Modelica — A Language for Physical System Modeling, Visualization and Interaction

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

Sven Erik Mattsson
Dynasim AB
Research Park Ideon
SE-223 70 Lund, Sweden
E-mail: SvenErik@Dynasim.se

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

Abstract

Modelica is an object-oriented language for modeling of large, complex and heterogeneous physical systems. It is suited for multi-domain modeling, for example for modeling of mechatronics including cars, aircrafts and industrial robots which typically consist of mechanical, electrical and hydraulic subsystems as well as control systems. General equations are used for modeling of the physical phenomena. No particular variable needs to be solved for manually. A Modelica tool will have enough information to do that automatically. The language has been designed to allow tools to generate efficient code automatically. The modeling effort is thus reduced considerably since model components can be reused and tedious and error-prone manual manipulations are not needed. The principles of object-oriented modeling and the details of the Modelica language as well as several examples are presented.

1. Introduction

Modeling and simulation are becoming more important since engineers need to analyse increasingly complex systems composed of components from different domains. Examples are mechatronic systems within automotive, aerospace and robotics applications. Such systems are composed of components from domains like electrical, mechanical, hydraulical, control, etc. Current tools are generally weak in treating 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. The *domain-specific* tools, such as circuit simulators or multibody programs, cannot handle components of other domains in a reasonable way.

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.

In Modelica equations are used for modeling of the physical phenomena. No particular variable needs to be solved for manually. A Modelica tool will have enough information to decide that automatically. This is an important property of Modelica to enable handling of large models having more than 100 000 equations. Modelica supports several formalisms: ordinary differential equations (ODE), differential-algebraic equations (DAE), bond graphs, finite state automata, Petri nets etc.

The language has been designed to allow tools to generate very efficient code. Modelica models are used, for example, in hardware-in-the-loop simulation of automatic gearboxes, which have variable structure models. Such models have so far usually been treated by hand, modeling each mode of operation separately. In Modelica, component models are used for shafts, clutches, brakes, gear wheels etc. and the tool can find the different modes of operation automatically. The modeling effort is thus reduced considerably since model components can be reused and tedious and error-prone manual manipulations are not needed.

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 to unify concepts and notations. A design group was formed in September 1996 and one year later, the first version of the Modelica[1] language was available (www.Modelica.org). Modelica is intended to serve as a standard format so that models arising in different domains can be exchanged between tools and users. It has been designed by a group of more than 15 experts with previous know-how of modeling languages and differential-algebraic equation models.

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

After 15 three-day meetings, during a 2-year period, version 1.1 of the language specification was finished in December 1998. Tools and model libraries are now available. The fact that the language is not tied to one particular software vendor is very important since a stable format for storing model knowledge and allowing reuse is necessary in order to handle the heterogeneous and complex models in the future.

2. Composition Diagrams

Modelica supports both high level modeling by composition and detailed library component modeling by equations. High level modeling by composition diagrams will first be discussed by giving some examples from different domains. Models of standard components are typically available in model libraries. Using a graphical model editor, a model can be defined by drawing a composition diagram as shown in Fig. 1–5, 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 composition diagrams portable.

Fig. 1 shows part of a larger multi-domain model. It consists of the composition diagram of one axis of the industrial robot Manutec r3. On the left side of the

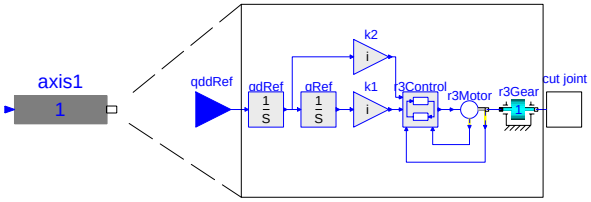

Figure 1 Composition diagram of one axis of the Manutec r3 robot.

figure the composition diagram of the overall *axis* is shown. It contains the desired reference acceleration of the axis as an input signal on the left connector, and a mechanical flange to drive a shaft on the right connector. The right side of the figure shows the decomposition of the axis: The reference acceleration is integrated twice to derive a reference velocity and a reference position. The reference values are fed into a controller `r3Control`. The output of the controller (the connector at the right side of the controller) is the reference current of the electric motor, `r3Motor`, that drives the gear box, `r3Gear`. The driven part of the gear box (connector at the right side) is a mechanical flange to which the axis of a shaft or of a robot joint can be connected.

