BOND-GRAPH MODELING IN MODELICA™

Jan F. Broenink
Control Laboratory, EE Department, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands
phone +31-53-489 2793; fax +31-53-489 2223; e-mail J.F.Broenink@ el.utwente.nl

ABSTRACT

This paper discusses a *bond–graph* model library implemented in *Modelica*. Modelica is a new language for physical systems modeling with main objective to facilitate exchange of models and simulation specifications. Bond graphs are a domain–independent way of modeling the dynamics of physical systems.

Besides the presentation of the Modelica basic bond–graph library, also an application example is discussed, to test the usefulness of the library in practice.

The translation of existing bond–graph models to Modelica was a straightforward process, indicating that Modelica has the proper features for bond–graph modeling. However, the implicitly generated *sum–to–zero* connection equations are *not* useful for bond–graph modeling and in fact can obscure model specification in Modelica.

Our future aim is to build a Modelica import / export facility for our bond–graph / block–diagram modeling and simulation software 20-SIM.

Keywords: Object–oriented modeling, Bond graphs, Continuous languages, Control systems.

1 INTRODUCTION

The currently available modeling and simulation tools have their own proprietary model description languages, which complicate exchange of models between different tools. Having a real expressive and standardized modeling language, exchange of models is really possible without losing non-trivial language constructs of the proprietary language from which the porting process starts.

Modelica™ is a new language for physical systems modeling being developed in an international effort (Modelica, 1997; Mattsson et al., 1997). The main objective of this development is to facilitate the exchange of models, model libraries and simulation specifications. The Modelica design group covers a wide range of application domains, thus ensuring that the versatility of Modelica will be taken care of (Otter et al. 1997; Tummescheit and Klose, 1997; Mattsson, 1997).

Bond graphs are a domain-independent notion of physical systems modeling, where the physical processes are directly represented as vertices in a directed graph (Breedveld, 1984; Thoma, 1989; Cellier 1991) and the edges represent the *ideal* exchange of energy between the vertices. This domain independence makes bond graphs especially attractive in a multi-domain context. Furthermore, the equations associated with bond–graph elements can be automatically converted into simulation code, thus releasing the modeler of writing computable code as a simulation model.

The contribution of this paper is the implementation of a library of bond–graph elements in Modelica. This work serves as a test on the versatility of the language.

First, a brief overview of Modelica is given (section 2), and then the Modelica model library of bond–graph elements is presented (section 3). Section 4 presents an application example consisting of a computer–controlled physical system. In section 5, we draw conclusions.

2 MODELICA

Modelica is a language to describe the dynamic behavior of physical systems. It is inspired on principle of object–oriented software development (e.g. Rumbaugh et. al. 1991). Features of object orientation essential for physical systems modeling are covered, namely *hierarchical modeling*, *encapsulation*, and *inheritance*.

Essential features of Modelica are the following:

Models and submodels are declared as *classes*, with interfaces that are called *connectors*. A connector must contain all quantities needed to describe the interaction. Modification of a model definition is possible using the *extends* construct. Models can have submodels that can have submodels themselves. So, *hierarchical modeling* and *inheritance* are supported.

The equations of the models are described non-causally, and are *real* equations, i.e. the equality holds for all values of *time*, the independent variable. Furthermore, it is possible to describe the connections in terms of physical connections, i.e. pairs of bilaterally computed power–conjugated variables. This implies that the internal description of a model (its equations) can be separated from the interface, i.e. when using such a submodel the *exact* contents of it need not be known. Consequently, symbolic manipulation including sorting of the equations is necessary to obtain simulatable code. This is taken care of by the Modelica compiler. With this facility,

™ Modelica is a trademark of the *Modelica Design Group*, TC–1 of EUROSIM

of by the Modelica compiler. With this facility, *encapsulation* is taken care of.

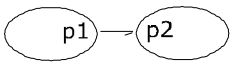
Furthermore, Modelica has a facility to specify how the connections are converted to computable code. Tagging a variable in a connection with *flow* results in simulation code where the variables involved are summed to zero. This is an implicit implementation of Kirchhoff's current law generalized to all physical domains. This generalization is legitimate, as is done in bond graphs. However, the connect statement with *flow* tags on a variable is not a pure connection anymore. Note that in the standard network–like descriptions in Modelica, one of the two variables in a connection has the tag *flow*.

