Bond Graphs for Mechanical Systems

- We shall look today in a bit more detail at the modeling of 1D mechanical systems using bond graphs.
- First, we shall look at the problem of holonomic constraints in mechanical systems.
- Then, we shall discuss how a wrapped mechanical bond graph library may be constructed.
- Finally, we shall look at a symbolic algorithm for state selection.

Table of Contents

State Variables in Mechanical Systems

- We have already seen that masses (inertias) can be modeled using bond graph inductors, whereas springs can be modeled using bond graph capacitors. Hence the natural state variables in a bond graph description of a mechanical system are the absolute (angular) velocities of the bodies and the spring forces (torques).
- In a model of a mechanical system described in this fashion, the (angular) positions are missing. They are not needed for a proper and complete description of the dynamics of the system.

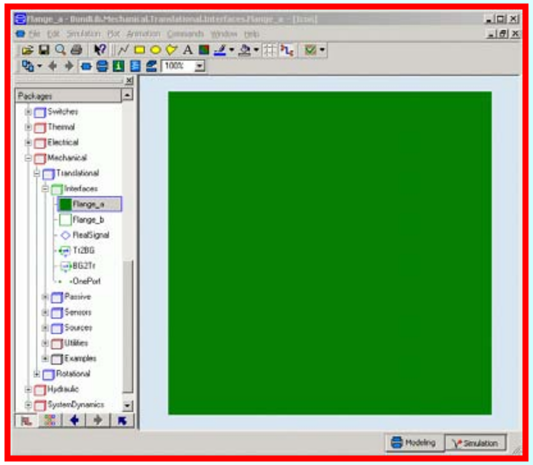
Holonomic Constraints

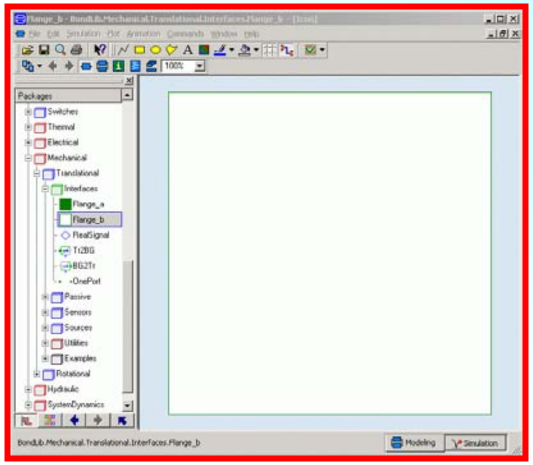
- This causes problems, as it is relevant to know whether two bodies occupy the same space at the same time, i.e., whether they bump into each other or not.
- Also, when two bodies are connected at a point (e.g. through a joint), it is insufficient to state that the ***velocities*** of these points are equal. It should be stated that their ***positions*** are identical.
- Such positional constraints are being referred to as ***holonomic constraints*** in the mechanical literature.
- The corresponding velocity constraints (non-holonomic constraints) don't need to be specified separately, because they can be derived automatically from the holonomic constraints.

Holonomic Constraints II

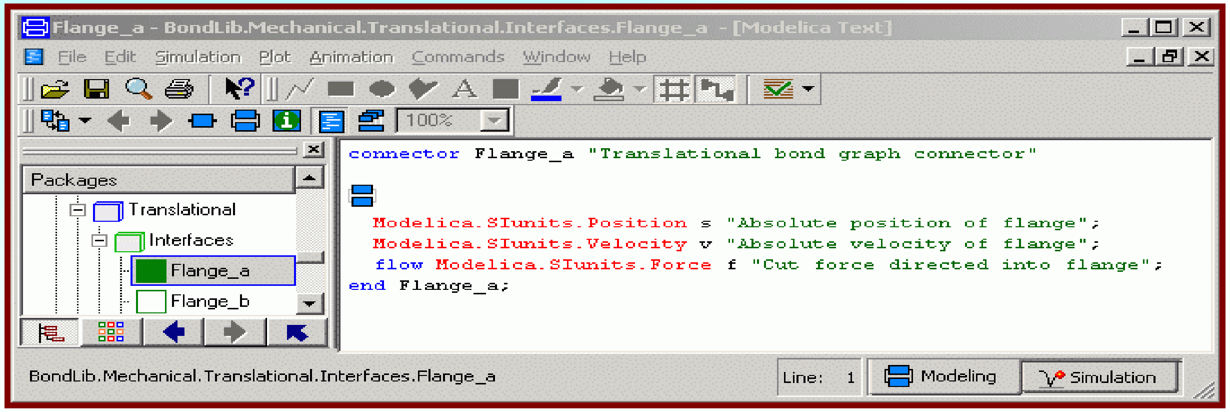
- For this reason, it may be better to find an alternate description that uses the ***absolute velocities and positions of bodies*** as state variables, leaving the spring forces out.
- Can this be done within the framework of the bond graph methodology?
- It can, and this is how the two wrapped 1D mechanical bond graph libraries (for translational and rotational motions) have been built.

The Mechanical Connectors





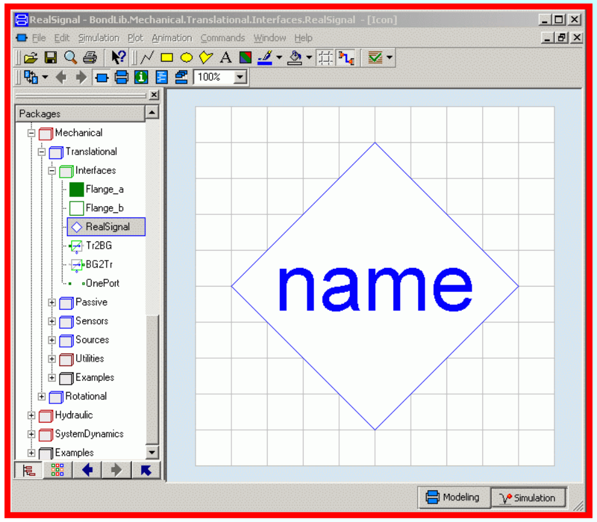
- We introduce two translational flange connectors. These are similar to those of the standard library, but they are not identical, as they shall contain a second across variable: the velocity, ***v***.
- Hence our mechanical models will be ***incompatible with those of the standard library***.
- The two flange models are actually identical. They are both offered for optical reasons only.

