Object-Oriented Modeling of Complex Physical Systems Using the Dymola Bond-Graph Library

François E. Cellier
University of Arizona
P.O. Box 210104
Tucson, AZ, 85721-0104, USA
cellier@ece.arizona.edu

Robert T. McBride
Raytheon Missile Systems
P.O. Box 11337, Bldg. 805, M/S L-5
Tucson, AZ, 85734-1337, USA
rtmcbride@raytheon.com

Keywords: Dymola, object-oriented modeling, topological modeling, hierarchical modeling, graphical modeling.

Abstract

In this paper, a new bond-graph library is introduced, programmed as part of the *Dymola* object-oriented graphical modeling environment. It is shown that the embedding of bond graphs into the *Dymola* modeling framework adds both expression power and flexibility to the bond-graph modeling methodology.

The *Dymola* modeling framework is summarized, and the new bond-graph library is introduced. An (academic) example of a simple position control system involving a hydraulic motor demonstrates the power of the modeling environment.

INTRODUCTION

Finding just the right balance between generality and specificity, determining which aspects of the modeling enterprise ought to be hard-coded into the software environment, and which other portions ought to be kept open to modification by the end user is a difficult decision to make.

In the *Dymola* object-oriented graphical modeling framework [1], this lesson had to be learnt the hard way, and it took many iterations to get it right. Until version 4 of *Dymola* had been released, the authors of this paper rejected to make use of the graphical front-end of the environment, because it hadn't been flexible enough. With version 4 of the software, graphical modeling became easy and intuitive, and the creation of a powerful hierarchical graphical bond graph library had not only become feasible, but was even easy to accomplish. The new bond graph library is being introduced in this paper.

THE BOND-GRAPH CONNECTORS

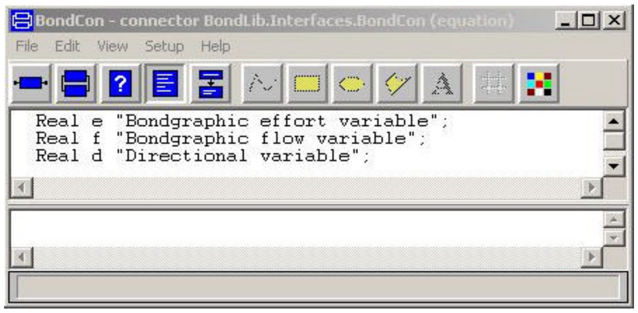
In [2], Cellier advocated the use of gyro-bonds as a means for exploiting *Dymola* nodes to represent both 0- and 1-junctions.

Dymola offers two types of variables: across variables and through variables. In a *node*, the *across variables* are set equal across all connectors, whereas the *through variables* add up to zero. This modeling construct can be exploited in bond graph modeling, since, in a 0-junction, the across variables correspond to efforts, whereas the through variables correspond to flows. Yet, in a 1-junction, the correspondence is reversed. Consequently, both types of junctions can be represented by nodes, if each bond itself is a symplectic gyrator, and if additional rules are specified that ensure that (i) junctions always toggle, i.e., no two 0-junctions or 1-junctions are connected by a bond, and (ii) all bond-graphic elements are connected to 0-junctions only.

These rules are, however, too constraining for a graphical modeling environment. For example, thermal systems exhibit often 0-junctions with many bonds attached to them, representing e.g. a room at a certain temperature, with many bonds, representing the walls, the windows, the doors, the ceiling, and the floor interacting with that room. It must be possible to graphically split that 0-junction into a series of distinct 0-junctions, each one with a smaller number of connections.

For this reason, the new bond-graph library treats bonds as symplectic transformers rather than symplectic gyrators, and it treats both efforts and flows as across variables. As a consequence of this decision, the junctions now need to be explicitly modeled.