In Fig. 2 the details of the axis *controller* `r3Control`

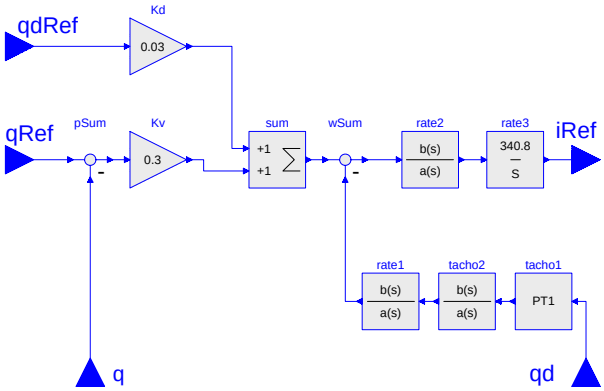

Figure 2 A controller model for the Manutec r3 robot.

are shown as a block diagram including transfer functions. Typical for such axis controllers, it has two cascaded parts consisting of a velocity and of a position controller. The output of the controller is the desired reference current of the electric motor (the current of the motor is approximately proportional to the produced motor torque which is the quantity to be "really" controlled).

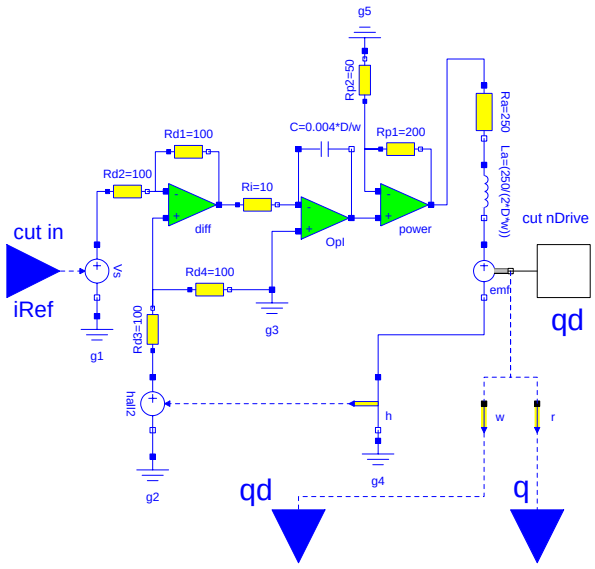

Figure 3 A motor model for the Manutec r3 robot.

In Fig. 3 the model `r3Motor` of the electric motor is shown. It consists of the current controller realized by operational amplifiers and the DC motor (`Ra`, `La`, `emf`). The reference current is the input signal to the motor (the connector at the left side) and drives a controlled voltage source. The DC motor produces a torque which drives a mechanical flange (the connector at the right side).

The composition diagram of the gearbox `r3Gear` of the drive train is shown in Fig. 4. The gearbox is modeled

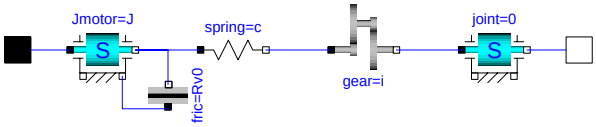

Figure 4 A gearbox model for the Manutec r3 robot.

by the motor inertia, a (rotational) spring to model the gear elasticity, an ideal gearbox representing the gear ratio and a load inertia to model the rotational inertia of all parts at the driven side of the gear. Component `fric` connected between the motor shaft and the shaft bearings models the Coulomb friction of the bearings.