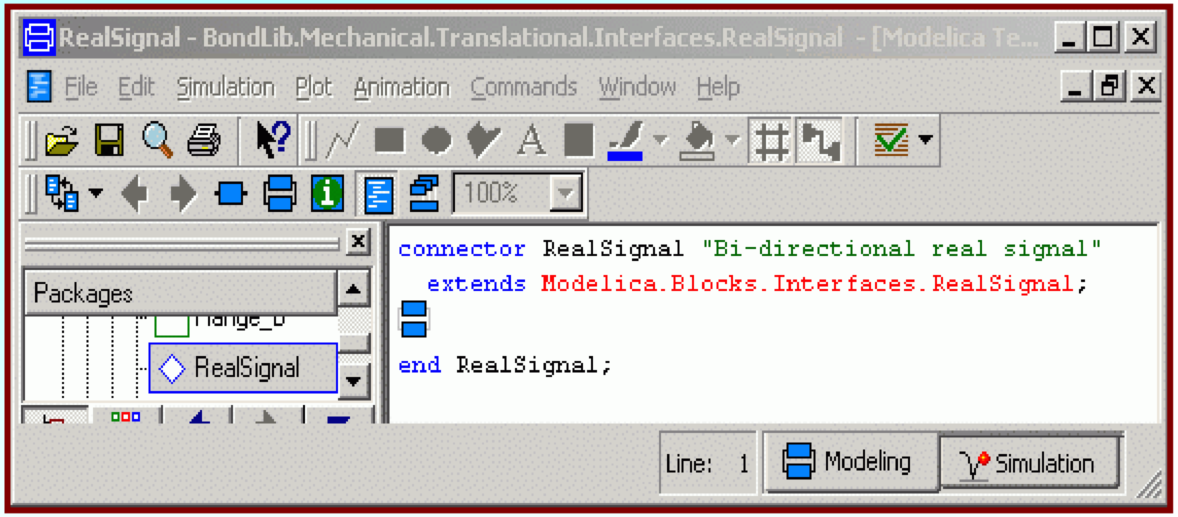

Flange_a - BondLib.Mechanical.Translational.Interfaces.Flange_a - [Modelica Text]
File Edit Simulation Plot Animation Commands Window Help
100%
Packages
Translational
Interfaces
Flange_a
Flange_b
BondLib.Mechanical.Translational.Interfaces.Flange_a
Line: 1
Modeling
Simulation

```
connector Flange_a "Translational bond graph connector"

  Modelica.SIunits.Position s "Absolute position of flange";
  Modelica.SIunits.Velocity v "Absolute velocity of flange";
  flow Modelica.SIunits.Force f "Cut force directed into flange";
end Flange_a;
```

The Mechanical Connectors II

- We also need a real signal connector. This is similar to the input and output connectors of the blocks library, but the signal is bidirectional, rather than being unidirectional.

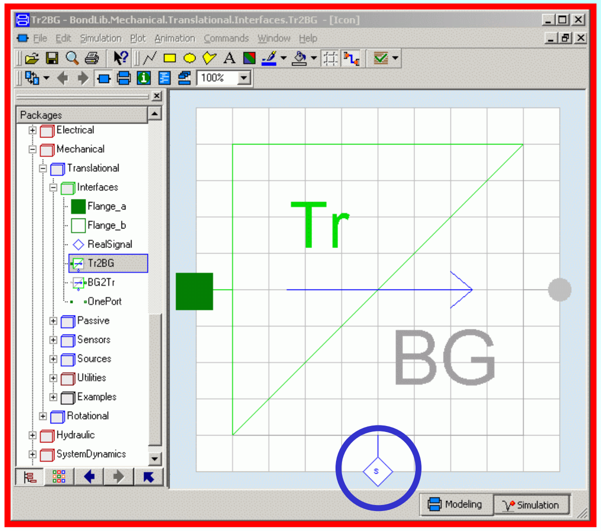


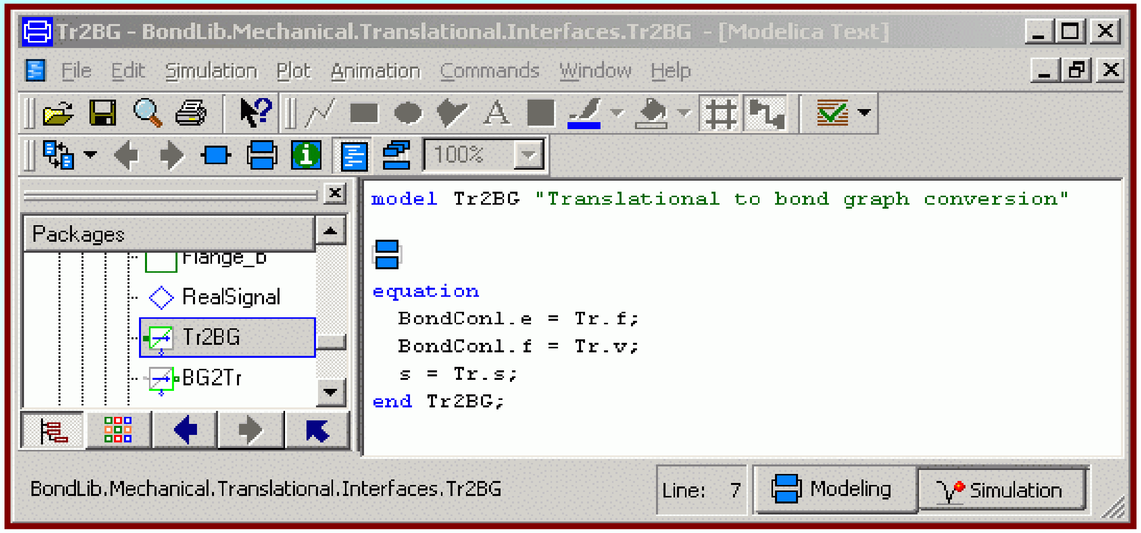

RealSignal - BondLib.Mechanical.Translational.Interfaces.RealSignal - [Modelica Te...
File Edit Simulation Plot Animation Commands Window Help
100%
Packages
RealSignal
Line: 1
Modeling
Simulation

```
connector RealSignal "Bi-directional real signal"
  extends Modelica.Blocks.Interfaces.RealSignal;

end RealSignal;
```

The Wrapper Models

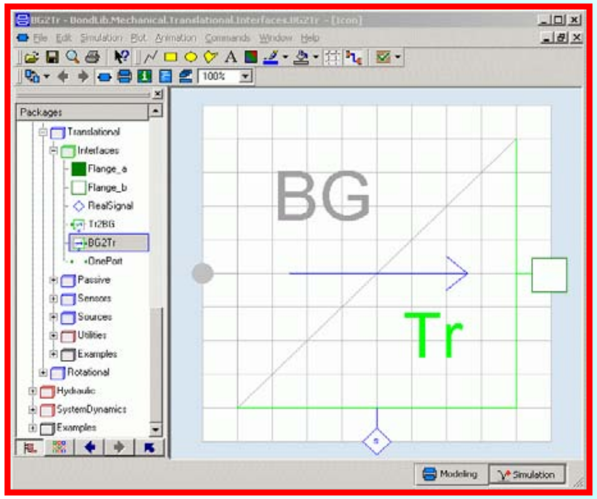
- We need wrapper models that convert the mechanical connectors to bond graph connectors and back.
- Since the bond graph connector cannot include the positional information, this must be separated out into a second signal connector.

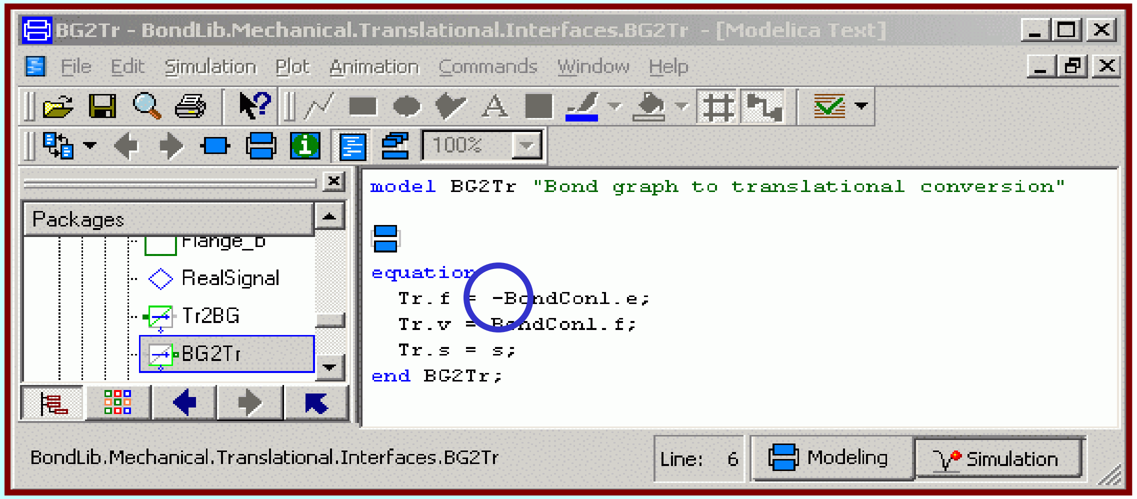



Tr2BG - BondLib.Mechanical.Translational.Interfaces.Tr2BG - [Modelica Text]
File Edit Simulation Plot Animation Commands Window Help
100%
Packages
Flange_b
RealSignal
Tr2BG
BG2Tr
BondLib.Mechanical.Translational.Interfaces.Tr2BG
Line: 7
Modeling
Simulation

```
model Tr2BG "Translational to bond graph conversion"

equation
  BondCon1.e = Tr.f;
  BondCon1.f = Tr.v;
  s = Tr.s;
end Tr2BG;
```