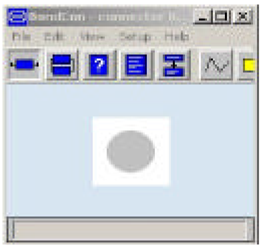
The bond-graphic connector is programmed as follows:



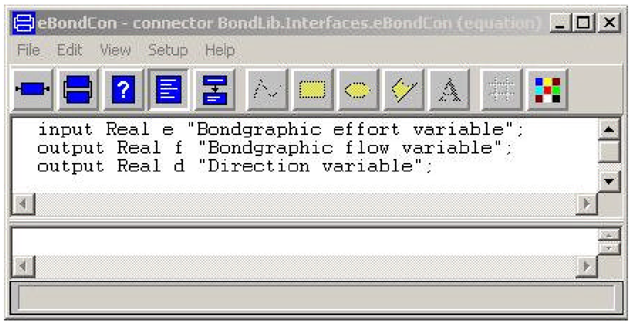
```
Real e "Bondgraphic effort variable";
Real f "Bondgraphic flow variable";
Real d "Directional variable";
```

There are three variables referenced in the connector, the usual effort and flow variables, *e* and *f*, as well as a directional variable, *d*, which indicates whether the direction of positive flow is into the connector (*d*=+*1*) or out of the connector (*d*=-*1*).

Each *Dymola* modeling entity contains three windows: an *equation window*, shown above; a *diagram window*, used for graphical modeling by topologically interconnecting previously defined models, and an *icon window*, used to represent the model graphically at the next hierarchical level. The iconic representation of the bond-graphic connector is a grey dot:



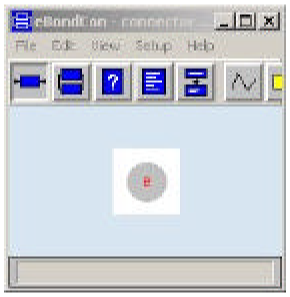
There exist two additional definitions, used for causal bonds. For example, the *e*-connector is defined as follows:


eBondCon - connector BondLib.Interfaces.eBondCon (equation)
File Edit View Setup Help

```
 input Real e "Bondgraphic effort variable";
 output Real f "Bondgraphic flow variable";
 output Real d "Direction variable";
```

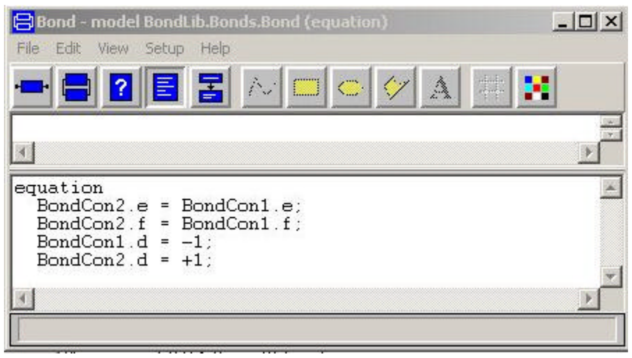
In *Dymola*, all variables are assumed a-causal by default, but they can be made causal, by declaring them accordingly.

The icon of an *e*-connector is a grey dot with the letter "*e*" embedded in it:



The *f*-connector has the causalities of the effort and flow variables reversed.

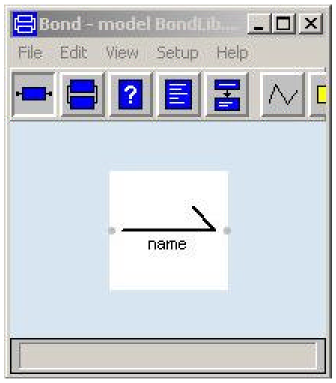
Using these connectors, it is now possible to define the bonds. A-causal bonds are defined using a *Dymola model*:


Bond - model BondLib.Bonds.Bond (equation)
File Edit View Setup Help

```
equation
  BondCon2.e = BondCon1.e;
  BondCon2.f = BondCon1.f;
  BondCon1.d = -1;
  BondCon2.d = +1;
```

Two bond-graph connectors were dragged into the diagram window. They were named *BondCon1* and *BondCon2*, respectively. The equation window defines the equations governing a bond.