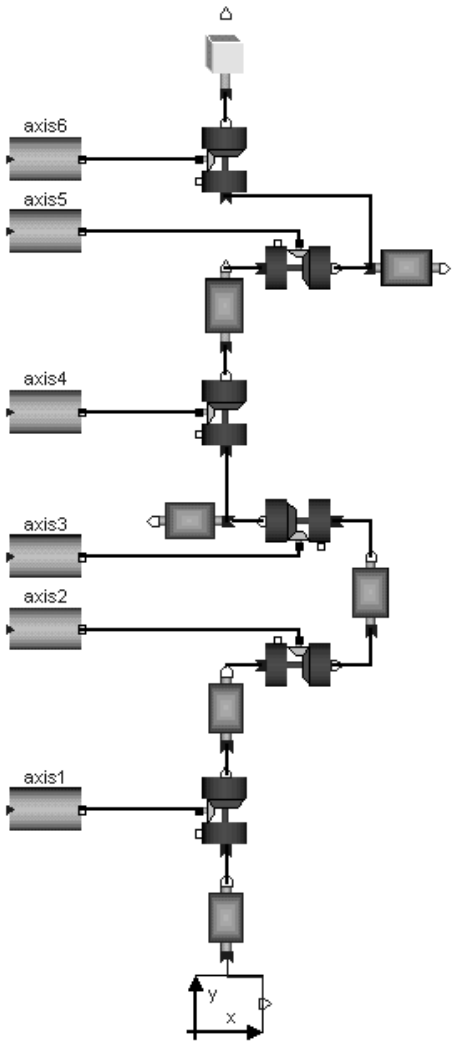

Figure 5 Overall model of the Manutec r3 robot.

Finally, Fig. 5 contains the composition diagram of the complete robot. On the right side of the figure, the mechanical part of the robot is given in form of a multibody system. It consists of six revolute joints, six bodies and the load. A body component describes the mass and inertia effects of the body and defines the visual appearance for an animation program. The joints of the robot are driven by the axes on the left side of the figure (`axis1,...,axis6`) which are instances of the already explained axis component (Fig. 1–4).

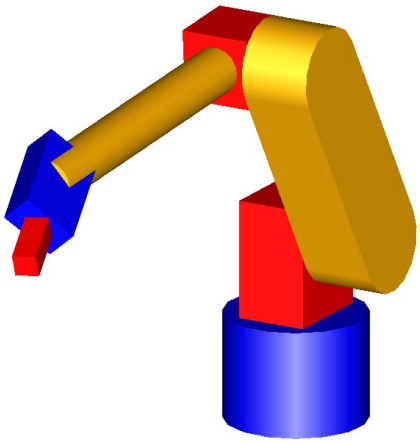

Figure 6 3D-view of a robot.

Simulation results can be animated if visual properties have been given for the bodies. An example of such 3D views is given in Fig. 6.

For 3D mechanical models, the 2D composition diagram of Fig. 5 does not show the positions and directions correctly. In such a case it might be better to use a CAD tool for definition of the mechanism and convert it to Modelica. A tool for converting SolidWork models to Modelica has already been developed (Engelson *et al.* (1999)).

3. Modelica Details

To describe how the details of a component are modeled, consider a simple motor drive system as defined in Fig. 7. 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. 8 (excluding graphical annotations).

It is a composite model which specifies the topology of the system to be modeled in terms of components

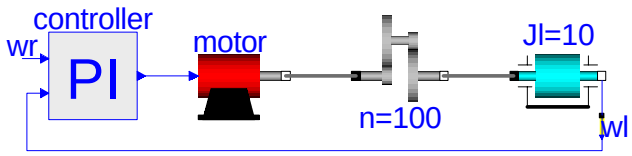

Figure 7 Schematic picture of a motor drive.

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

Figure 8 A Modelica model of the system in Fig. 7.

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.

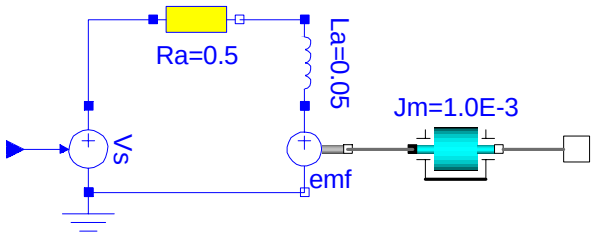

Figure 9 A motor model.

A component model may be a composite model to support hierarchical modeling. The object diagram of the model class `Motor` is shown in Fig. 9.

The meaning of connections will be discussed next as well as the description of behavior on the lowest level using real equations.

3.1 Variables

Physical modeling deals with the specification of 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. The *Modelica base library*, which is an intrinsic part of Modelica includes these kinds of type definitions.