The Wrapper Models II



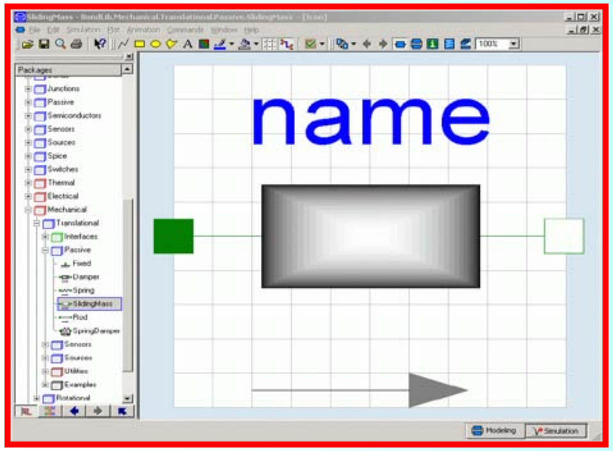

BG2Tr - BondLib.Mechanical.Translational.Interfaces.BG2Tr - [Modelica Text]
File Edit Simulation Plot Animation Commands Window Help
100%
Packages
Flange_b
RealSignal
Tr2BG
BG2Tr
BondLib.Mechanical.Translational.Interfaces.BG2Tr
Line: 6
Modeling
Simulation

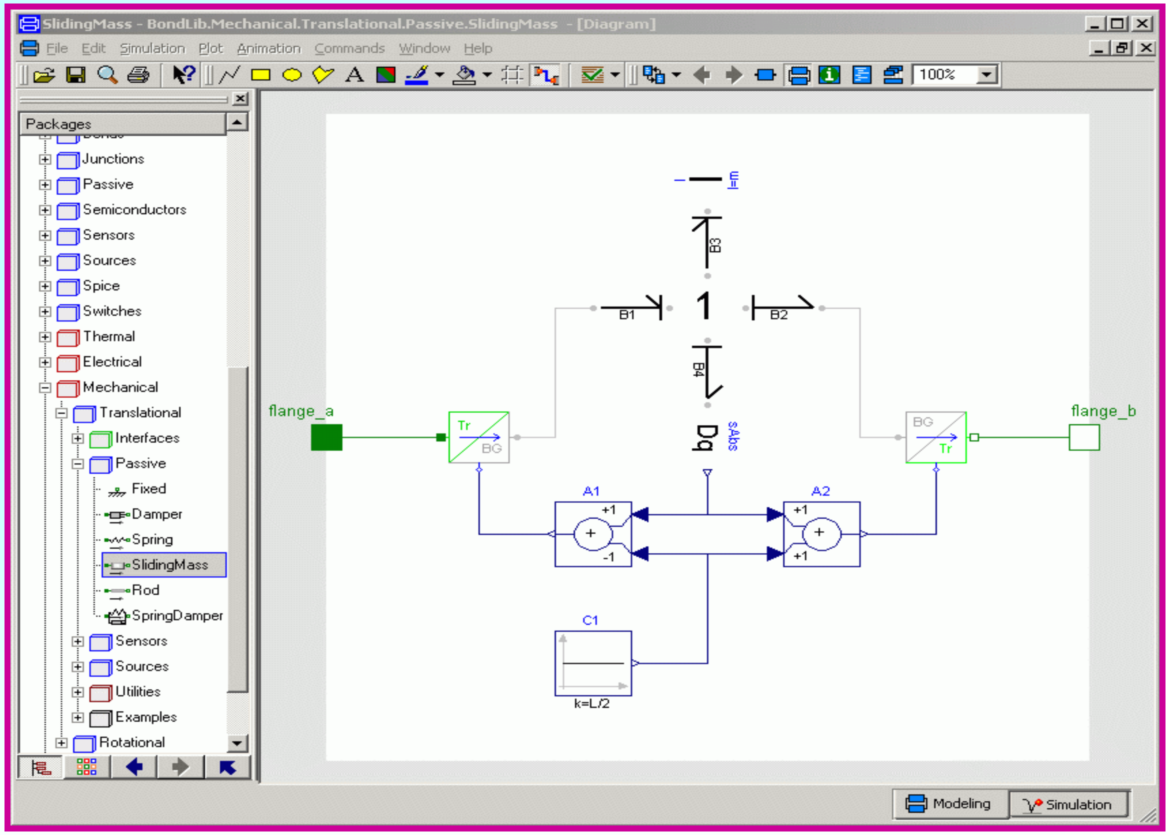
```
model BG2Tr "Bond graph to translational conversion"

equation
  Tr.f = -BondCon1.e;
  Tr.v = BondCon1.f;
  Tr.s = s;
end BG2Tr;
```

- Since the mechanical connector corresponds to a mechanical node, i.e., a point where the sum of all forces (torques) adds up to zero, it corresponds to a bond graph 1-junction, rather than to a bond graph 0-junction. Consequently, it is here the effort variable that must get the negative sign.

The Sliding Mass Model

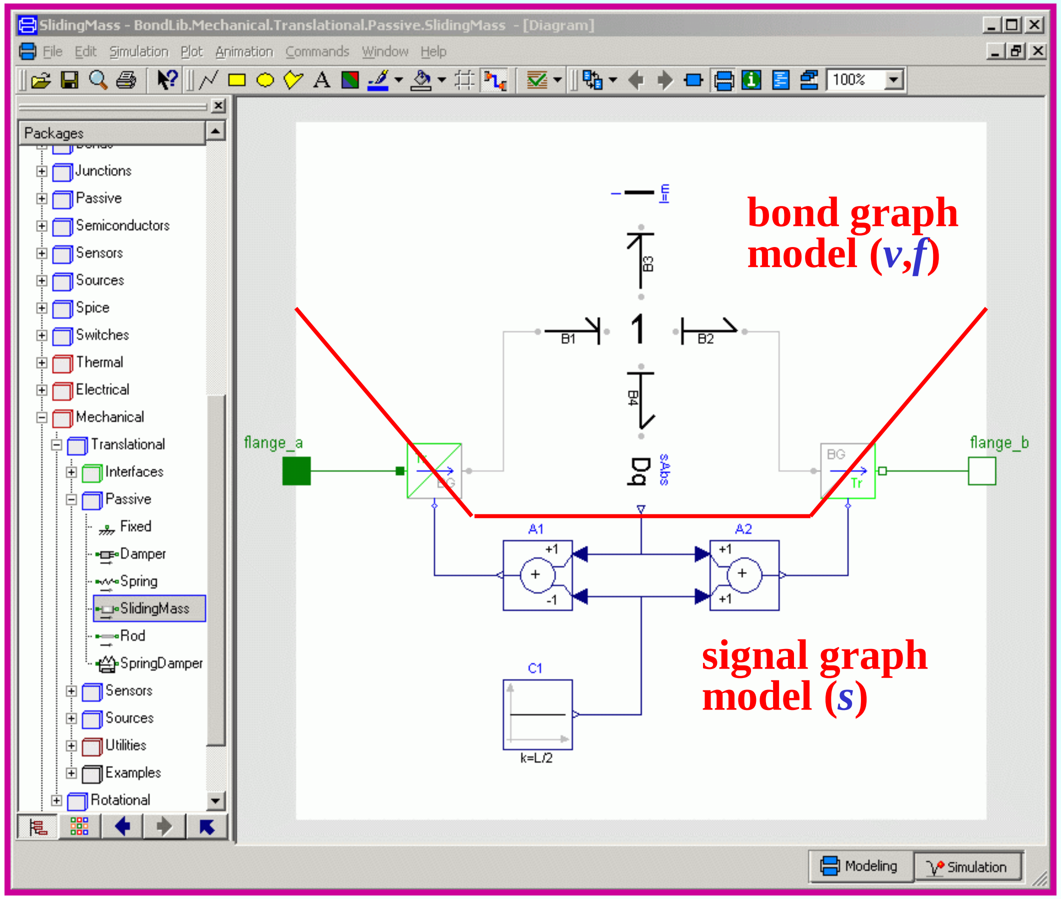
- We are now ready to look at the model of a sliding mass.





