MODELICA — AN INTERNATIONAL EFFORT TO DESIGN THE NEXT GENERATION MODELING LANGUAGE

Sven Erik Mattsson

Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
E-mail: SvenErik@control.LTH.se

Hilding Elmqvist

Dynasim AB
Research Park Ideon
SE-223 70 Lund, Sweden
E-mail: Elmqvist@Dynasim.se

Abstract   A new language called Modelica for physical modeling is developed in an international effort. The main purpose is to make it easy to exchange models and model libraries. The design approach builds on non-casual modeling with true equations and the use of object-oriented constructs to facilitate reuse of modeling knowledge. There are already several modeling language based on these ideas available from universities and small companies. There is also significant experience of using them in various applications. The aim of the Modelica effort is to unify the concepts and design a new uniform model language for model representation. The paper describes the effort and gives an overview of Modelica.

Keywords   Modelica; modeling language; object-orientation; hierarchical systems; modeling; simulation; differential-algebraic equations

1. INTRODUCTION

Mathematical modeling and simulation are emerging as key technologies in engineering. Relevant computerized tools, suitable for integration with traditional design methods are essential to meet future needs of efficient engineering.

In October 1996 an effort started within the ESPRIT project "Simulation in Europe Basic Research Working Group (SiE-WG)" (18) to design a new language for physical modeling. The language is called Modelica. The main purpose is to make it easy to exchange models and model libraries and to allow for the end user to benefit from the advances in object-oriented modeling methodology. This paper presents the status of the Modelica design as of April 1997.

1.1 *Today's simulation tools*

There is a large amount of simulation software on the market. All languages and model representations are proprietary and developed for certain tools. There are general-purpose tools such as ACSL, SIMULINK, System Build, which representations are essentially based on the same modeling methodology, input-output blocks, as in the previous standardization attempt, CSSL, from 1967. There are domain oriented packages: electronic programs (SPICE, Saber), multibody systems (ADAMS, DADS, SIMPACK), chemical process (ASPEN Plus, SpeedUp) etc. With very few exceptions, all simulation packages are only strong in one domain and are not capable to model components from other domains in a reasonable way. This is a major disadvantage since technical systems are becoming more and more heterogeneous with components from many engineering domains.

1.2 *The state-of-the art*

Techniques for general-purpose physical modeling have been developed during the last decades, but it did not receive much attention from the simu-

lation market. The modern approaches build on non-causal modeling with true equations and the use of object-oriented constructs to facilitate reuse of modeling knowledge. There are already several modeling languages with such a support available from universities and small companies. Examples of such modeling languages are ASCEND (16, 15), Dymola (7, 6), gPROMS (3, 14), NMF (17), ObjectMath (8, 19), Omola (13, 1), SIDOPS+ (5), Smile (4, 12), U.L.M. (11, 10) and VHDL-AMS (2, 9). There is also significant experience of using these languages in various applications. The aim of the Modelica effort is to unify the concepts of these languages and introduce a common basic syntax and semantics and design a new unified modeling language for model representation.

1.3 *The Modelica effort*

The work was started in the continuous time domain since there is a common mathematical framework in the form of differential-algebraic equation (DAE) systems and there are several existing modeling languages based on similar ideas. There is also significant experience of using these languages in various applications. It was thus appropriate to collect all knowledge and experience in order to design a new unified modeling language or neutral format for model representation. The short range goal is to design a modeling language based on DAE systems with some discrete event features to handle discontinuities and sampled systems. The design should allow an evolution to a multi-formalism, multi-domain, general-purpose modeling language.

The members of the Modelica design group are listed in Table 1. Hilding Elmqvist is the

Table 1   The active members of the Modelica design group.

Fabrice Boudaud, Gaz de France
Jan Broenink, Univ. of Twente, Netherlands
Dag Brück, Dynasim AB, Lund, Sweden
Hilding Elmqvist, Dynasim AB, Lund, Sweden
Thilo Ernst, GMD-FIRST, Berlin, Germany
Peter Fritzson, Linköping University, Sweden
Alexandre Jeandel, Gaz de France
Kaj Juslin, VTT, Finland
Sven Erik Mattsson, Lund University, Sweden
Bernt Nilsson, Lund University, Sweden
Martin Otter, DLR Oberpfaffenhofen, Germany
Per Sahlin, BrisData AB, Stockholm, Sweden
Hubertus Tummescheit, GMD FIRST, Berlin
Hans Vangheluwe, University of Gent, Belgium

chair man. The activity started in October 1996 as an effort within the ESPRIT project "Simulation in Europe Basic Research Working Group (SiE-WG)". Information on SiE-WG can be found in (18) and the at home page http://hobbes.rug.ac.be/SiE/. In February 1997 the Modelica design effort was made into a Technical Committee within the Federation of European Simulation Societies, EUROSIM.