3.2 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
  Voltage v;
  flow Current i;
end Pin;
```

```
connector Flange
  Angle r;
  flow Torque t;
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. The Modelica base library includes also connector definitions.

3.3 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`.

4. Non-Causal Modeling

In order to allow reuse of component models, the equations should be stated in a neutral form without consideration of computational order, i.e., non-causal modeling.

4.1 Background

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 (Åström *et al.* (1998)). This means that the models are expressed as an interconnection of submodels on explicit state-space form,

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

where $\mathbf{u}$ is input, $\mathbf{y}$ is output and $\mathbf{x}$ is the state. It is rare that a natural decomposition into subsystems leads 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 the difficulties, a Simulink model for the simple motor drive in Fig. 7 is shown in Fig. 10–11. The structure of the block diagram does not reflect the topology of the physical system. It is easy to recognize the controller in the Simulink model in Fig. 10, but the

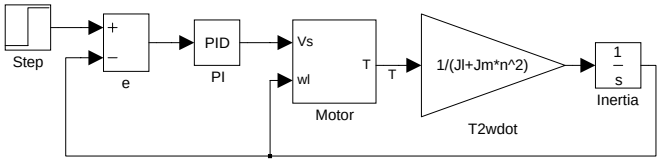

Figure 10 A Simulink model for the motor drive in Fig. 7.

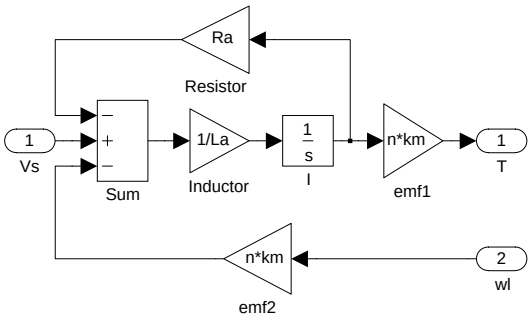

Figure 11 A Simulink model for the motor in Fig. 9.

gearbox and the inertias of the motor and the load are no longer visible. They appear combined into a gain coefficient $1/(J_l + J_m n^2)$.

There is a fundamental limitation of block diagram modeling. The blocks have a unidirectional data flow from inputs to outputs. This is the reason why an object like a gearbox in the simple motor drive cannot be dealt with directly. It is also the reason why motor and load inertia appear in the mixed expression in the Simulink model. If it is attempted to simulate the basic equations directly there will be a loop which only contains algebraic equations. Several manual steps including differentiation are required to transform the equations to the form required by Simulink. The need for manual transformations imply that it is cumbersome to build physics based model libraries in the block diagram languages. A general solution to this problem requires a paradigm shift.

4.2 Differential-algebraic equations

In Modelica it is possible to write balance and other equations in their natural form as a system of differential-algebraic equations, DAE,

$$0 = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, \mathbf{u})$$

where $\mathbf{x}$ is the vector of unknowns that appear differentiated in the equation and $\mathbf{y}$ is the vector of unknowns that do not appear differentiated.

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. 7.

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 variables appearing differentiated 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 and an algorithm for selection of state variables by Mattsson and Söderlind (1993).

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.

5. Advanced Modeling Features

The modeling power of Modelica is great. Some of the more powerful constructs are summarized below.

5.1 Vectors, matrices and arrays

Modeling of, for example, multi-body systems and control systems is done conveniently with *matrix equations*. Multi-dimensional arrays and the usual matrix operators and matrix functions are thus supported in Modelica.