The two state variables are the ***f*** variable of the inductor model and the output of the internal integrator of the ***q*** sensor.

The Sliding Mass Model II



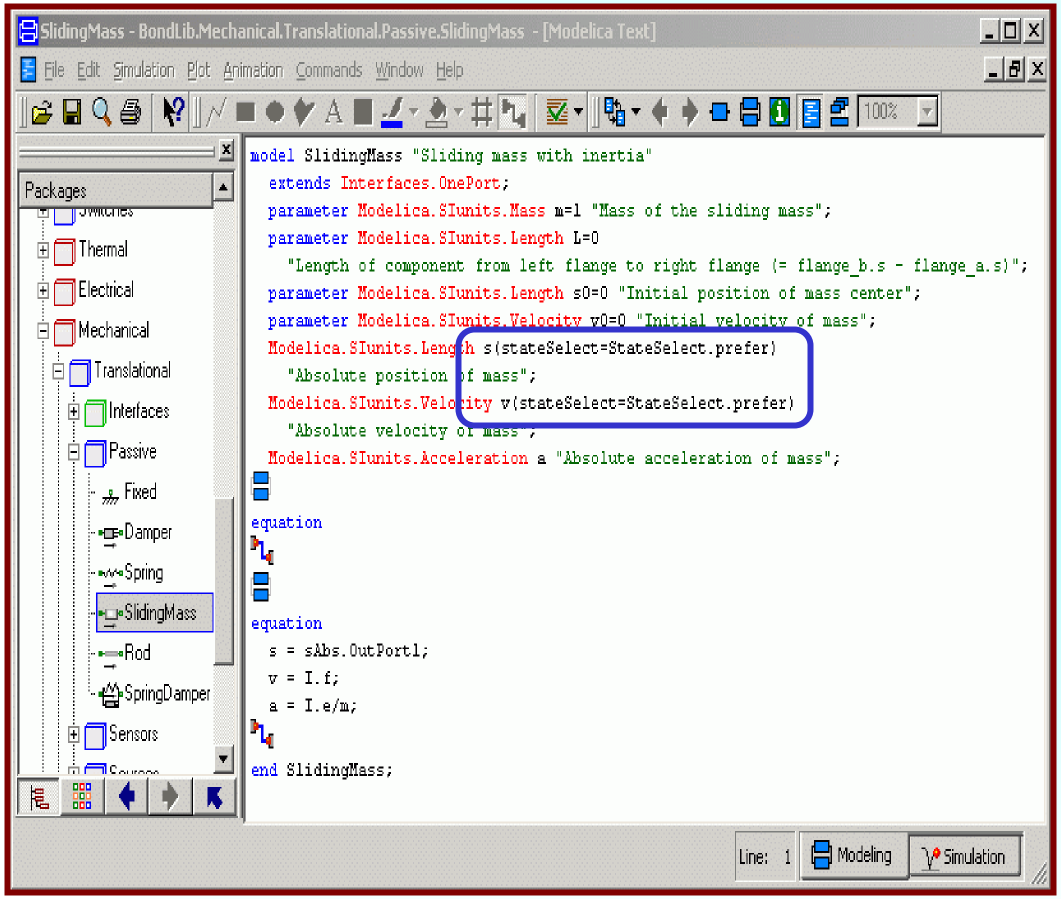
The model is split into an upper (bond graph) part that deals with velocities and forces, and a lower (signal) part that deals with positional information.

The position ***s*** calculated by the sensor is the position of the center of the mass bar.

The position of the left connector is ***L/2*** to the left of the center, and the position of the right connector is ***L/2*** to the right of the center.

These positional values are distributed out to the left and to the right through the mechanical connectors

The Sliding Mass Model III



```
model SlidingMass "Sliding mass with inertia"
  extends Interfaces.OnePort;
  parameter Modelica.SIunits.Mass m=1 "Mass of the sliding mass";
  parameter Modelica.SIunits.Length L=0
    "Length of component from left flange to right flange (= flange_b.s - flange_a.s)";
  parameter Modelica.SIunits.Length s0=0 "Initial position of mass center";
  parameter Modelica.SIunits.Velocity v0=0 "Initial velocity of mass";
  Modelica.SIunits.Length s(stateSelect=StateSelect.prefer)
    "Absolute position of mass";
  Modelica.SIunits.Velocity v(stateSelect=StateSelect.prefer)
    "Absolute velocity of mass";
  Modelica.SIunits.Acceleration a "Absolute acceleration of mass";

equation

equation
  s = sAbs.OutPort1;
  v = I.f;
  a = I.e/m;

end SlidingMass;
```

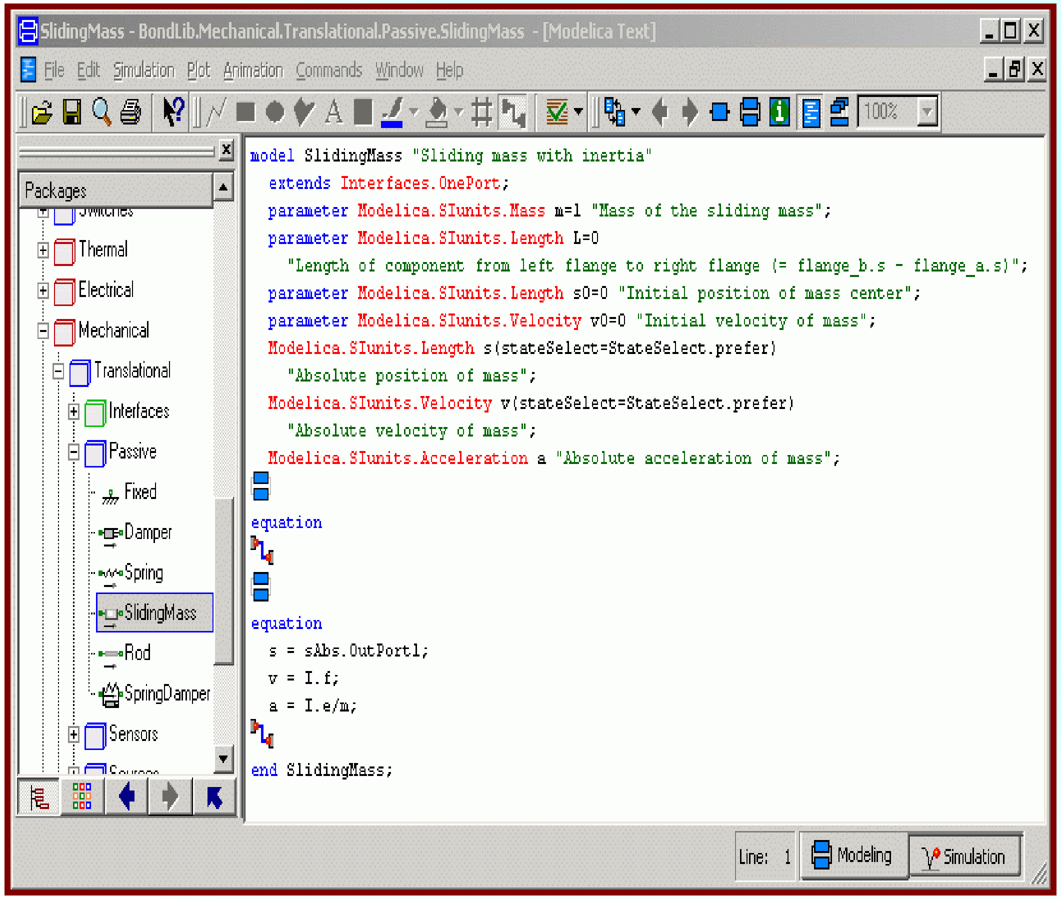
The natural state variables of this model are the internal variables ***I.f*** and ***sAbs.Integrator1.y***. This is inconvenient.