The icon associated with the bond model is shown below:



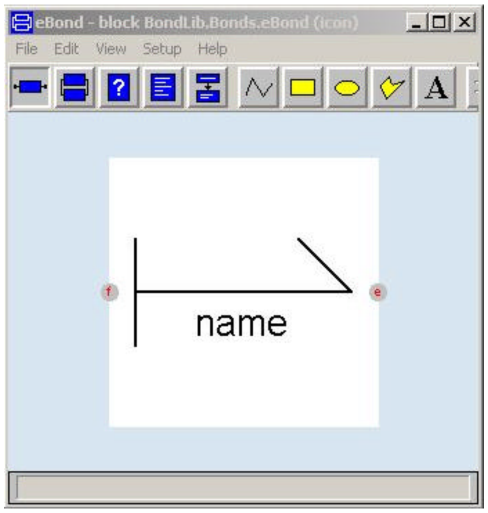
The icon shows the two inherited bond-graph connectors as well as the bond connecting them.

Each *Dymola* entity has a name. By placing the text "*%name*" on the icon, Dymola knows to place the true name of an invoked bond underneath the bond upon dragging it into the diagram window.

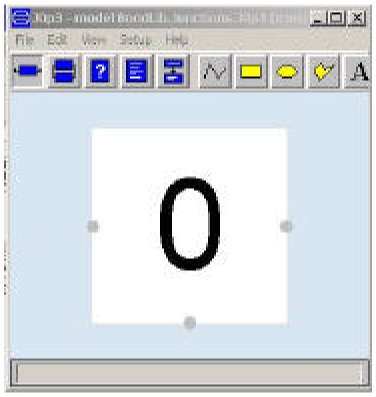
Causal bonds make use of *e*- and *f*-connectors. For example, the *f*-bond invokes an *f*-connector at the side of the causality stroke, and an *e*-connector at the other side.

In Dymola, *e*-bonds and *f*-bonds are modeled as *blocks* rather than as *models*, which forces pre-defined causalities upon all variables.

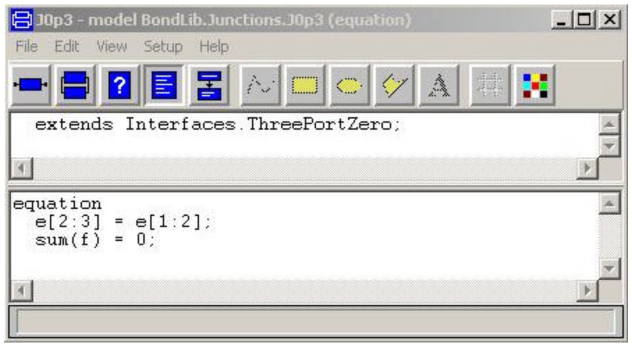
There is no need to ever use causal bonds in *Dymola*, since *Dymola* is perfectly capable of determining the correct causalities on its own, but their use supports both readability and debugging of models, and therefore, the authors of this paper prefer them whenever possible, i.e., whenever the causality is fixed.


eBond - block BondLib.Bonds.eBond (icon)
File Edit View Setup Help
f
e
name

Let us now look at a 0-junction with three bond attachments. Junctions exist only in their a-causal form. No pre-assigned causality is needed here, because the correct causality will be inherited by the bonds that are attached to the junction.


J0p3 - model BondLib.Junctions.J0p3 (icon)
File Edit View Setup Help
0

Its equation window is programmed as follows:


J0p3 - model BondLib.Junctions.J0p3 (equation)
File Edit View Setup Help

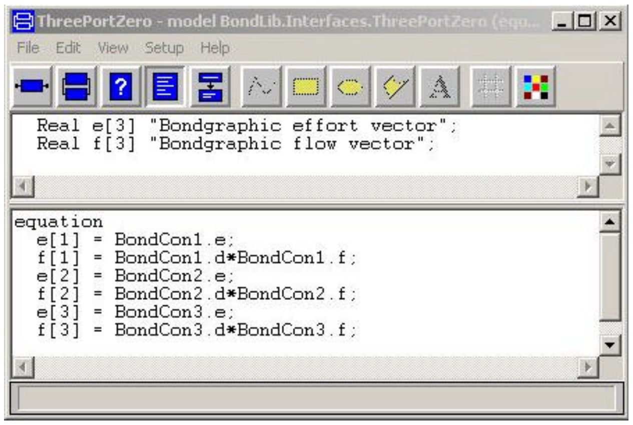
```
  extends Interfaces.ThreePortZero;
```

```
equation
  e[2:3] = e[1:2];
  sum(f) = 0;
```

Rather than dragging three bond-graph connectors into the diagram window, this model inherits (extends) another model that contains most of the equations that are needed to describe a three-bond 0-junction.