The modeling of continuous time transfer function is given below as an example. It uses a restricted model called block having inputs and outputs with given causality. The polynomial coefficients in $a_0 + a_1 s + \ldots + a_n s^n$ are give as a vector $\{a_0, a_1, \ldots, a_n\}$.

```
partial block SISO
  "Single Input/Single Output block"
  input  Real u "input";
  output Real y "output";
end SISO;

block TransferFunction
  extends SISO;
  parameter Real a[:]={1, 1} "Denominator";
  parameter Real b[:]={1} "Numerator";
protected
  constant Integer na=size(a, 1);
  constant Integer nb(max=na) = size(b, 1);
  constant Integer n=na-1 "System order";
  Real b0[n] = cat(1, b, zeros(na - nb))
    "Zero expanded b vector.";
  Real x[n] "State vector";
equation
  // Controllable canonical form
  der(x[2:n]) = x[1:n-1];
  a[na]*der(x[1]) + a[1:n]*x = u;
  y = (b0[1:n] - b0[na]/a[na]*a[1:n])*x +
       b0[na]/a[na]*u;
end TransferFunction;
```

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. The use of component arrays for spatial discretization when modeling heat exchangers is illustrated in Mattsson *et al.* (1998).

5.2 Class parameters

Component parameters such as resistance values have been discussed. Reuse of model library components is further supported by allowing *model class parameters*.

As an example assume that we would like to replace the PI controller in Fig. 7 by an auto tuning controller. It is of course possible to just replace the controller 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 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. Based on the model `MotorDrive` in Fig. 8 a model `MotorDrive2` with redeclared controller is described as

```
model MotorDrive2 = MotorDrive
  (redeclare AutoTuningPI controller);
```

This is a strong modification of the motor drive model and there is the issue of possible invalidation of the model. The keyword **`redeclare`** clearly marks such

modifications. Furthermore, the new component must be a *subtype* of PI. i.e., have compatible connectors and parameters. The type system of Modelica is greatly influenced by type theory, Abadi and Cardelli (1996), in particular the notion of subtyping (the structural relationship that determines type compatibility) which is different from subclassing (the mechanism for inheritance). The main benefit is added flexibility in the composition of types, while still maintaining a rigorous type system. Inheritance is not used for classification and type checking in Modelica.

The public components of a class are typically its connectors and parameters. A model of a PI controller has connectors for the reference signal, measured value and control output and parameters such as gain and integral time. So it is natural to require that also an autotuning controller has those components.

In many real applications there are many PI controllers. This makes it clumsy to use the approach described above to change controllers, because we need to know the names of all controllers. To avoid this problem and prepare for replacement of a set of models, one can define a replaceable class, `ControllerModel` in the drive model:

```
partial block SISOController
  input  Real ref;
  input  Real inp;
  output Real out;
end SiSOController;

model MotorDrive3
  replaceable block ControllerModel =
    SISOController;
protected
  ControllerModel controller;
 // then same as MotorDrive.
end MotorDrive3;
```

where the replaceable model `ControllerModel` is declared to be of type `SISOController`, which means that it will be enforced that the actual class will have the inputs `ref` and `inp` and the output `out`, but it may be parameterized in any way. Setting `ControllerModel` to for example `PID` is done as

```
model PIDDrive = MotorDrive3
  (redeclare block ControllerModel = PID);
```

The use of model class parameters to support machine-medium decomposition is illustrated in Mattsson *et al.* (1998), Ernst *et al.* (1997) and Tummescheit and Eborn (1998).

5.3 Hybrid modeling

Realistic physical models often contain discontinuities, discrete events or changes of structure. Examples are relays, switches, friction, impact, sampled data systems etc. Modelica has introduced special language constructs allowing a simulator to introduce efficient handling of such events. Special design emphasis was given to synchronization and propagation of events and the possibility to find consistent restarting conditions after an event. Hybrid modeling is further discussed in another CACSD'99 paper, Otter *et al.* (1999). Modelica supports the development of efficient model libraries for finite state machines and Petri nets, see Mosterman *et al.* (1998). Modeling of automatic gearboxes in Modelica for the purpose of real-time simulation is described in Mattsson *et al.* (1998). Such models are non-trivial because of the varying structure during gear shift utilizing clutches, free wheels and brakes.

5.4 Algorithms and functions

Algorithms and functions are supported in Modelica for modeling parts of a system in procedural programming style. Modelica functions have a syntax similar to other Modelica classes and matrix expressions can be used. Assignment statements, if statements and loops are available in the usual way. A function for polynomial multiplication is given as an example. It takes two coefficient vectors as inputs and returns the coefficient vector for the product.

```
function polynomialMultiply
  input  Real a[:], b[:];
  output Real c[:] =
     zeros(size(a,1) + size(b, 1) - 1);
algorithm
  for i in 1:size(a, 1) loop
    for j in 1:size(b, 1) loop
      c[i+j-1] := c[i+j-1] + a[i]*b[j];
    end for;
  end for;
end polynomialMultiply;
```

6. 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 in 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, aircraft flight system dynamics components, bond graphs, finite state machines and Petri nets.

