, Ph.D. Dissertation, Report CODEN: LUTFD2/(TFRT–1015), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Elmqvist, H., F.E. Cellier, and M. Otter. 1993. "Object–Oriented Modeling of Hybrid Systems." In *Proc. ESS'93, European Simulation Symposium* (Delft, The Netherlands, October 25–28), xxxi–xli.

Mattsson, S.E. and G. Söderlind. 1993. "Index Reduction in Differential–Algebraic Equations Using Dummy Derivatives," *SIAM J. Sci. Comput.*, no. 14 (3): 677–692.

Mitchell & Gauthier Associates, Inc. 1991. *Advanced Continuous Simulation Language (ACSL) — Reference Manual*, Concord, Mass.

Nagel, L.W. 1975. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, Report ERL–M520, Electronic Research Laboratory, University of California Berkeley, Berkeley, Calif.

Otter, M., H. Elmqvist, and F.E. Cellier. 1994. "Modeling of Multibody Systems With the Object–Oriented Modeling Language Dymola," *J. Nonlinear Dynamics*, accepted for publication.

Pantelides, C.C. 1988. "The Consistent Initialization of Differential–Algebraic Systems," *SIAM J. Scientific and Statistical Computing*, no. 9 (2): 213–231.

Ramshaw, R.S. 1993. *Power Electronics Semiconductor Switches*, Chapman & Hall, London.

Rashid, M.H. 1994. *Spice for Power Electronics and Electric Power*, Prentice–Hall, Englewood Cliffs, N.J.