MODELICA — THE NEXT GENERATION MODELING LANGUAGE AN INTERNATIONAL DESIGN EFFORT

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

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

Abstract

A new language called Modelica for physical modeling is developed in an international effort. The main objective is to make it easy to exchange models and model libraries. The design approach builds on non-casual modeling with true ordinary differential and algebraic 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 language for model representation. The paper describes the effort and gives an overview of Modelica.

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 international effort started to design a new language for physical modeling. The language is called Modelica. The main objective is to make it easy to exchange models and model libraries and to allow users to benefit from the advances in object-oriented modeling methodology. This paper presents the status of the Modelica design as of June 1997.[1]

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 are based on the same modeling methodology, input-output blocks, as in the previous standardization effort, CSSL, from 1967 [18]. There are domain oriented packages: electronic programs (SPICE, Saber), multibody systems (ADAMS, DADS, SIMPACK), chemical processes (ASPEN Plus, SpeedUp) etc. With few exceptions, all simulation packages are only strong in one domain and are not capable of modeling components in other domains reasonably. This is a major disadvantage since technical systems are becoming more and more heterogeneous with components from many engineering domains.

1.2 State-of-the art

Techniques for general-purpose physical modeling have been developed during the last decades, but did not receive much attention from the simulation 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 [2, 14], NMF [17], ObjectMath [8, 20], Omola [13, 1], SIDOPS+ [4], Smile [3, 12], U.L.M. [11, 10] and VHDL-AMS [9, 5]. 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 in order to introduce a common basic syntax and semantics and to design a new unified modeling language for model representation.

1.3 The Modelica effort

The work 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

[1]The paper in the proceedings presents the status of the Modelica design as of May 1997. To update this paper to the current status as of August 1997, the examples have been modified to adapt to some minor syntactical design changes.

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 chairman. Information on the Modelica effort is available on WWW at `http://www.Dynasim.se/Modelica/`.

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 [19] and the at home page `http://hobbes.rug.ac.be/SiE/`.

In February 1997 the Modelica design effort became 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 Figure 1. The system can be broken up into a set of connected electrical standard components. We have a voltage source, two resistors, an inductor,

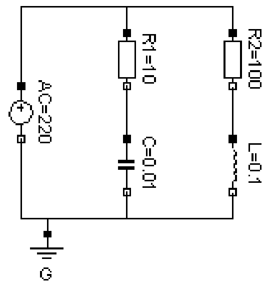

Figure 1 A simple electrical circuit.

a capacitor and a ground point. Models of these components are typically available in model libraries. Using a graphical model editor we can define a model by drawing an object diagram as shown in Figure 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);
end circuit;
```

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.

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. Their types are declared as

```
type Voltage = Real(Unit="V");
type Current = Real(Unit="A");
```

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

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
Matthias Klose, Technical Univ. of Berlin, Germany
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

measure is set to be the SI units. A connector class is defined as

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

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 a mechanical system. The sum-to-zero equations are generated when the prefix flow is used in the connector equations. In Modelica it is assumed that the value is positive when the current or the flow is into the component.

Defining a set of connector classes is a good start, when developing model libraries for a new application domain. It promotes 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;
end TwoPin;
```

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 common 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. Between the name of a class and its body it is allowed to have a string. It is treated 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;
end Resistor;
```

The keyword parameter specifies that the quantity is constant during a simulation experiment, but can change values between experiments. 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;
end Capacitor;

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

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]";
private
  constant Real PI=3.141592653589793;
equation
  v = VA*sin(2*PI*f*Time);
end VsourcdAC;
```

Finally, we must not forget the ground point.

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

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 p.i, which implicitly is defined to be zero by the equations.

3. More Advanced Modeling Features

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. The more powerful constructs are summarized below.

Modeling of, for example, multi-body systems, control systems and approximations to partial differential equations is done conveniently 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 the 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 further supported by allowing also model classes to be parametricized. An example is a controlled plant where some 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 must be maintained. Modelica has the capability to instead just substitute the model class of certain components using a language construct at the highest hierarchical level, so 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 in a way that a simulator can introduce efficient handling of such events. Special design emphasis is given to synchronization and propagation of events and the possibility to find consistent restarting conditions after an event.

4. Non-Causal Modeling

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 ACSL, SIMULINK and SystemBuild 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:

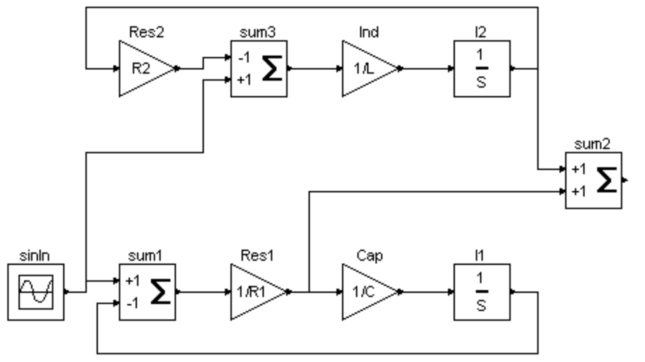

Figure 2 A block diagram for the system in Figure 1.

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

where u is input and y is output. The connections of outputs to to inputs must not lead to algebraic loops. It is rare that a natural decomposition into subsystems lead to such a model. Often a significant effort in terms of analysis and analytical transformations is needed 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 Figure 1 is shown in Figure 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 and algebraic equations, so called differential-algebraic equation (DAE) systems, which is the natural mathematical framework for continuous time models. The Modelica language has been carefully designed in such a way that computer algebra can be utilized to achieve as efficient simulation 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. The design is still evolving (May 1997). A first version of the language definition is scheduled to be available in September 1997. More information, including modeling requirements, rationale and definition of the Modelica language is available on WWW at

URL: `http://www.Dynasim.se/Modelica/`.

Acknowledgements
The authors would like to thank the other members of the Modelica Design Group for inspiring discussions and their contributions to the Modelica design.

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

[3] 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.

[4] 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.

[5] E. Christen and K. Bakalar. "VHDL 1076.1: Analog and mixed-signal extensions VHDL." In Vachoux, Ed., *Current Issues in Electronic Modeling*, vol. 10, chapter 2. Kluwer Academic Publishers, 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] J. C. Strauss (ed.). "The SCi continuous system simulation language (CSSL)." *Simulation*, **9**, pp. 281–303, 1967.

[19] 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.

[20] 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.