MODELICA — THE NEW OBJECT-ORIENTED MODELING LANGUAGE

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

Martin Otter
DLR Oberpfaffenhofen
D-82230 Wessling, Germany
E-mail: Martin.Otter@DLR.de

KEYWORDS

Modelica, modeling language, object-orientation, hierarchical systems, differential-algebraic equations

ABSTRACT

A standardized language for reuse and exchange of models is needed. An international design group has designed such a language called Modelica. Modelica is a modern language built on non-causal modeling with mathematical equations and object-oriented constructs to facilitate reuse of modeling knowledge.

INTRODUCTION

Modeling and simulation are becoming more important since engineers need to analyse increasingly complex systems often composed of subcomponents from different domains. Typical examples comes from mechatronic systems within automotive, aerospace and robotics applications. Such systems are composed of components from domains like electrical, mechanical, hydraulical, control, etc. Today's tools are generally weak in treating such multi-domain models because the general tools are block-oriented and thus demand a huge amount of manual rewriting to get the equations into explicit form.

There is too large a gap between the user's problem and the model description that the simulation program understands. Modeling should be much closer to the way an engineer builds a real system, first trying to find standard components like motors, pumps and valves from manufacturers' catalogues with appropriate specifications and interfaces. Only if there does not exist a particular subsystem, the engineer would actually construct it.

Reuse is a key issue for handling complexity. There have been several attempts to define object-oriented languages for physical modeling. However, the ability to reuse and exchange models relies on a standardized format. It was thus important to bring this expertise together and unify the concepts and notations. A language design group was thus formed in September 1996 and one year later the Modelica[1] language was available.

Modelica is intended for modeling within many application domains such as electrical circuits, multi-body systems, drive trains, hydraulics, thermodynamical systems, and chemical processes etc. It supports several formalisms: ordinary differential equations (ODE), differential-algebraic equations (DAE), bond graphs, finite state automata, and Petri nets etc. Modelica is intended to serve as a standard format so that models arising in different domains can be exchanged between tools and users.

This paper gives an introduction to Modelica by means of an example. An overview of the work on supporting libraries and the plans for further development is given. Another aim of the paper is to interest individuals to participate in developing Modelica further.

[1]Modelica^{TM} is a trade mark of the Modelica Design Group

MODELICA BASICS

As an introduction to Modelica, consider modeling of a simple motor drive system as defined in Fig. 1. The system can be broken up into a set of connected components: an electrical motor, a gearbox, a load and a control system. A Modelica model of the motor drive system is given in Fig. 2. It is a composite model which specifies the topology of the system to be modeled in terms of components and connections between the components. The statement "`Gearbox gearbox(n=100);`" declares a component `gearbox` of class `Gearbox` and sets the default value of the gear ratio, `n`, to 100.

Much of the Modelica syntax is normally hidden from the end-user. Models of standard components are typically available in model libraries. Using a graphical model editor, a model can be defined by drawing a model diagram as shown in Fig. 1, by positioning icons that represent the models of the components, drawing connections and giving parameter values in dialogue boxes. Constructs for including graphical annotations in Modelica make icons and model diagrams portable.

A component model may be a composite model to support hierarchical modeling. The object diagram of the model class `Motor` is shown in Fig. 3.

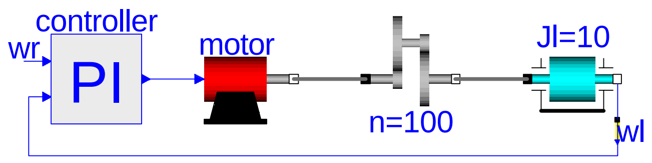

Figure 1 Schematic picture of a motor drive.

```
model MotorDrive
  Motor      motor;
  PI         controller;
  Gearbox    gearbox(n=100);
  Shaft      Jl(J=10);
  Tachometer wl;
equation
  connect(controller.out, motor.in);
  connect(motor.flange  , gearbox.a);
  connect(gearbox.b     , Jl.a);
  connect(Jl.b          , wl.a);
  connect(wl.w          , controller.in);
end MotorDrive;
```

Figure 2 A Modelica model of the system in Fig. 1.