7. Future Development

The Modelica effort has so far been concentrated on physical modeling with differential-algebraic equation systems with some discrete event features to handle discontinuities and sampled systems. There is a need to consider extensions of Modelica for handling of partial differential equations, more advanced discrete event models, user interaction, etc.

7.1 User Interaction

When using a mathematical model for simulation or optimization, the model itself is only a part of the problem specification which also needs to include parameter values, initial values, start time, stop time or stop condition. On a lower level it may be of interest to specify solvers and their parameters.

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 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.

Much design effort has been devoted to describing interfaces to external function written in other languages like C, C++ and FORTRAN. The Modelica design group is now working on extending the applicability of the language for experimentation and design, i.e., as a base for the design engineer's environment. There will be a standardized view of the interaction with a model to handle parameter sets, result trajectories etc. and a standardized simulator API to allow automated parameter studies etc. Interfaces to control numerics subroutine packages like Slicot (Benner *et al.* (1998)) for control analysis/synthesis and multi-objective design optimization packages like MOPS (Joos (1999)) will be convenient due to the external function interface of Modelica.

The following simple example illustrates the use of Modelica matrix expressions, for-loops and predefined functions to operate on the model.

```
openModel("controllerTest.mo");
omega = 1;    // Declare omega.
k = 1;        // Declare gain.
for D in {0.1, 0.2, 0.4, 0.7} loop
  // Parameter sweep over damping
  // coefficient.
  tr.a = {1, 2*D*omega, omega**2};
  tr.b = {k*omega**2};
  simulateModel("controllerTest", 0, 10);
  plot(u, y);
end for;
```

7.2 Visualization

The object-oriented approach of modeling also allows that other aspects than the dynamic properties are described within the same class.

Modelica already has provisions to describe the graphical layout of icons and the 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. It is also natural to describe the graphical user interface of for example a controller in an object-oriented fashion. Work is thus going on to define how visualization of data shall be described, i.e. the use of numeric presentation, bar graphs, curves, etc. Additionally there will be a set of user input primitives for interactively changing parameters etc. The annotation attributes could be extended to handle such cases.

Incorporation of 3D graphical objects are considered to allow animation of mechanical systems, Engelson *et al.* (1999).

8. Organization of the Modelica Design

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.

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 tech-

nology from existing modeling languages such as ASCEND [Piela *et al.* (1991)], Dymola [Elmqvist (1978); 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)].

The Modelica design effort started in September 1996 within an action of the ESPRIT project "Simulation in Europe Basic Research Working Group (SiE-WG)" and became in February 1997 the Technical Committee 1 within Eurosim. It is since July 1998 a Technical Chapter within the Society for Computer Simulation International (www.SCS.org). The Modelica Design Group has had 17 meetings to work out the Modelica details.

The Modelica Design Group includes simulation tool builders, users from different application domains, and computer scientists. The present and past members (May 1999) are *Manuel Alfonseca*, Universidad Autonoma de Madrid, Spain, *Bernhard Bachmann*, ABB Network Partner Ltd., Baden-Dättwil, Switzerland, *Fabrice Boudaud*, *Alexandre Jeandel* and *Nathalie Loubere*, Gaz de France, *Jan Broenink*, University of Twente, The Netherlands, *Dag Brück*, *Hilding Elmqvist* (chairman), *Sven Erik Mattsson* and *Hans Olsson*, Dynasim AB, Lund, Sweden, *Thilo Ernst*, GMD-FIRST, Berlin, Germany, *Jorge Ferreira*, Universidade de Aveiro, Portugal, *Rüdiger Franke*, ABB Corporate Research Center, Heidelberg, Germany, *Peter Fritzson*, *David Kågedal*, and *Henrik Nilsson*, Linköping University, Sweden, *Pavel Grozman* and *Per Sahlin*, BrisData AB, Stockholm, Sweden, *Kaj Juslin*, VTT, Finland, *Matthias Klose*, Technical University of Berlin, Germany, *Pieter Mosterman* and *Martin Otter*, DLR Oberpfaffenhofen, Germany, *André Schneider* and *Peter Schwarz*, Fraunhofer Institute for Integrated Circuits, Dresden, Germany, *Hubertus Tummescheit*, Lund University, Sweden, *Hans Vangheluwe*, University of Gent, Belgium.