The user of the model would prefer to use the local variables ***v*** and ***s*** of the mass model as state variables.

Dymola can be told to modify the equations such that, if possible, the desired variables are being used as state variables, i.e., show up in the simulation code with a ***der()*** operator.

We shall discuss later in this lecture, how this can actually be accomplished.

The Sliding Mass Model IV



```
model SlidingMass "Sliding mass with inertia"
  extends Interfaces.OnePort;
  parameter Modelica.SIunits.Mass m=1 "Mass of the sliding mass";
  parameter Modelica.SIunits.Length L=0
    "Length of component from left flange to right flange (= flange_b.s - flange_a.s)";
  parameter Modelica.SIunits.Length s0=0 "Initial position of mass center";
  parameter Modelica.SIunits.Velocity v0=0 "Initial velocity of mass";
  Modelica.SIunits.Length s(stateSelect=StateSelect.prefer)
    "Absolute position of mass";
  Modelica.SIunits.Velocity v(stateSelect=StateSelect.prefer)
    "Absolute velocity of mass";
  Modelica.SIunits.Acceleration a "Absolute acceleration of mass";

equation


equation
  s = sAbs.OutPort1;
  v = I.f;
  a = I.e/m;

end SlidingMass;
```

One question that remains is, for which variables we now have to specify the ***initial conditions***.

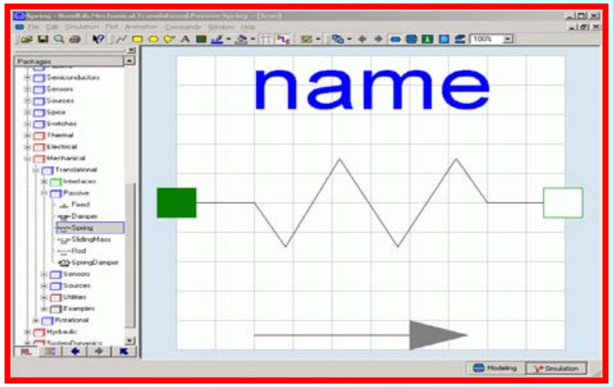
Do we do it for the new state variables ***s*** and ***v***, or do we still do it for the natural state variables?

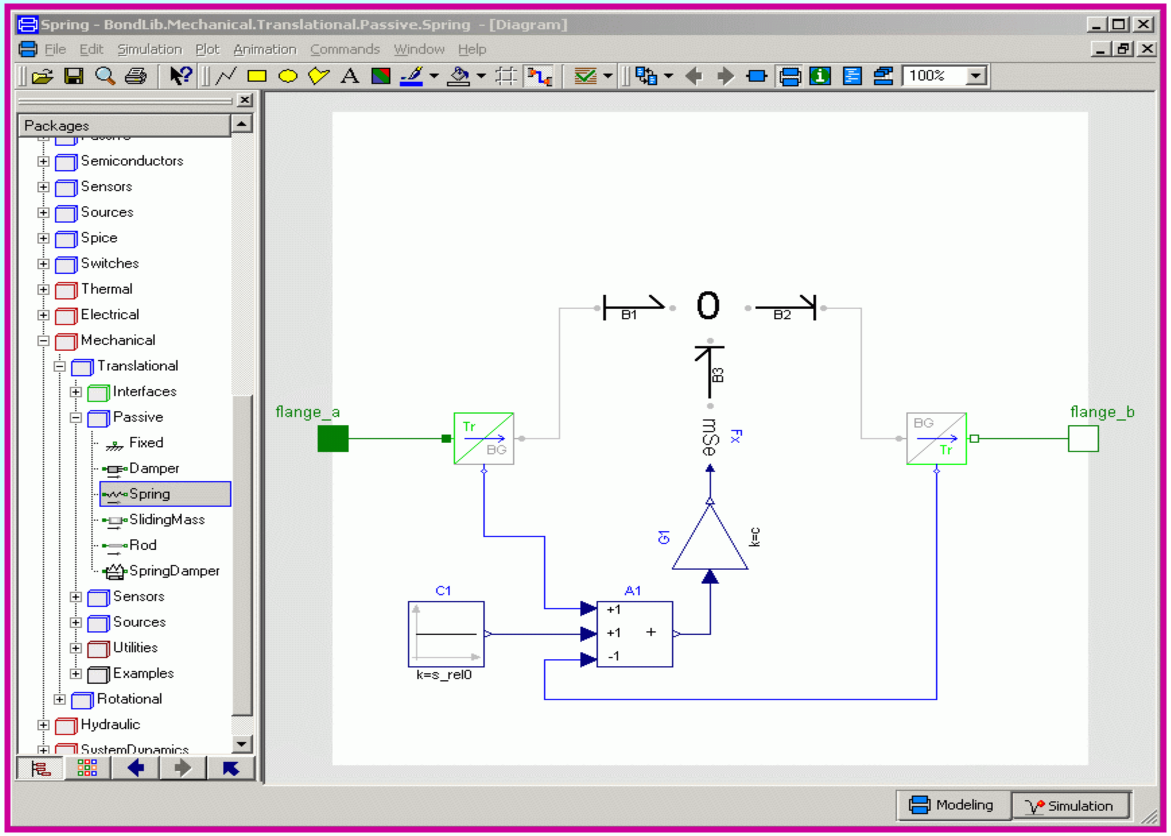
It turns out that we can do either or, but not both.

Dymola will use the specified values as ***start values of an iteration*** and iterate on the unknown initial values of the new state variables, until it finds a set of initial conditions that is consistent with the information that has been specified by the user.