2. AN INTRODUCTION TO MODELICA

In order to give an introduction to Modelica we will consider modeling of a simple electrical circuit as defined in Fig. 1. The system can be broken up into a set of connected electrical standard components. We have a voltage source, two resistors, an inductor, a capacitor and a ground point. Models of these components are typically available in model libraries and by using a graphical model editor we can define a model by drawing an object diagram as shown in Fig. 1 by positioning icons that represent the models of the components and drawing connections.

The corresponding Modelica model looks like

```
model circuit (
  Resistor:  R1 (R=10);
  VsourceAC: AC;
  Capacitor: C  (C=0.01);
  Ground:    G;
  Resistor:  R2 (R=100);
  Inductor:  L  (L=0.1);
equation
  CONNECT(AC.n, C.n);
  CONNECT(G.p, AC.n);
  CONNECT(R1.n, C.p);
  CONNECT(AC.p, R1.p);
  CONNECT(L.p, R2.n);
  CONNECT(R1.p, R2.p);
  CONNECT(C.n, L.n);
)
```

This composite model specifies the topology of the system to be modeled. It specifies the components and the connections between the components.

The statement 'Resistor: R1 (R=10);' declares a component R1 of class Resistor and sets the default value of the resistance R to 10.
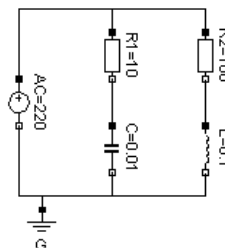

Fig. 1   A simple electrical circuit.

Connections specify interactions between components. In other modeling languages connectors are referred as cuts, ports or terminals. A connector must contain all quantities needed to describe the interaction. For electrical components we need the quantities voltage and current. The types to represent them are declared as

```
type Voltage = Real(Unit="V")
type Current = Real(Unit="A", Connection=Sum)
```

where `Real` is the name of a predefined class or type. A real variable has in addition to its value, a set of attributes such as unit of measure, initial value, minimum and maximum value. Here, the units of measure is set to be the SI units. A connector class is defined as

```
connector Pin (
  Voltage: v;
  Current: i;
)
```

A connection `CONNECT(Pin1, Pin2)`, with `Pin1` and `Pin2` of connector class `Pin`, connects the two pins such that they form one node. This implies two equations, namely `Pin1.v = Pin2.v` and `Pin1.i + Pin2.i= 0`. The first equation indicates that the voltages on both branches connected together are the same, and the second corresponds to Kirchhoff's current law saying that the current sums to zero at a node. Similar laws apply to flow rates in a piping network and to forces and torques in mechanical systems. The predefined class `Real` has an attribute `Connection` which can have the values `Equal` (default) or `Sum`. The declaration of the type `Current` defines its value of `Connection` to be `Sum`. In Modelica it is assumed that the value is positive when the current or the flow is into the component.

It is good to start by defining a set of connector classes, when developing model libraries for a new domain of application. It supports compatibility of the component models.

A common property of many electrical components is that they have two pins. It means that it is useful to define a "shell" model class

```
partial model TwoPin
  "Shell model with two electrical pins" (
  Pin: p, n;
  Voltage: v;
  Current: i;
equation
  v = p.v - n.v;
  p.i + n.i = 0;
  i = p.i;
)
```

that has two pins, p and n, a quantity, v, that defines the voltage drop across the component and a quantity, i, that defines the current into the pin p, through the component and out from the pin

n. The equations define generic relations between quantities of a simple electrical component. In order to be useful a constitutive equation must be added. The keyword `partial` indicates that this model class is incomplete. The key word is optional. It is meant as an indication to a user that he cannot use the class as it is to instantiate components. Between the name of a class and its body it is allowed to have a string. It is treat as a comment attribute. Tools may display this documentation in special ways.

To define a model for a resistor we exploit `TwoPin` and add a definition of parameter for the resistance and Ohm's law to define the behavior:

```
model Resistor "Ideal resistor" (
  extends TwoPin;
  parameter Real: R(Unit="Ohm") "Resistance";
equation
  R*i = v;
)
```

The keyword `parameter` specifies that the quantity is constant during a simulation run, but can change values between runs. A parameter is a quantity which makes it simple for a user to modify the behavior of a model.