9. Conclusions

The Modelica effort has been described and an overview of Modelica has been given. Version 1.1 of Modelica was finished in December 1998. Tools and model libraries are now available. For more information, including rationale and definition of Modelica, upcoming meetings, future developments and available tools, see http://www.Modelica.org.

Model classes and their instantiation form the basis of hierarchical modeling. Connectors and connections correspond to physical connections of components. Inheritance supports easy adaptation of components. These concepts can be successfully employed to support hierarchical structuring, reuse and evolution of large and complex models independent from the application domain and specialized graphical formalisms.

The benefits of non-causal modeling with DAE's has been clearly demonstrated and compared to traditional block diagram modeling. It has also been pointed out that tools can incorporate computer algebra methods to translate the high-level Modelica descriptions to efficient simulation code.

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.

10. References

ABADI, M. and L. CARDELLI (1996): *A Theory of Objects*. Springer-Verlag.

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

BENNER, P., V. MEHRMANN, V. SIMA, S. VAN HUFFEL, and A. VARGA (1998): "SLICOT – A subroutine library in systems and control theory." In DATTA, Ed., *Applied and Computational Control, Signals and Circuits*, vol. 1. Birkhäuser.

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. (1978): *A Structured Model Language for Large Continuous Systems*. PhD thesis TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

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

ENGELSON, V., H. LARSSON, and P. FRITZON (1999): "Design, simulation and visualization environment for object-oriented mechanical and mult-domain models in Modelica." In *Proceedings of the IEEE International Conference on Information Visualisation*. IEEE Computer Society, London, UK.

ERNST, T., M. KLOSE, and H. TUMMESCHEIT (1997): "Modelica and Smile — A case study applying object-oriented concepts to multi-facet modeling." In *Proceedings of the 1997 European Simulation Symposium (ESS'97)*. The Society for Computer Simulation, Passau, Germany.

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.

JOOS, H.-D. (1999): "A methodology for multi-objective design assessment and flight control synthesis tuning." *Aerospace Science and Technology*, **3**.

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.

MATTSSON, S. E., H. ELMQVIST, and M. OTTER (1998): "Physical system modeling with Modelica." *Control Engineering Practice*, **6**, pp. 501–510.

MATTSSON, S. E. and G. SÖDERLIND (1993): "Index reduction in differential-algebraic equations using dummy derivatives." *SIAM Journal of Scientific and Statistical Computing*, **14:3**, pp. 677–692.

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

MOSTERMAN, P. J., M. OTTER, and H. ELMQVIST (1998): "Modeling Petri nets as local copnstraint equations for hybrid systems using Modelica." In *Proceedings of the 1998 Summer Simulation Conference*, pp. 314–319. Society for Computer Simulation International, Reno, Nevada, USA.

OTTER, M., H. ELMQVIST, and S. E. MATTSSON (1999): "Hybrid modeling in Modelica based on the synchronous data flow principle." In *Proceedings of the 1999 IEEE Symposium on Computer-Aided Control System Design, CACSD'99*. IEEE Control Systems Society, Hawaii, USA.

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.

ÅSTRÖM, K. J., H. ELMQVIST, and S. E. MATTSSON (1998): "Evolution of continuous-time modeling and simulation." In ZOBEL AND MOELLER, Eds., *Proceedings of the 12th European Simulation Multiconference, ESM'98*, pp. 9–18. Society for Computer Simulation International, Manchester, UK.

TUMMESCHEIT, H. and J. EBORN (1998): "Design of a thermohydraulic model library in Modelica." In ZOBEL AND MOELLER, Eds., *Proceedings of the 12th European Simulation Multiconference, ESM'98*, pp. 132–136. Society for Computer Simulation International, Manchester, UK.