The Spring Model

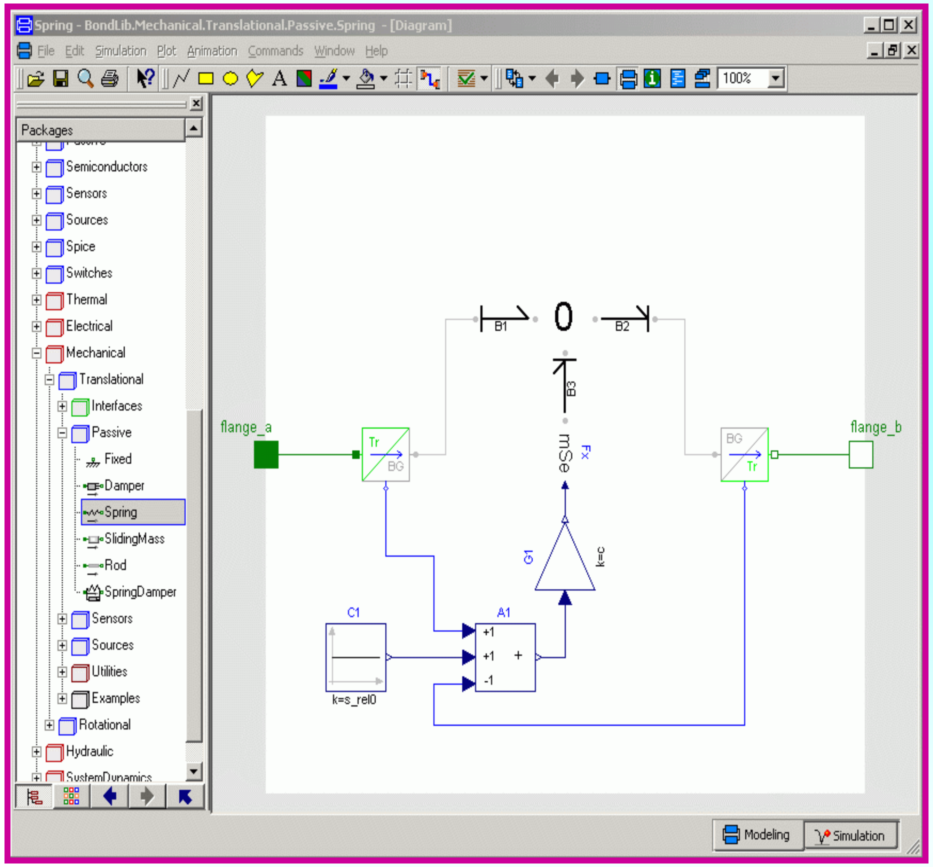
- We are now ready to look at the model of a spring.





The spring is modeled as a modulated effort source, i.e., without an integrator of its own. It imports the positional information through its two terminals.

The Spring Model II



The spring can be either described using the equation:

$$f_x = k \cdot (x_{left} - x_{right})$$

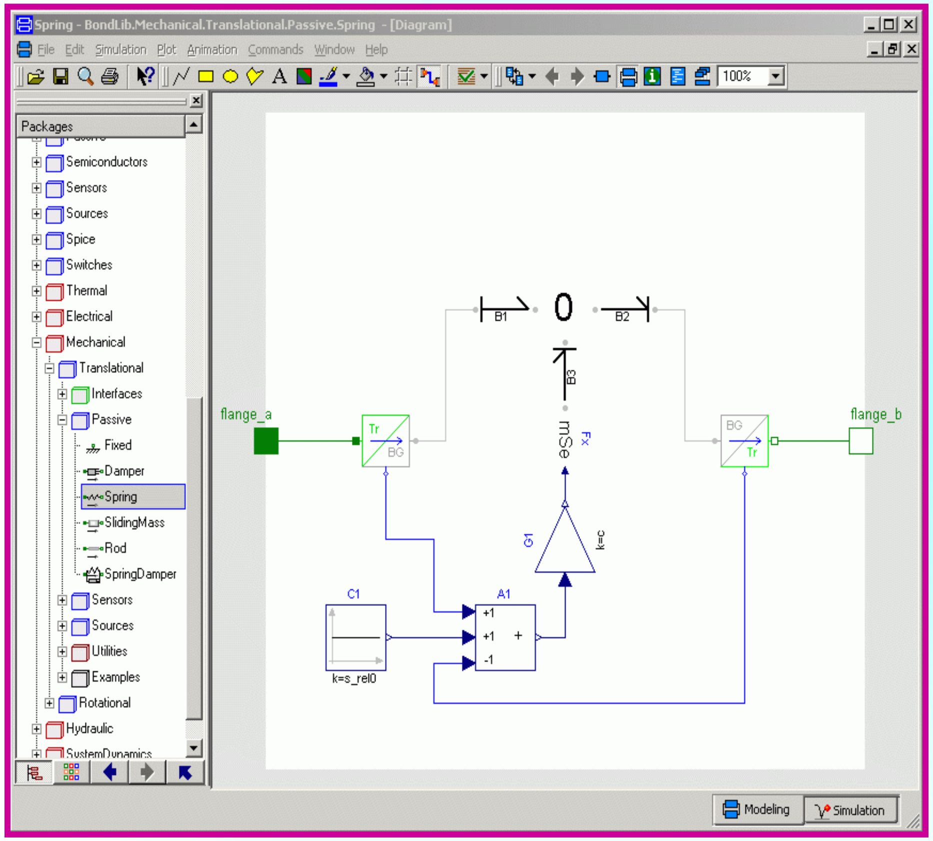
or using the differentiated form:

$$df_x/dt = k \cdot (v_{left} - v_{right})$$

Until now, we always used the latter representation, i.e., a capacitor, whereas now, we are using the former equation.

Both work equally well, in principle.

The Spring Model III



There is however a problem.

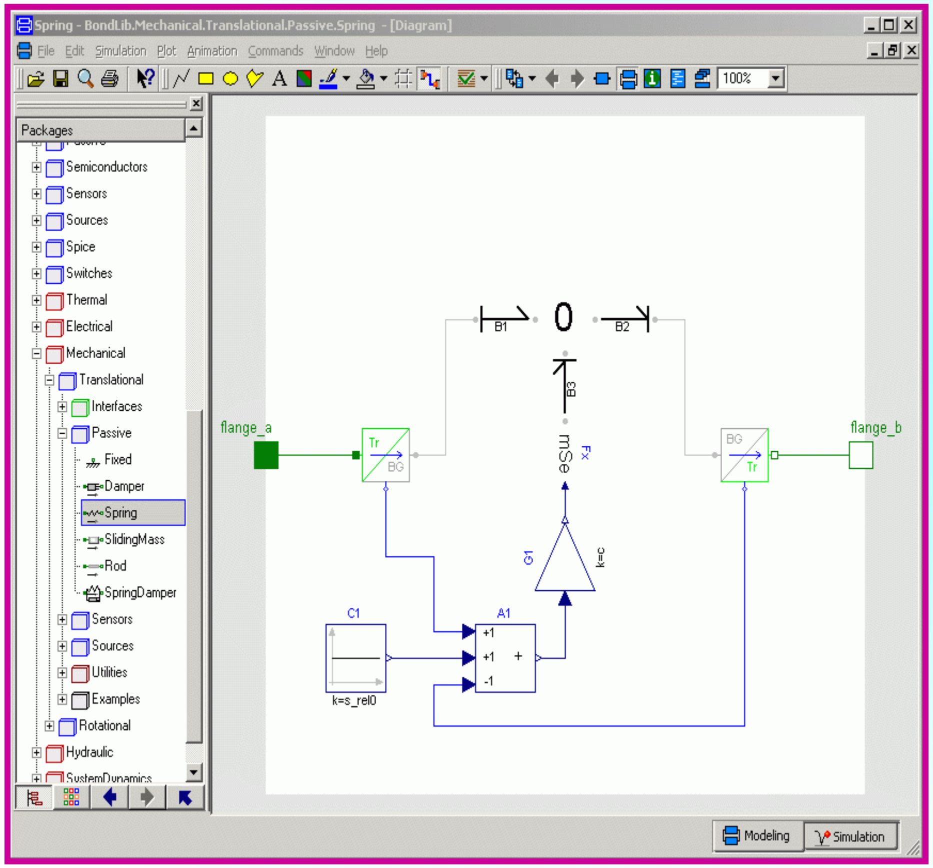
The new spring model imports the needed positional information through its two terminals.

Since positional information is only computed by masses and inertial frames, this spring model can only be placed either between two masses or between a mass and the ground.

In particular, it is not possible to place a spring and a damper in series with each other.

(Placing two springs in series would have worked correctly if we had used a-causal models for signal processing rather than relying on the causal blocks from the block library.)