Dymola offers a matrix manipulation language similar to *Matlab* for the description of matrix and vector operations.

The *ThreePortZero* model contains the following equations:


ThreePortZero - model BondLib.Interfaces.ThreePortZero (equ...
File Edit View Setup Help

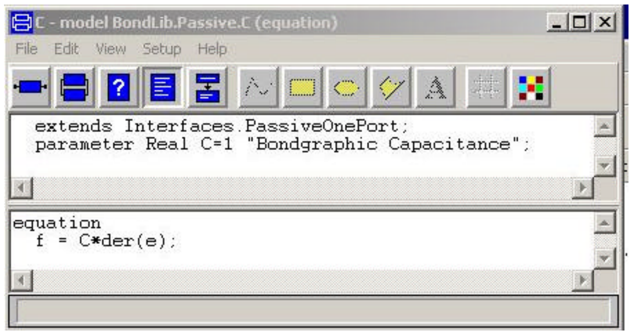
```
  Real e[3] "Bondgraphic effort vector";
  Real f[3] "Bondgraphic flow vector";
```

```
equation
  e[1] = BondCon1.e;
  f[1] = BondCon1.d*BondCon1.f;
  e[2] = BondCon2.e;
  f[2] = BondCon2.d*BondCon2.f;
  e[3] = BondCon3.e;
  f[3] = BondCon3.d*BondCon3.f;
```

The model packs the individual bond connector variables into an effort vector and a flow vector, and in the case of the flow variables, takes into account the direction of flow.

THE BOND-GRAPH LIBRARY

Let us now look at the bond-graphic element models. The capacitor model may serve as an example.

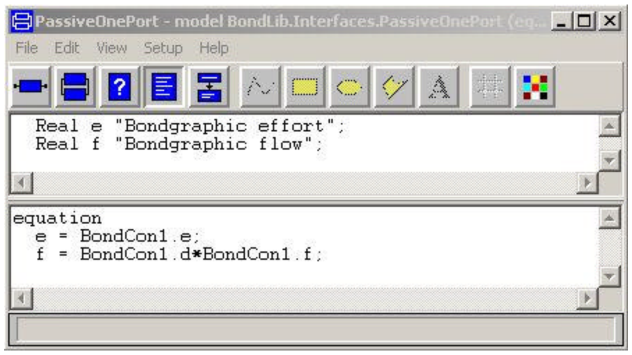

C - model BondLib.Passive.C (equation)
File Edit View Setup Help

```
  extends Interfaces.PassiveOnePort;
  parameter Real C=1 "Bondgraphic Capacitance";
```

```
equation
  f = C*der(e);
```

It inherits the *PassiveOnePort* model, declares the parameter C, assigns a default value of 1.0 to it, and defines the capacitive equation.

The *PassiveOnePort* model defines the following equations:

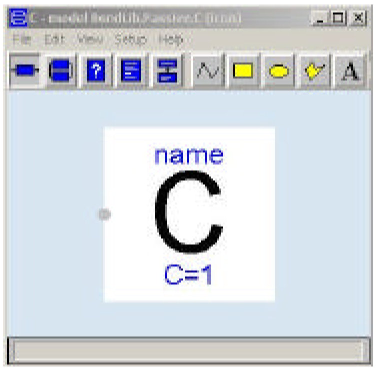

PassiveOnePort - model BondLib.Interfaces.PassiveOnePort (eq...
File Edit View Setup Help

```
  Real e "Bondgraphic effort";
  Real f "Bondgraphic flow";
```

```
equation
  e = BondCon1.e;
  f = BondCon1.d*BondCon1.f;
```