Variables

Physical modeling deals with specifying relations between physical quantities. For the drive system, quantities such as angle and torque are of interest. Their types are declared in Modelica as

```
type Angle  = Real(quantity = "Angle",
                   unit = "rad",
                   displayUnit = "deg");
type Torque = Real(quantity = "Torque",
                   unit = "N.m");
```

where `Real` is a predefined type, which has a set of attributes such as name of quantity, unit of measure, default display unit for input and output, minimum value, maximum value and initial value.

Connectors and Connections

Connections specify interactions between components. A connector should contain all quantities needed to describe the interaction. Voltage and current are needed for electrical components. Angle and torque are needed for drive train elements.

```
connector Pin               connector Flange
  Voltage v;                  Angle r;
  flow Current i;              flow Torque t;
end Pin;                    end Flange;
```

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 declarations.

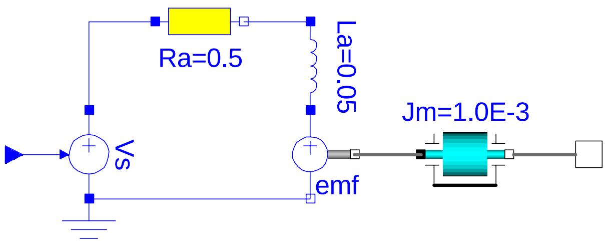

Figure 3 A motor model.

Partial models and inheritance

A very important feature in order to build reusable descriptions is to define and reuse *partial models*. A common property of many electrical components is that they have two pins. This means that it is useful to define an interface model class `TwoPin`, that has two pins, `p` and `n`, and a quantity, `v`, that defines the voltage drop across the component.

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

The equations define common relations between quantities of a simple electrical component. The keyword **partial** indicates that the model class is incomplete. To be useful, a constitutive equation must be added. To define a model for a resistor, start from `TwoPin` and add a parameter for the resistance and Ohm's law to define the behavior.

```
model Resistor "Ideal resistor"
  extends TwoPin;
  parameter Resistance R;
equation
  R*p.i = v;
end Resistor;
```

A string between the name of a class and its body is treated as a comment attribute. Tools may display this documentation in special ways. The keyword **parameter** specifies that the quantity is constant during a simulation experiment, but can change values between experiments.

For the mechanical parts, it is also useful to define a shell model with two flange connectors,

```
partial model TwoFlange
  Flange a, b;
end TwoFlange;
```

A model of a rotating inertia is given by

```
model Shaft
  extends TwoFlange;
  parameter Inertia J = 1;
  AngularVelocity w;
equation
      a.r   = b.r;
  der(a.r)  = w;
  J*der(w)  = a.t + b.t;
end Shaft;
```

where **der**(`w`) means the time derivative of `w`.

THE IDEAS BEHIND MODELICA

Among the recent research results in modeling and simulation the two concepts *object-oriented* and *non-causal modeling* have had a strong impact on the Modelica design. A new attempt at introducing interoperability and openness to the world of modeling and simulation systems is justified by the combined power of the two concepts together with proven technology from existing modeling languages such as ASCEND [Piela *et al.* (1991)], Dymola [Elmqvist *et al.* (1996)], gPROMS [Barton and Pantelides (1994)], NMF [Sahlin *et al.* (1996)], ObjectMath [Fritzson *et al.* (1995)], Omola [Mattsson *et al.* (1993)], SIDOPS+ [Breunese and Broenink (1997), Smile [Kloas *et al.* (1995)], U.L.M. [Jeandel *et al.* (1996)] and VHDL-AMS [IEEE (1997)].

Non-Causal Modeling

In Modelica it is possible to write balance and other equations in their natural forms as a system of differential-algebraic equations (DAE). Modelica 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. For example, define a gearbox model as

```
model Gearbox "Ideal gearbox without inertia"
  extends TwoFlange;
  parameter Real n;
equation
    a.r = n*b.r;
  n*a.t = b.t;
end Gearbox;
```

without bothering about what are inputs from a computational point of view and use it as a component model, when modeling the drive system in Fig. 1.