Modelica is a textual model description language, and is as such more relevant for software tool builders then for the end user (i.e. physical systems modeler). Our experience is that graphically representing models as interconnected submodels displayed as icons supports their quick understanding. Furhtermore, most contemporary tools have graphical model editing facilities.

3 BOND–GRAPH LIBRARY IN MODELICA

The number of basic bond-graph elements is limited, and can therefore be enumerated in a bond–graph library. However, several properties of bonds need special attention when writing bond–graph models in Modelica:

1. Bonds are connected between ports of submodels and are point-to-point connections, implying that *exactly* one bond must be connected to a port. Compare this to (computer) cables which must be connected to the connectors of the devices.
2. Bonds are two-signal connections where these signals have opposite directions. These signals have the generalized names *effort* and flow. Their product is the power being exchanged.
3. Bonds embody an *ideal* connection, meaning that the port variables at one side of the bond *equal* the port variables at the other side of the bond. This implies that simulation–code statements coming from a bond connection are *2 equalities* (see also figure 1):



$$p1.effort = p2.effort$$
$$p1.flow = p2.flow$$

Figure 1 A bond connecting 2 ports (p1 and p2) of different submodels and its equations.

The above implies that the standard facility of Modelica that a variable in a connection tagged with *flow* sums to zero with the corresponding variable at the other end of the connection, may *not* be used in the bond–graph library. Therefore, in a bond–graph port definition, *neither* the effort *nor* the flow variable have a tag *flow* indicating that a sum–to–zero equation must be generated when converting the connection to computable code.

A bond–graph port is defined as follows:

```
connector BondPort "Bond Graph power port"
    Real e      "Effort variable";
    Real f      "Flow variable";
equation
ASSERT(Cardinality(This)==1,  "Power ports
    have only one edge connected to");
end BondPort;
```

A bond has a *power direction* and the elements put demands on the possible power directions of the bonds connected to their ports. Passive elements (R, C, I) have a power direction pointing inward, while active elements (Sources) have the power pointing outward. Using this convention results in positive values for parameters and helps resolving sign-placing problems. An attribute to the port is used to specify the restrictions:

```
partial model OnePortPassive
      "One port passive bond graph element"
    BondPort p  "Generic power port p";
equation
    ASSERT(Direction(p)==1,
       "Power direction towards element for
       passivity");
end OnePortPassive;
```

A bond is comprised of two signals flowing in opposite directions. This signal direction is called the *computational causality*, and is *not* needed during modeling, but *is* needed for deriving computable code. Again, the equations constituting the elements put demands on the computational causality. These demands can be specified as *causality restrictions* in the form of attributes of the ports. Especially the *storage elements*, also called energetic elements, impose a preferred causality on their ports. This means, that when the applied causality complies with the preferred causality, the generated equations are ODE's, otherwise DAE's are generated. The state variable used here is the conserved quantity, which is a specific quantity in each physical domain (p.e. charge and flux linkage in the electro–magnetic domain, position and impulse in the mechanical domain).

```
partial model OnePortEnergetic
  "One port storage element, being passive"
    extends OnePortPassive;
    Real  state      "Conserved quantity";
end OnePortEnergetic;
```

The C storage element is shown below. C elements describe the ideal behavior of mechanical springs (both translational and rotational), electrical capacitors, hydraulic vessels, heat capacitances.

```
model C
        "Bond Graph C element, storage of
        q-type conserved quantity"
    extends   OnePortEnergetic;
    parameter Real c        "Capacitance";
```

```
equation
    der(state) = p.f;
    p.e = state / c;
end C;
```

The initial condition of the state variable can be specified as an *attribute* belonging to that state variable. Note that the *causality restrictions* are not explicitly formulated in the Modelica description. Causality restrictions are not essential for deriving simulation code, but are useful during the model compilation process.

The I storage element is the dual form of the C storage element (the state is now the integral of the effort variable). Its conserved quantity is the generalized impulse. I elements describe the ideal behavior of masses, rotational inertias, electrical inductances, hydraulic inertias. Since the Modelica description of the I element does not have any new language feature, it is not given here.