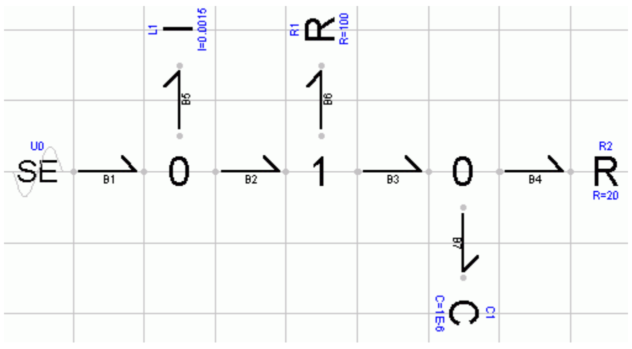
It drags the bond-graph connector into the diagram window, and assigns the connector variables to the appropriate effort and flow variables of the bond-graphic OnePort.

The icon window of the capacitor looks as follows:



Again, the name is added to the model. In addition, the parameter value is displayed by adding the text: "*C=%C*" to the icon.

By now, we are ready to create bond graphs, such as:

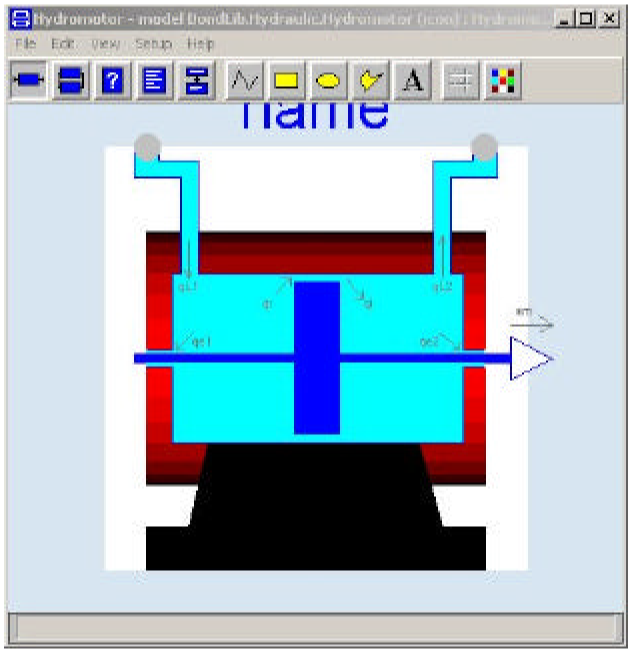


This looks no different than a bond graph generated with any other bond graph drawing tool. However, there *is* a big difference: *Dymola* doesn't know anything about bond graphs. The entire bond graph modeling environment was created within the *Dymola* modeling framework under full control by the modeler. This provides a degree of flexibility to the modeling environment that, to the best knowledge of the authors of this paper, cannot be found elsewhere.

A POSITION CONTROL SYSTEM

In the following section, a position control system involving a hydraulic motor is described. It shall be shown, how component models of increasing complexity can be built hierarchically on the basis of simpler models, and how these can be connected to describe complex physical systems.

We start by describing the hydraulic motor:

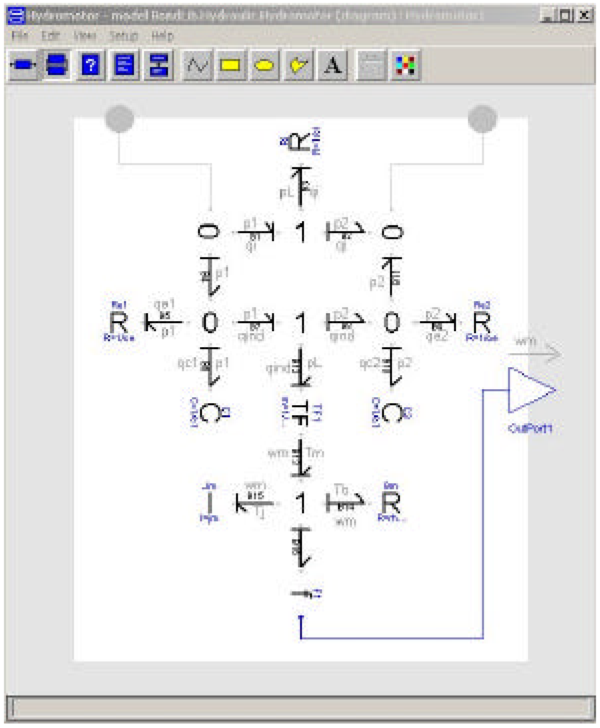

Hydromotor - model BondLib.Hydraulic.Hydromotor (icon)
File Edit View Setup Help
name
qL1
qL2
qi
qe1
qe2
wm

The schematic of the hydraulic motor is at the same time also its iconic representation. The hydraulic motor has two bond-graphic connectors, describing the in- and outflow of fluid, as well as a signal connector, describing the angular velocity of its axle.

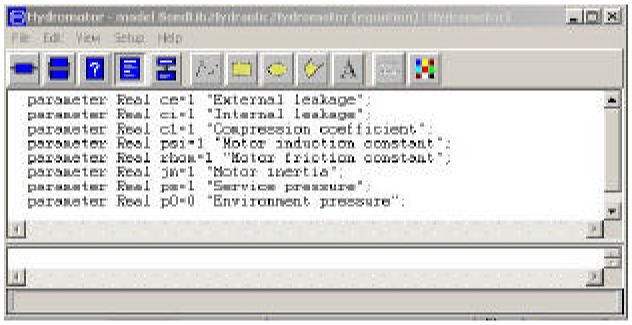
The diagram window is given next. It is an ordinary bond graph describing the hydraulics of the motor. The variable names of efforts and flows were added manually to the diagram window as text for documentation purposes.

The only unusual element is the *f*-element. It represents a flow sensor. This element sets the corresponding effort to 0, and senses the incoming flow, which is then passed on as a signal. Conventional bond graphs make use of activated bonds for this purpose.

The modeler will have to remember that the model contains the two 0-junctions at the top, but not the associated bonds, i.e., the bond-graph connectors connect to the junctions, not to the bonds. Consequently, the model that connects to the hydraulic motor will have to end in bonds, since it is not possible, using the *Dymola* bond graph library, to have two bonds next to each other without a junction in between, or to have two junctions next to each other without a bond connecting them. Otherwise, the directional *d*-variable wouldn't be defined.

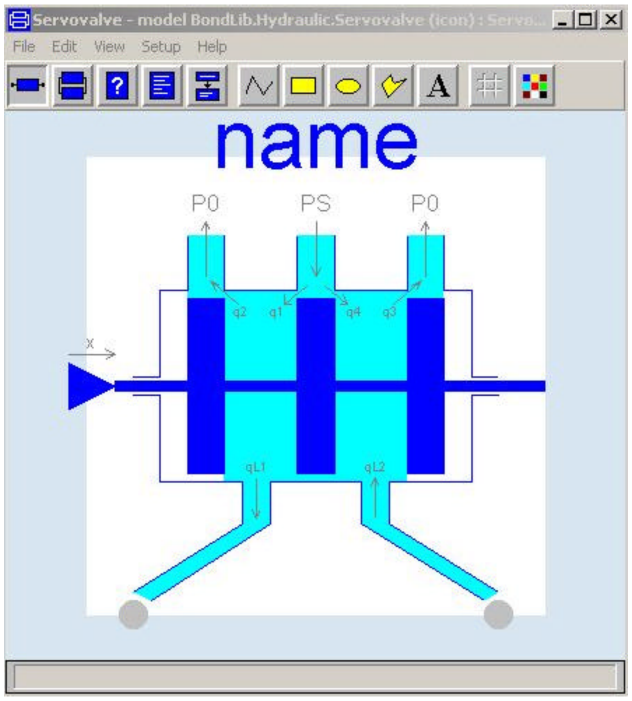

Hydromotor - model BondLib.Hydraulic.Hydromotor (diagram) : Hydromotor
File Edit View Setup Help
R
0
1
C
TF
I
wm
OutPort1