The Spring Model IV

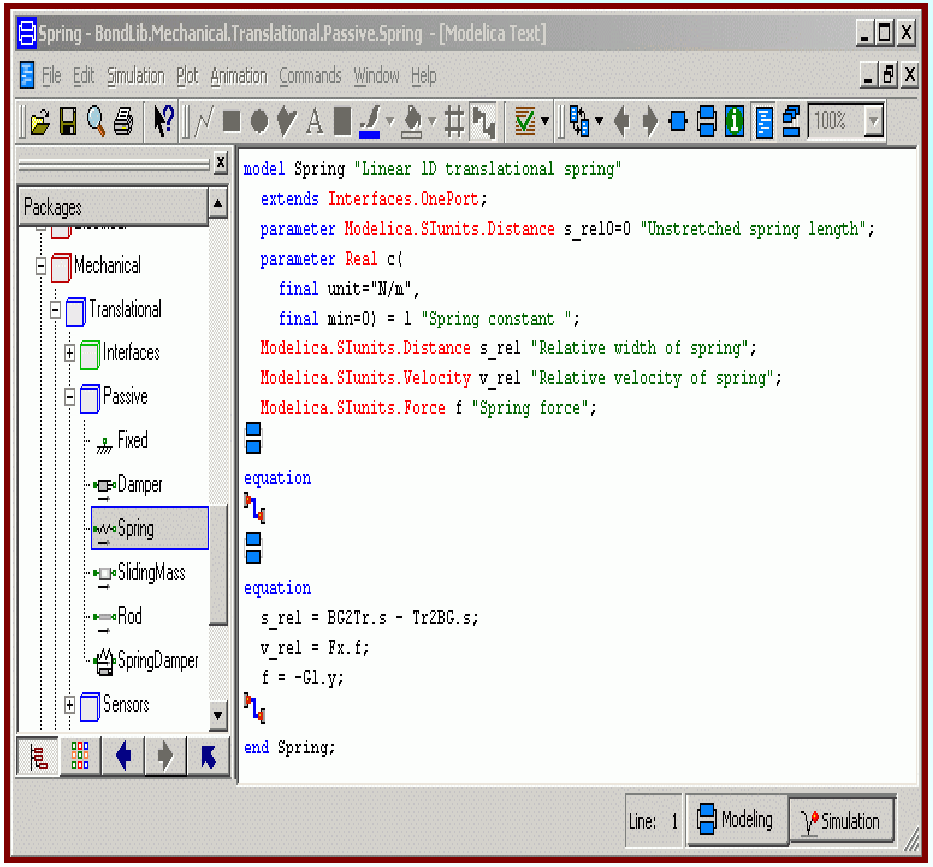


The former model (using a capacitor) did not share this limitation.

Hence our new spring model is a bit of a dirty trick.

… the same dirty trick by the way that the standard library is using, albeit without offering a bond graph interpretation.

The Spring Model V


Spring - BondLib.Mechanical.Translational.Passive.Spring - [Modelica Text]
File Edit Simulation Plot Animation Commands Window Help
100%
Packages
Mechanical
Translational
Interfaces
Passive
Fixed
Damper
Spring
SlidingMass
Rod
SpringDamper
Sensors
Line: 1
Modeling
Simulation

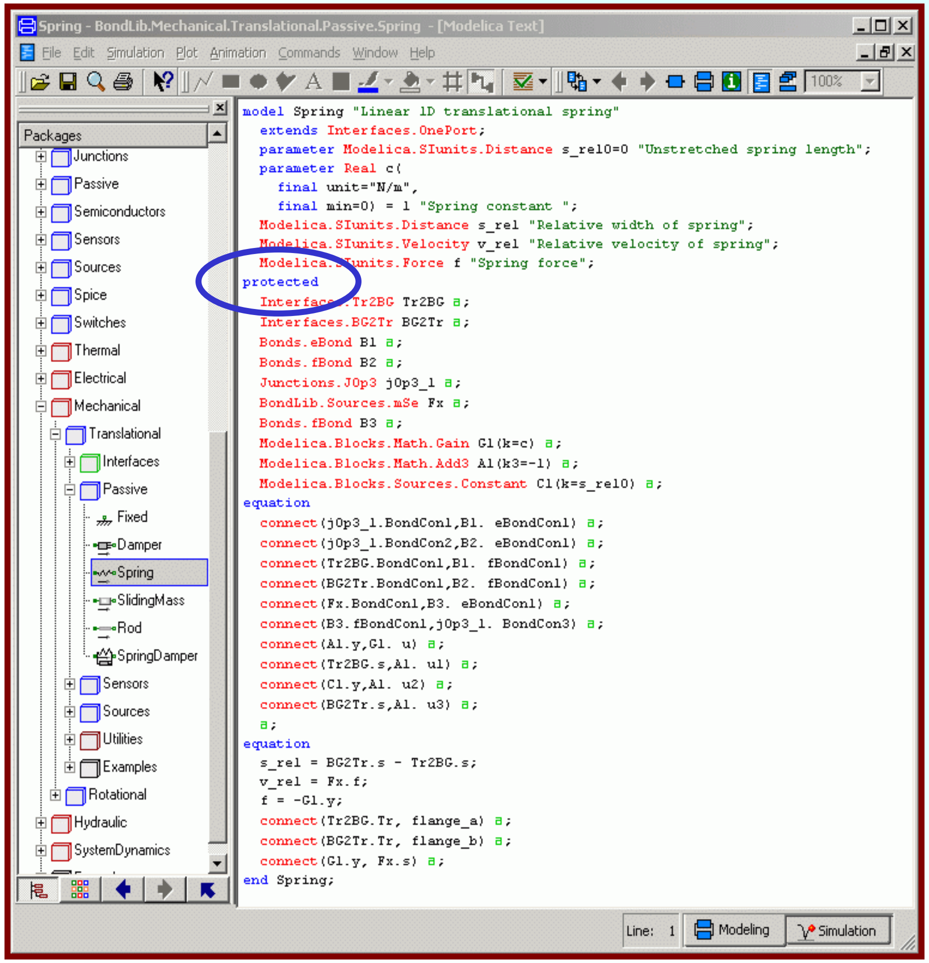
```
model Spring "Linear 1D translational spring"
  extends Interfaces.OnePort;
  parameter Modelica.SIunits.Distance s_rel0=0 "Unstretched spring length";
  parameter Real c(
    final unit="N/m",
    final min=0) = 1 "Spring constant ";
  Modelica.SIunits.Distance s_rel "Relative width of spring";
  Modelica.SIunits.Velocity v_rel "Relative velocity of spring";
  Modelica.SIunits.Force f "Spring force";

equation

equation
  s_rel = BG2Tr.s - Tr2BG.s;
  v_rel = Fx.f;
  f = -G1.y;

end Spring;
```

The equation layer doesn't offer any surprises.

The relative position and velocity of the spring are calculated here, since these are variables that the user may like to display.

The remaining models of the library are what we would expect them to be, thus they don't need to be discussed here.