The R element in its linear form imposes no causality restrictions, since its constitutive relation is an algebraic one, which can easily be inverted.

```
model R "Bond Graph resistor"
    extends   OnePortPassive;
    parameter Real R   "Resistance ";
equation
    p.e = R * p.f;
end R;
```

To describe the connecting structure, special junction structure elements are defined: The so-called *0-junction* and *1-junction*, to specify a common effort and a common flow structure respectively. Since these elements are ideal (i.e. no power dissipation or storage), besides the equality for the efforts (flows), there is a weighted sum for the other variable, namely the flows (efforts). The specification of a 1-junction in Modelica is as follows:

```
model J1       "Bond graph One junction"
   parameter Integer N  "# power ports";
   BondPort  P[N]      "extendable port";
equation
// Efforts sum up to 0, taking signs from
the power bond directions
   Direction(P)' * P.e = 0;
// Flows are all equal
   p.f[1] = p.f[2:N];
end J1;
```

Note that X' means the transpose of X.

The 0-junction is the dual form of the 1-junction: the efforts are all equal and the flows sum up to zero, taking signs from the power bond directions into account.

The other basic bond graph elements (TF - transformer, GY - gyrator) are defined in a similar way. Furthermore, resistors, sources, transformers and gyrators can have their parameter non-constant. The value is then fed through the interface as an incoming signal. Non-linear elements can be defined, reusing the linear ones. Only the equations need to be adapted. The extra variables and parameters used need of course be declared. An example is the *Modulated TransFormer*, shown below:

```
model MTF
"Bond graph modulated transformer element"
    extends TwoPortPassive;
    input   Real Modu "Modulation=e1/e2";
equation
    PowIn.e = Modu * PowOut.e;
    PowOut.f = Modu * PowIn.f;
end MTF;
```

This MTF is a specialization of a TwoPortPassive partial model, in which 2 power ports are declared (`PowIn` and `PowOut`).

In addition, block diagram elements can be specified easily. The interface now consists of signal inputs and

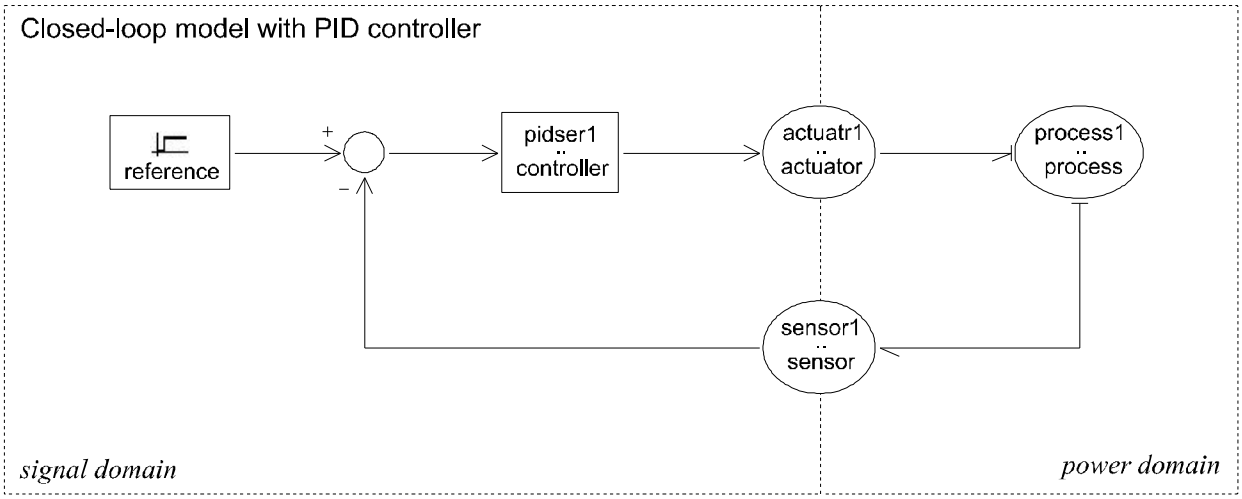


Figure 2 Application example

signal outputs.