Models for electrical capacitors and inductors are defined in similar ways

```
model Capacitor "Ideal capacitor" (
  extends TwoPin;
  parameter Real: C(Unit="F") "Capacitance";
equation
  C*der(v) = i;
)
```

```
model Inductor "Ideal inductor" (
  extends TwoPin;
  parameter Real: L(Unit="H") "Inductance";
equation
  L*der(i) = v;
)
```

where `der(v)` means the time derivative of v. A model for the voltage source can be defined as

```
model VsourceAC "Sine-wave voltage source" (
  extends TwoPin;
  parameter Real: VA = 220 "Amplitude [V]";
  parameter Real: f = 50 "Frequency [Hz]";
  parameter Real: t0 = 0 "Offset time [s]";
private
  constant Real: PI=3.141592653589793;
equation
  v = VA*sin(2*PI*f*(Time-t0));
)
```

Finally, we must not forget the ground point.

```
model Ground "Ground" (
  Pin: p;
equation
  p.v = 0;
)
```

The purpose of the ground model is twofold. First, it defines a reference value for the voltage levels. Secondly, the connections will generate one Kirchhoff's current law too many. The ground model handles this by introducing an extra current quantity `Pin.i`, which implicit by the equations will be calculated to zero.

3. MORE ADVANCED MODELING FEATURES

So far, the Modelica language has been introduced by giving an elementary example. Model classes and their instantiation form the basis for hierarchical modeling, connectors and connections corresponds to physical connections of components. At the lowest level, equations are used to describe the relation between the quantities of the model.

The expressive modeling power of Modelica is large. We will briefly summarize the more powerful constructs below.

Modeling of, for example, multi-body systems, control systems and approximations to partial differential equations is done naturally by utilizing matrix equations. Multi-dimensional matrices and the usual matrix operators and matrix functions are thus supported in Modelica. It is also possible to have arrays of components and to define regular connection patterns. A typical usage is modeling of a distillation column which consists of a set of trays connected in series.

We have so far discussed component parameters like the resistance value. Reuse of model library components is even more supported by allowing also model classes to be parametricized. An example is a controlled plant where certain PID controllers are replaced with auto tuning controllers. It is of course possible to just replace those controllers in a graphical user environment, i.e. to create a new model. The problem with this solution is that two models have to maintained. Modelica has the capability to instead just substitute the model class of certain components using a language construct at the highest hierarchical level, i.e. only one version of the rest of the model is needed.

Realistic physical models typically contains discontinuities, events and changes of structure. Examples of such phenomena are relays, switches, friction, impact, sampled data systems etc. Modelica supports such models by allowing the use of finite state machines and Petri Nets and in a way that a simulator can introduce efficient handling of such events. Special emphasis is given to synchronization and propagation of events and the possibility to find consistent restarting conditions after an event.

4. NON-CAUSAL MODELING

It is now widely recognized that graphical system input tools are an important part of simulationist's toolkit. However, graphical system input on its own is not sufficient to solve all problems. It is important to have an appropriate framework for model representation. Most of the general-purpose simulation software on the market such as SIMULINK, SystemBuild and ACSL assume that a system can be decomposed into block diagram structures with causal interactions. This means that the models are expressed as an interconnection of submodels on explicit state-space form:

$$\frac{dx}{dt} = f(x, u)$$
$$y = g(x, u)$$

where u is input and y is output. The connections of outputs to to inputs must not lead to algebraic loops. It is seldom that a natural decomposition into subsystems lead to such a model. It is often a significant effort in terms of analysis and analytical transformations to obtain a problem in this form. It requires a lot of engineering skills and manpower and it is error-prone. To illustrate this a block diagram description of the system in Fig. 1 is shown in Fig. 2. The topology of the circuit is not preserved in the block diagram. Furthermore, different types of blocks are needed for the two resistors. The block `Res2`, which represents the resistor R2 has the current as input and calculates the voltage drop as output, while the block `Res1`, which represents the resistor R1 has reversed computational causality with the voltage drop as input and the current as output. This means that we need to have two different blocks for resistors and it is the user's task to find out which to use. Furthermore, in most cases this is not sufficient, because despite how the blocks are selected there is an inherent algebraic loop. A very simple example is two resistors in series connected to a voltage source.

Modelica supports object-oriented modeling, where behavior on the lowest level may be expressed in terms of ordinary differential equations
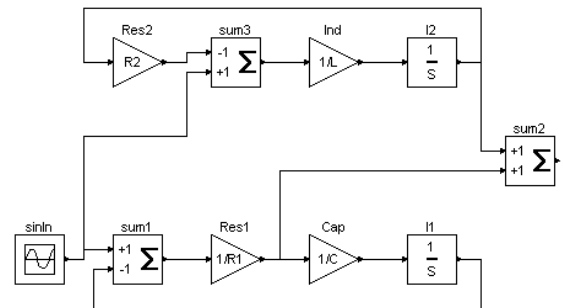

Fig. 2   A block diagram for the system in Fig. 1.

and algebraic equations, so called differential-algebraic equation (DAE) systems, which is the natural mathematical framework for continuous time models. On the other hand, the Modelica language has been carefully designed in such a way that computer algebra can be utilized to achieve as efficient code as if the model would be converted to ODE form manually.