Note, that this use actually leads to a non-trivial simulation problem. The ideal gearbox is rigidly connected to a rotating inertia on each side. It means the model includes two rigidly connected inertias, since there is no flexibility in the ideal gearbox. The angular position as well as the velocity of the two inertias should be equal. All of these four differentiated variables cannot be state variables with their own independent initial values.

A DAE problem, which includes constraints between differentiated variables is sometimes called a “high index DAE”. When converting it to ODE form, it is necessary to differentiate some equations and the set of state variables can be selected smaller than the set of differentiated variables. There is an efficient algorithm by Pantelides (1988) for the determination of what equations to differentiate. In the drive example, the position constraint needs to be differentiated twice to calculate the reaction torque in the coupling, and it is sufficient to select the angle and velocity of either inertia as state variables. The constraint leads to a linear system of simultaneous equations involving angular accelerations and torques. A symbolic solution will contain a determinant of the form “$J_\ell + J_m n^2$”. The tool thus automatically deduces how inertia is transformed through a gearbox.

ADVANCED MODELING FEATURES

The modeling power of Modelica is large. Modeling of, for example, multi-body systems and control systems is done conveniently by utilizing *matrix equations*. It is also possible to have arrays of components and to define regular connection patterns. Reuse of model library components is further supported by allowing *model class parameters*.

Modelica supports *hybrid modeling*. Realistic physical models often contain discontinuities, discrete events or changes of structure. Examples are relays, switches, friction, impact, sampled data systems etc. *Algorithms and functions* are supported in Modelica for modeling parts of a system in procedural programming style.

STANDARD LIBRARIES

In order that Modelica is useful for *model exchange*, it is important that libraries of the most commonly used components are available, ready to use, and sharable between applications. For this reason, an extensive *Modelica base library* is under development which will become an intrinsic part of Modelica. It includes mathematical functions (*sin*, *ln*, etc.), type definitions (e.g., `Angle, Voltage`), interface definitions (e.g., `Pin, Flange`) and component libraries for various domains.

Predefined quantity types and connectors are useful for standardization of the interfaces between components and achieve model compatibility without having to resort to explicit coordination of modeling activities.

Component libraries are mainly derived from already existing model libraries from various object-oriented modeling systems. They are realized by specialists in the respective area, taking advantage of the new features of Modelica not available in the original modeling system. Libraries of the following areas are under development: input/output blocks, electric and electronic components (SPICE3 elements), electric power systems, drive trains and gear boxes, 3D-mechanical systems (multi-body systems), hydraulic systems, 1D thermo-fluid flow (based on the finite volume method), aircraft flight system dynamics components, bond graphs, finite state machines and Petri nets.

DIRECTIONS OF FUTURE DEVELOPMENT

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

There is a need to consider extensions of Modelica for handling of partial differential equations, discrete event models, etc. Some of these areas are discussed below.

Partial Differential Equations

The underlying laws of a model of a technical system are often of the form of partial differential equations (PDE). The DAE form is then an approximation where certain quantities are considered independent of the spatial coordinates within a certain object. For certain phenomena like heat conduction, convection, laminar and turbulent flows, and vibrations in flexible mechanical structures, more accurate PDE models might be needed to capture the detailed behavior.

A PDE model is defined by (1) a partial differential equation, (2) the domain of validity of the PDE and (3) boundary conditions for spatial borders and initial conditions. The domain of validity is typically a geometrical domain in 1–3 dimensions. A variety of geometry formats are used by different CAD software. To support exchange it is necessary to have an application independent format for representing geometry in Modelica.