4 APPLICATION EXAMPLE

The application example we have chosen, is a model of a computer controlled physical system. The top–level model is shown in figure 2. Besides the bond–graph library, several block–diagram libraries are used. The Modelica code of the top-level model is shown below.

```
model ControlledDevice
"Controlled physical system as application
example"
    StepGen  reference;  // SigGen lib
    PID      control;    // Controller lib
    Actuator actuator;
    Sensor   sensor;
    Process  process;
    Plus     sum;        // BlockDiagram
equation
    connect(reference.outp, sum.plusIn);
    connect(sum.outp, control.error);
    connect(control.steer, actuator.inp);
    connect(actuator.act, process.inp);
    connect(process.outp, sensor.sens);
    connect(sensor.outp, sum.minIn);
end ControlledDevice;
```

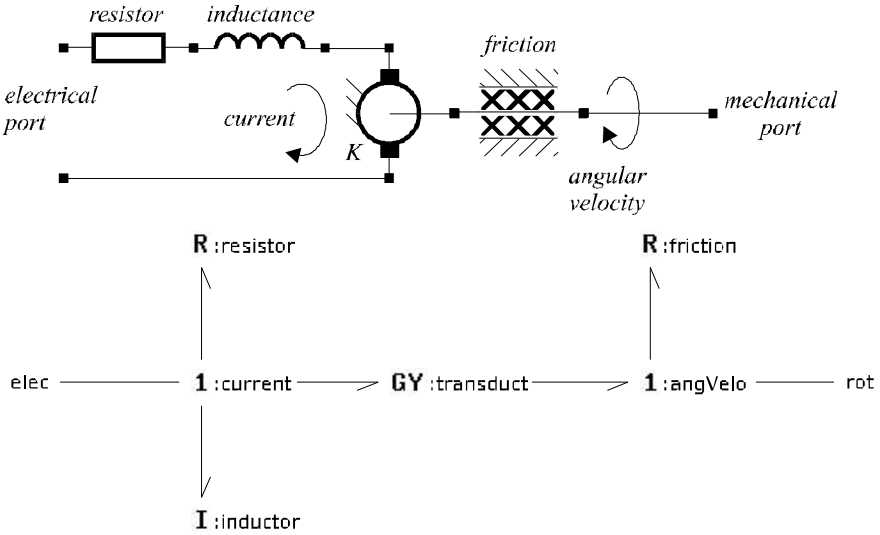
The listing of such a graph model clearly shows two parts: one is the declaration part, in which all used submodels get a local name (to distinguish two instantiations of the same submodel). These local names are used in the *equation* part of a graph description contains *connect* statements, to describe the connections between ports of the submodels. The ports of the submodels are identified by concatenating the submodel name and the port name using a dot. In fact, each connect statement denotes one edge in the graph. Also the ports (inputs or outputs) of the model itself are connected to submodels in connect statements.

The submodels `Actuator`, `Process` and `Sensor` are graphs having submodels themselves. The submodels `Actuator` and `Motor` are given below, and the physical model and bond graph of the submodel `Motor` is shown in figure 3.

```
model Actuator
"The actuator of our application example"
    input   Real inp  "Control input";
    BondPort act      "Actuation power"
    Limit    limit    "Prevent saturation"
    MSe    VoltSource "Ideal power supply"
    Motor    motor    "The engine"
equation
    connect(inp, limit.inp);
    connect(limit.outp, VoltSource.Modu);
    connect(VoltSource.p, Motor.elec);
    connect(Motor.rot, act);
end Actuator;

model Motor             "the engine"
    BondPort elec, rot;
    J1 current, angVelo;
    R resistor, friction;
    I inductance;
    GY transduct;
equation
    connect(elec, current.p[1]);
    connect(resistor.p, current.p[2]);
    connect(inductance.p, current.p[3]);
    connect(current.p[4],transduct.PowIn);
    connect(transduct.PowOut,
            angVelo.p[1]);
    connect(angVelo.p[2], friction.p);
    connect(angVelo.p[3], rot);
end Motor;
```

The Modelica listings presented here only show the information necessary to derive the simulation code. Modelica also has language constructs to describe the



Figuur 3 The physical model and the bond graph of the motor submodel

drawing itself. As most contemporary tools provide facilities to build models graphically, Modelica has language constructs to represent icons and the graphical layout (i.e. positions and sizes of the icons and additional information on the edges when they are not straight lines). The attributes describing an icon are separated from the mathematical description and specified using the *icon* attribute, because the graphical representation has no influence on the mathematical description of the model. A basic mechanism using a vector format, is provided to be as neutral as possible, allowing easy exchange to other tools. Furthermore, a vector format is superior over a bitmap or raster images, since a vector format allows scaling without loss of quality, and the amount of data needed is generally less. The icon is used on the next higher level in the model hierarchy.