Notice further the propagation of parameter values. Within the equation window, the parameters of the hydraulic motor are defined and assigned default values. These are passed on to the underlying *R*-, *C*-, *I*-, and *TF*-elements:

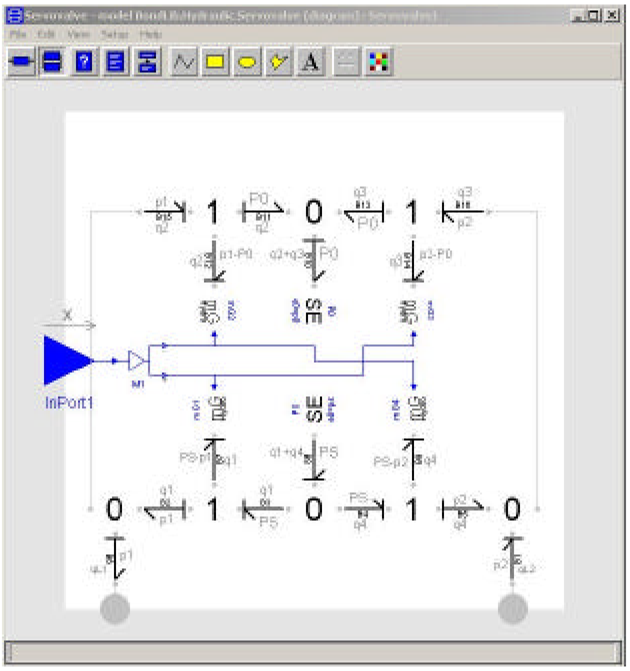

Hydromotor - model BondLib.Hydraulic.Hydromotor (equation) : Hydromotor
File Edit View Setup Help

```
parameter Real ce=1 "External leakage";
parameter Real ci=1 "Internal leakage";
parameter Real c1=1 "Compression coefficient";
parameter Real psi=1 "Motor induction constant";
parameter Real rhom=1 "Motor friction constant";
parameter Real jm=1 "Motor inertia";
parameter Real ps=1 "Service pressure";
parameter Real p0=0 "Environment pressure";
```

The hydraulic motor is controlled by a servo valve. The servo valve operates as follows:


Servovalve - model BondLib.Hydraulic.Servovalve (icon) : Servo...
File Edit View Setup Help
name
P0
PS
P0
q2
q1
q4
q3
x
qL1
qL2

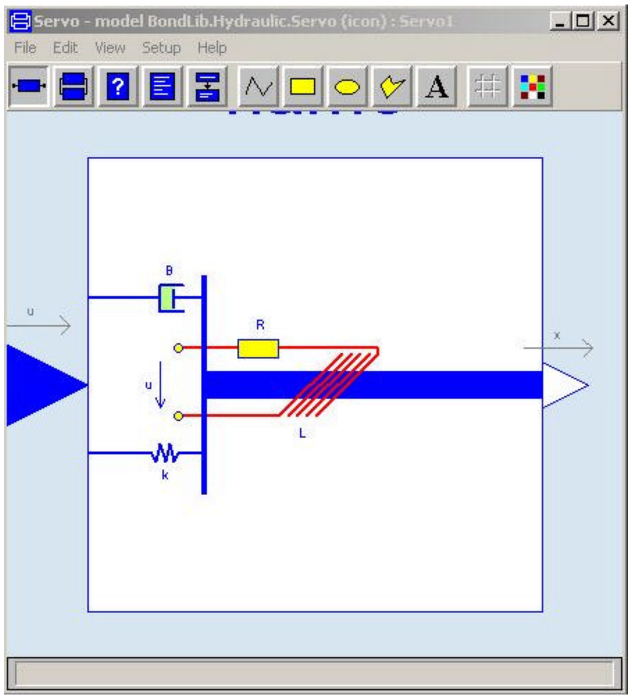
It is modeled by the bond graph shown below.