Discrete Event Models

Further model approximations are done when quantities are considered to be constant over intervals of time, i.e., when dealing with discrete event models. Modelica already have features for handling discrete variables and describe how they are changed. However, many issues of standard discrete event packages have not been considered yet, such as: function library for various random distributions, queue handling, how to gather statistical information, processes and their interaction, animation features, etc. It would be possible to extend Modelica with such features. However, there are also many different formalisms for discrete event modeling, such as: process oriented, activity oriented, Petri nets, Grafcet, DEVS, State charts and VHDL (discrete circuit modeling). There are also many discrete event programs available. In certain cases, modeling is done in a standard programming language like C++ or Java with the use of a library of standard functions. Probably, Modelica should be both extended with basic discrete event features, as well as having appropriate interfaces defined for coupling to external discrete event packages.

Simulator Environment

Most issues about simulating a Modelica model is an internal matter for the simulator. However, for hardware-in-the-loop simulation one typically needs to specify which fixed step-size algorithm and what step-size to use. There is also a need to specify the coupling to external input and output hardware. To make such information portable, a notation in Modelica is needed.

In cases when a Modelica model, for example, needs to access medium properties from a data base or be simulated simultaneously with models in other programs like a discrete event package, a finite element package or in a network performing distributed interactive simulation (DIS), certain information needs to be exchanged. Such specification needs to be portable, i.e., language notations are needed in Modelica. A possibility would be to use the High Level Architecture, HLA.

There are also cases when the model description needs to be augmented in order to help the tool achieve fast enough simulation. An example is to partition the model and use multi-rate integration, i.e., to integrate parts of a model with fast dynamics with shorter step-size. Another example is the ability to help the tool partition the model for execution on different processors in a parallel architecture. For models with changing topology, there might also be a need to specify which configurations are useful. A model with 10 switches can be in 1024 different modes. If the modeler knows that maybe only 20 modes are actually used, more efficient code can be generated than would be possible for the general case of 1024 modes.

Experiment Specification

When using a mathematical model for simulation or optimization, the model itself is only a part of the problem specification. Parameter values, initial values, start time, stop time or stop condition and how to treat the result of the simulations are also needed. On a more lower level it may be of interest to specify solvers and their parameters.

The separation of model, integration algorithm and experiment is common since a long time. However, what constitute an experiment is not always obvious. Does a change of a model parameter belong to an experiment or to a model? Sometimes the environment of the system under study is described by some input signals connected to the model. In certain cases, the "environment" is described as a dynamic model, i.e., the full power of Modelica is needed. Similarly, describing an optimization criterion typically contain integration that can be converted to a differential equation. Even hybrid features might be needed, for example, to define a maximum of a signal accurately would involve checking for the sign shift of the derivative of the signal.

Issues like handling of parameter sets, functions defined by tables, coupling of inputs, how initialization should be done, definition of an optimization problem, etc. needs to be portable between different tools. This means that Modelica should be extended in these directions.

User Environment

Modelica already has provisions to describe the graphical layout of icons and connection topology by means of annotations. So far, only static pictures have been considered. When using models for operator training, typically a live process layout is used to show the status of the process by means of updated numeric text, changing water level of a tank, etc. There is also a need to input new parameter values in specially designed forms. The annotation attributes could be extended to handle this.

Typically, an interactive user interface for modeling and simulation needs extensive capabilities for general matrix calculations and control design algorithms. It should, of course, be possible to use a Modelica tools with close connections to available packages like Matlab, Xmath, Matematica, etc. However, for many users it would be beneficial to use the Modelica syntax, the strong typing property and matrix expressions in an interactive fashion. Modelica functions could then be used both within a model and be called interactively.

Predefined external functions would specify the interfaces of simulators, optimizers, display tools etc. These interfaces should define parameters and operations which could be done. Modelica already supports a powerful function concept. However, it may be useful to introduce language constructs which supports member functions in the ordinary object-oriented meaning.

It should be remembered that high quality tools should have good interactive graphical user interfaces with menu selection, dialogue boxes, ability to record menu commands in script form, etc. A typical user who wants to solve a problem should not need to write or read this textual representation.

ORGANIZATION OF MODELICA DESIGN

The Modelica design effort started as an action in the ESPRIT project "Simulation in Europe Basic Research Working Group (SiE-WG)" and is since February 1997 the Technical Committee 1 within Eurosim. The Modelica Design Group has had 11 meetings to work out the Modelica fundamentals. The plan is to make the Modelica design effort a Technical Chapter within SCS.