5 CONCLUSION

A Modelica bond graph library has been defined, and our experience is that Modelica has possibilities to describe hierarchical bond graph models. The application example also approves this. However, some remarks need to be made:

All bond–graph elements were elegantly described in the bond–graph Modelica library, using the essential object orientation features *inheritance* and *encapsulation*. Furthermore, facilities to write equations in a (computational) acausal form, and constructs to build and check power–bond connections also contribute to the elegant implementation of the bond–graph library in Modelica. However, causality restrictions cannot be expressed in Modelica, which might complicate the model compilation process.

A more relevant problem is that the sum–to–zero facility in the connect statement of Modelica is *not* useful for bond–graph like modeling. It is therefore *not* used in the bond–graph library.

Furthermore, the translation of bond-graph models from specific bond–graph modeling and simulation software, such as 20-SIM (Broenink, 1997) to Modelica is in principle a straightforward process. In fact, the 20-SIM model libraries were a basis for the Modelica bond–graph library. However, to *really* facilitate the exchange of models and model libraries, those existing (bond graph) modeling tools need to have both a export module which generates directly Modelica code and an import module capable of reading Modelica models. It depends on the expressiveness of the bond graph modeling language used compared to Modelica if both the import and export facility can cover all possible language constructs.

It should be noted that Modelica is a textual model description language, and is primary a means to exchange models. As such it is more relevant for software tool builders then for the end user (i.e. physical systems modeler).

A more general conclusion is that bond–graph modeling is in fact a form of *object–oriented physical systems modeling.*

Our aim is to build a Modelica import / export facility for our bond graph / block diagram modeling and simulation software, 20-SIM (Broenink, 1990; Broenink and Weustink, 1995, Broenink, 1997).

References

Breedveld, P.C., (1984), *Physical systems theory in terms of bond graphs*, Ph.D. thesis, University of Twente, Enschede Netherlands.

Broenink, J.F. and P.B.T. Weustink, (1995), PC-version of the bond graph modeling and simulation tool CAMAS, *Proc Int conf, Bond graph modeling and simulation*, Las Vegas, SCS simulation series 27 no. 1: 203-208.

Broenink, J.F., (1990), *Computer aided modeling and simulation: a bond graph approach*, Ph.D. thesis, University of Twente, Enschede Netherlands.

Broenink, J.F., (1997), Modelling, simulation and analysis with 20-SIM, *Journal A* 38, no.3 (Sept): 22-25.

Cellier, F.E., 1991, *Continuous system modeling*, Springer Verlag.

Mattsson, S.E., H.E. Elmqvist and J.F.Broenink (1997), Modelica™ – An international effort to design the next generation modeling language, *Journal A* 38, no.3 (Sept): 16-19.

Mattsson, S.E., (1997), On Modeling of Heat Exchangers in Modelica, *European Simulation Symposium* (this issue).

Modelica, (1997), Modelica™ - a unified object oriented language for physical systems modeling, *http://www.Dynasim.se/Modelica*

Otter, M., C. Schlegel, H.E. Elmqvist, (1997), Modeling and Realtime Simulation of an Automatic Gearbox using Modelica, *European Simulation Symposium* (this issue).

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, (1991), *Object oriented modeling and design*, Prentice Hall.

Thoma, J.U., (1989), *Simulation by bond graphs – Introduction to a graphical method*, Springer Verlag.

Tummescheit H. and M. Klose, (1997), A Case Study Applying Object-oriented Concepts for Multi-facet Modeling, *European Simulation Symposium* (this issue).

Bibliography

Jan F. Broenink received the MSc degree in electrical and biomedical engineering in 1984 and the PhD degree in electrical engineering in 1990 from the University of Twente. His PhD research was in the design of computer facilities for modeling and simulation of physical systems using bond graphs.

He is presently an Assistant Professor of the Department of Electrical Engineering, University of Twente. His research interest include development of computer tools for modeling,simulation and implementation of embedded control systems; and robotics.