5. CONCLUSIONS

The Modelica effort has been described and an overview of Modelica has been given. More information is available on WWW at URL: http://www.Dynasim.se/Modelica.

6. REFERENCES

[1] M. Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis ISRN LUTFD2/TFRT--1043--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, December 1994.

[2] J. Barby. "The need for a unified modeling language and VHDL-A." In *Proceedings of the 1996 IEEE International Symposium on Computer-Aided Control System Design*, pp. 258–263, Dearborn, Mi, USA, September 1996.

[3] P. Barton and C. Pantelides. "Modeling of combined discrete/continuous processes." *AIChE J.*, **40**, pp. 966–979, 1994.

[4] M. Biersack, V. Friesen, S. Jähnichen, M. Klose, and M. Simons. "Towards an architecture for simulation environments." In Vren and Birta, Eds., *Proceedings of the Summer Computer Simulation Conference (SCSC'95)*, pp. 205–212. The Society for Computer Simulation, 1995.

[5] A. P. Breunese and J. F. Broenink. "Modeling mechatronic systems using the SIDOPS+ language." In *Proceedings of ICBGM'97, 3rd International Conference on Bond Graph Modeling and Simulation*, Simulation Series, Vol.29, No.1, pp. 301–306. The Society for Computer Simulation International, January 1997.

[6] H. Elmqvist, D. Brück, and M. Otter. *Dymola — User's Manual*. Dynasim AB, Research Park Ideon, Lund, Sweden, 1996.

[7] H. Elmqvist, F. Cellier, and M. Otter. "Object-oriented modeling of hybrid systems." In *Proceedings of European Simulation Symposium, ESS'93*. The Society of Computer Simulation, October 1993.

[8] P. Fritzson, L. Viklund, D. Fritzson, and J. Herber. "High-level mathematical modeling and programming." *IEEE Software*, **12:3**, July 1995.

[9] IEEE. "Standard VHDL Analog and Mixed-Signal Extensions." Technical Report IEEE 1076.1, IEEE, March 1997.

[10] A. Jeandel, F. Boudaud, P. Ravier, and A. Buhsing. "U.L.M: Un Langage de Modélisation, a modelling language." In *Proceedings of the CESA'96 IMACS Multiconference*. IMACS, Lille, France, July 1996.

[11] A. Jeandel, P. Ravier, and A. Buhsing. "U.L.M.: Reference guide." Technical Report M DéGIMA.1205, Gaz de France, 1995.

[12] M. Kloas, V. Friesen, and M. Simons. "Smile — A simulation environment for energy systems." In Sydow, Ed., *Proceedings of the 5th International IMACS-Symposium on Systems Analysis and Simulation (SAS'95)*, vol. 18–19 of *Systems Analysis Modelling Simulation*, pp. 503–506. Gordon and Breach Publishers, 1995.

[13] S. E. Mattsson, M. Andersson, and K. J. Åström. "Object-oriented modelling and simulation." In Linkens, Ed., *CAD for Control Systems*, chapter 2, pp. 31–69. Marcel Dekker Inc, New York, 1993.

[14] M. Oh and C. Pantelides. "A modelling and simulation language for combined lumped and distributed parameter systems." *Computers and Chemical Engineering*, **20**, pp. 611–633, 1996.

[15] P. Piela. *ASCEND: An Object-Oriented Environment for Modeling and Analysis*. PhD thesis EDRC 02-09-89, Engineering Design Research Center, Carnegie Mellon Univeristy, Pittsburgh, PA, USA, 1989.

[16] P. Piela, T. Epperly, K. Westerberg, and A. Westerberg. "ASCEND: An object-oriented computer environment for modeling and analysis: the modeling language." *Computers and Chemical Engineering*, **15:1**, pp. 53–72, 1991.

[17] P. Sahlin, A. Bring, and E.F.Sowell. "The Neutral Model Format for building simulation, Version 3.02." Technical Report, Department of Building Sciences, The Royal Institute of Technology, Stockholm, Sweden, June 1996.

[18] H. L. Vangheluwe, E. J. Kerckhoffs, and G. C. Vansteenkiste. "Simulation for the Future: Progress of the ESPRIT Basic Research working group 8467." In Bruzzone and Kerckhoffs, Eds., *Proceedings of the 1996 European Simulation Symposium (Genoa)*, pp. XXIX – XXXIV. Society for Computer Simulation International (SCS), October 1996.

[19] L. Viklund and P. Fritzson. "ObjectMath – An object-oriented language and environment for symbolic and numerical processing in scientific computing." *Scientific Programming*, **4**, pp. 229–250, 1995.