The Modelica Design Group includes simulation tool builders, users from different application domains, and computer scientists. The present members (May 1998) are *Manuel Alfonseca*, Universidad Autonoma de Madrid, Spain, *Bernhard Bachmann*, ABB Corporate Research, Baden-Dättwil, Switzerland, *Fabrice Boudaud* and *Alexandre Jeandel*, Gaz de France, *Jan Broenink*, University of Twente, The Netherlands, *Dag Brück* and *Hilding Elmqvist* (chairman), Dynasim AB, Lund, Sweden, *Thilo Ernst*, GMD-FIRST, Berlin, Germany, *Rüdiger Franke*, Technical University of Ilmenau, Germany, *Peter Fritzson*, Linköping University, Sweden, *Kaj Juslin*, VTT, Finland, *Matthias Klose*, Technical University of Berlin, Germany, *Sven Erik Mattsson*, Lund University, Sweden, *Pieter Mosterman* and *Martin Otter*, DLR Oberpfaffenhofen, Germany, *Per Sahlin*, BrisData AB, Stockholm, Sweden, *André Schneider* and *Peter Schwarz*, Fraunhofer Institute for Integrated Circuits, Dresden, Germany, *Hubertus Tummescheit*, GMD FIRST, Berlin, Germany, *Hans Vangheluwe*, University of Gent, Belgium.

CONCLUSIONS

The Modelica effort has been described and an overview of Modelica has been given. Version 1.0 of Modelica was finished in September 1997. There is ongoing work to develop model libraries and tools. For more information, including rationale and definition of Modelica and future developments, see www at Modelica (1998).

The paper has also outlined some areas that needs to be developed. We invite individuals with expertize to participate in extending Modelica. If you are interested, please, contact any of the authors.

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

BARTON, P. and C. PANTELIDES (1994): “Modeling of combined discrete/continuous processes.” *AIChE J.*, **40**, pp. 966–979.

BREUNESE, A. P. and J. F. BROENINK (1997): “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.

ELMQVIST, H., D. BRÜCK, and M. OTTER (1996): *Dymola — User's Manual.* Dynasim AB, Research Park Ideon, Lund, Sweden.

FRITZSON, P., L. VIKLUND, D. FRITZSON, and J. HERBER (1995): “High-level mathematical modeling and programming.” *IEEE Software*, **12:3**.

IEEE (1997): “Standard VHDL Analog and Mixed-Signal Extensions.” Technical Report IEEE 1076.1. IEEE.

JEANDEL, A., F. BOUDAUD, P. RAVIER, and A. BUHSING (1996): “U.L.M: Un Langage de Modélisation, a modelling language.” In *Proceedings of the CESA'96 IMACS Multiconference.* IMACS, Lille, France.

KLOAS, M., V. FRIESEN, and M. SIMONS (1995): “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.

MATTSSON, S. E., M. ANDERSSON, and K. J. ÅSTRÖM (1993): “Object-oriented modelling and simulation.” In LINKENS, Ed., *CAD for Control Systems*, chapter 2, pp. 31–69. Marcel Dekker Inc, New York.

MODELICA (1998): *A unified object-oriented language for physical systems modeling.* Modelica homepage: http://www.Dynasim.se/Modelica/.

PANTELIDES, C. (1988): “The consistent initialization of differential-algebraic systems.” *SIAM Journal of Scientific and Statistical Computing*, **9**, pp. 213–231.

PIELA, P., T. EPPERLY, K. WESTERBERG, and A. WESTERBERG (1991): “ASCEND: An object-oriented computer environment for modeling and analysis: the modeling language.” *Computers and Chemical Engineering*, **15:1**, pp. 53–72.

SAHLIN, P., A. BRING, and E.F.SOWELL (1996): “The Neutral Model Format for building simulation, Version 3.02.” Technical Report. Department of Building Sciences, The Royal Institute of Technology, Stockholm, Sweden.