Servovalve - model BondLib.Hydraulic.Servovalve (diagram) : Servovalve
File Edit View Setup Help
1
0
SE
x
InPort1

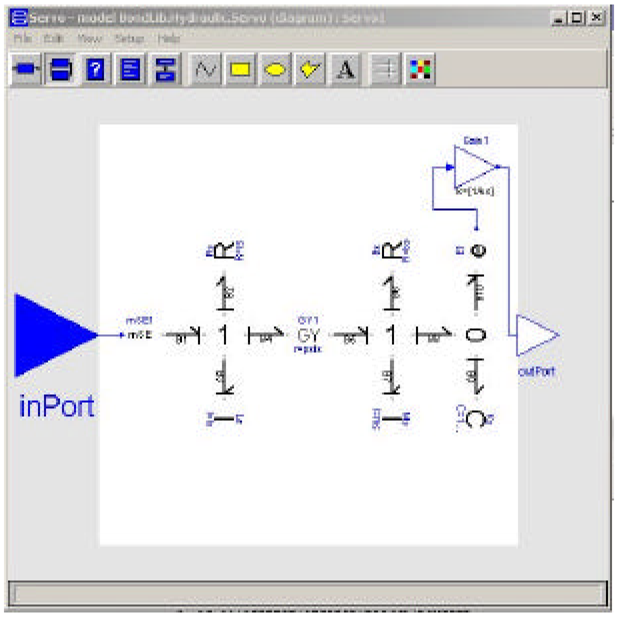
The four turbulent flows, q_1, q_2, q_3, and q_4, are modeled as non-linear resistors (or rather conductors). These are furthermore modulated by the under-lap caused by the position of the tongue, x, which is imported into the model as a signal input.

Notice that the bond-graphic connectors here indeed connect to bonds rather than junctions.

We shall now model the control of the tongue of the servo valve.

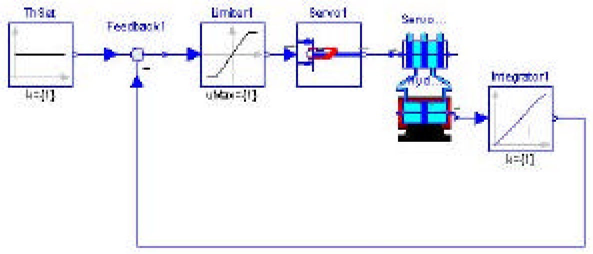

Servo - model BondLib.Hydraulic.Servo (icon) : Servo1
File Edit View Setup Help
B
R
u
x
L
k

This model has no bond-graphic connectors. Yet, internally, the device is modeled using bond graphs, as shown below:


Servo - model BondLib.Hydraulic.Servo (diagram) : Servo1
File Edit View Setup Help
Gain1
inPort
mSE
GY
outPort

The voltage is converted to a bond-graphic signal by use of a modulated effort source. The position of the tongue is proportional to the effort of the capacitor. It is sensed by use of an *e*-element, which sets the flow to zero, and senses the effort.

The control system can now be built. It is modeled by use of standard block-diagram methodology.



By handing full control of the modeling environment over to the modeler, the *Dymola* framework enables the user to employ the most adequate modeling methodology for each task. For control systems, block diagrams are the appropriate tool of choice. Yet, each one of the underlying physical systems has been modeled using bond graphs, which again, was the most appropriate tool for the task at hand.

CONCLUSIONS

The paper has introduced a new bond graph library programmed as an application of the *Dymola* modeling framework and software. It was demonstrated by means of an example that this environment offers very powerful features enhancing greatly the flexibility of the bond graph approach to modeling.

REFERENCES

[1] Brück, D., H. Elmqvist, S.E. Mattsson, and H. Olsson (2002), "Dymola for Multi-Engineering Modeling and Simulation," *Proc. Modelica'2002 Conference*, Munich, Germany, p. 55:1-9, http://www.modelica.org/Conference2002/papers/p07_Brueck.pdf .

[2] Cellier, F.E. (1991), *Continuous System Modeling*, Springer-Verlag, New York.