Wrapping Tightly – Protected Variables



```
model Spring "Linear 1D translational spring"
  extends Interfaces.OnePort;
  parameter Modelica.SIunits.Distance s_rel0=0 "Unstretched spring length";
  parameter Real c(
    final unit="N/m",
    final min=0) = 1 "Spring constant ";
  Modelica.SIunits.Distance s_rel "Relative width of spring";
  Modelica.SIunits.Velocity v_rel "Relative velocity of spring";
  Modelica.SIunits.Force f "Spring force";
protected
  Interfaces.Tr2BG Tr2BG a;
  Interfaces.BG2Tr BG2Tr a;
  Bonds.eBond B1 a;
  Bonds.fBond B2 a;
  Junctions.J0p3 j0p3_1 a;
  BondLib.Sources.mSe Fx a;
  Bonds.fBond B3 a;
  Modelica.Blocks.Math.Gain G1(k=c) a;
  Modelica.Blocks.Math.Add3 A1(k3=-1) a;
  Modelica.Blocks.Sources.Constant C1(k=s_rel0) a;
equation
  connect(j0p3_1.BondCon1,B1. eBondCon1) a;
  connect(j0p3_1.BondCon2,B2. eBondCon1) a;
  connect(Tr2BG.BondCon1,B1. fBondCon1) a;
  connect(BG2Tr.BondCon1,B2. fBondCon1) a;
  connect(Fx.BondCon1,B3. eBondCon1) a;
  connect(B3.fBondCon1,j0p3_1. BondCon3) a;
  connect(A1.y,G1. u) a;
  connect(Tr2BG.s,A1. u1) a;
  connect(C1.y,A1. u2) a;
  connect(BG2Tr.s,A1. u3) a;
  a;
equation
  s_rel = BG2Tr.s - Tr2BG.s;
  v_rel = Fx.f;
  f = -G1.y;
  connect(Tr2BG.Tr, flange_a) a;
  connect(BG2Tr.Tr, flange_b) a;
  connect(G1.y, Fx.s) a;
end Spring;
```

Let’s now look at the ***expanded view*** of the equation layer of the spring model.

I manually placed the keyword ***protected*** in front of the declarations of variables to the inside of the wrapper models.

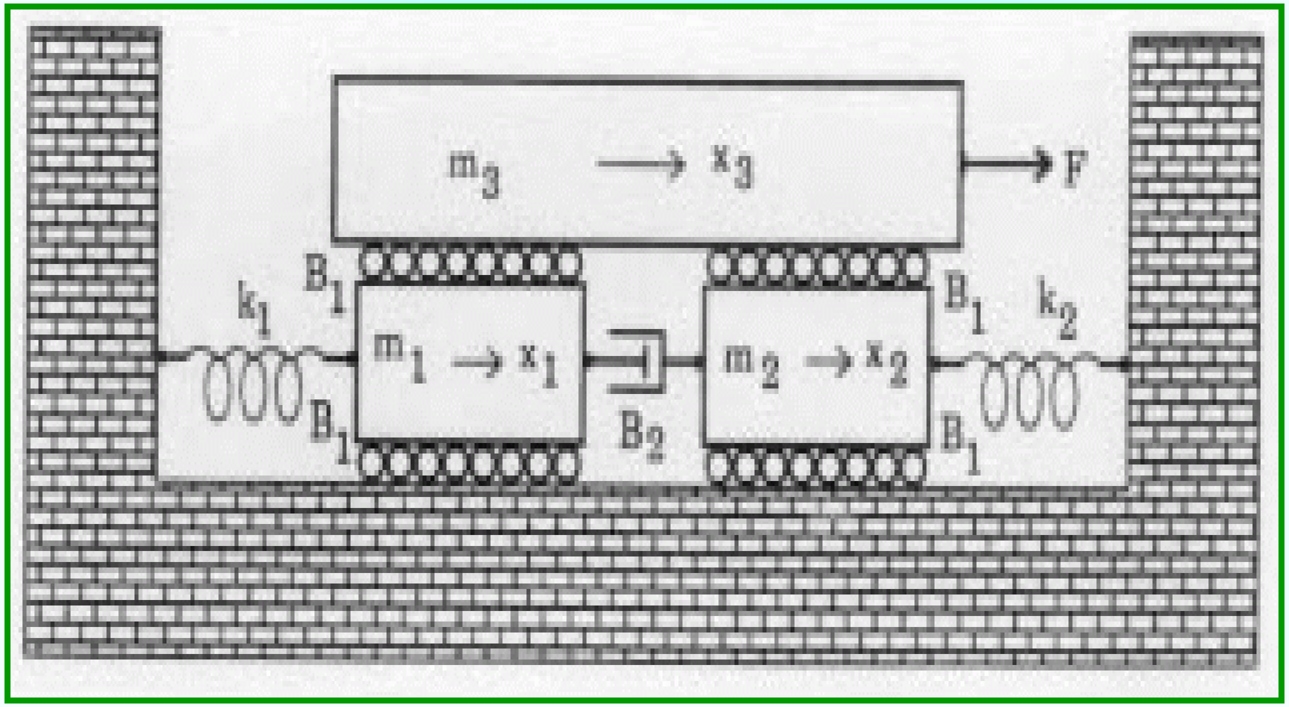
The effect of this measure is to prevent these variables from being displayed in the ***simulation window***.

In this way, the model parameters will look exactly the same in the simulation window as using the corresponding model of the standard library.

The model has been ***wrapped tightly***.

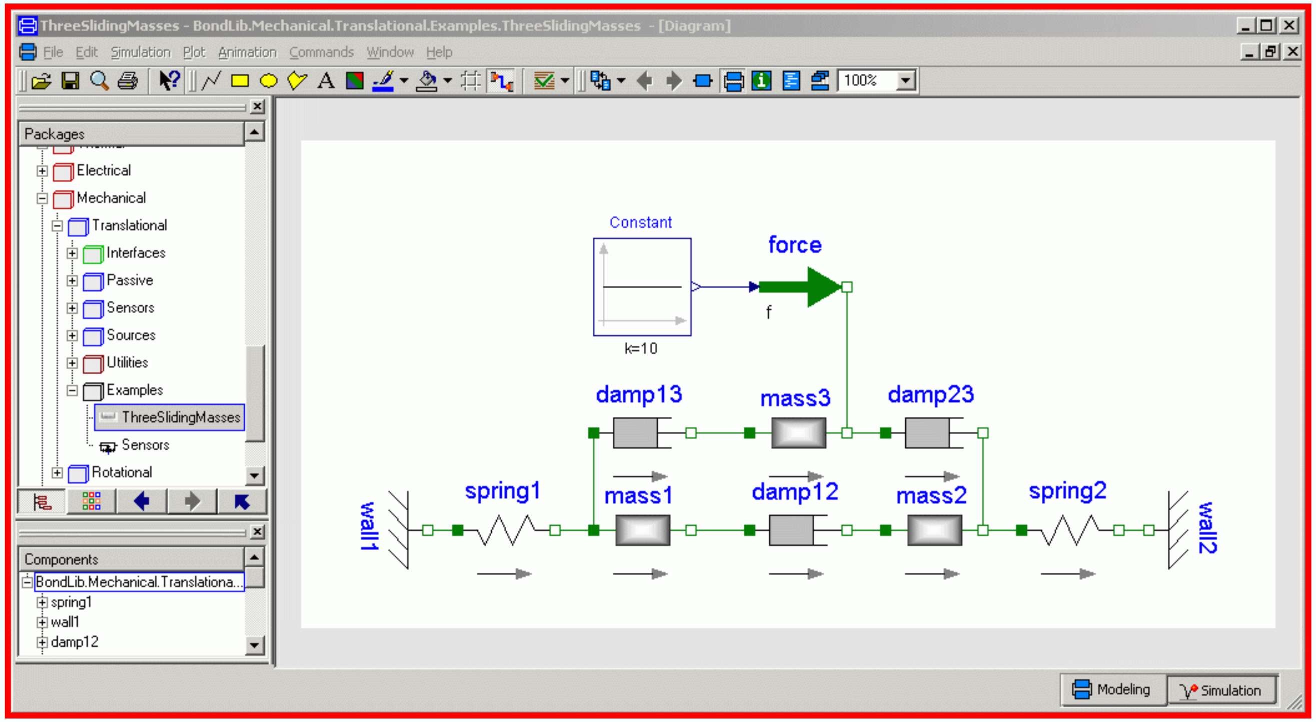
An Example

- Given the following mechanical system:

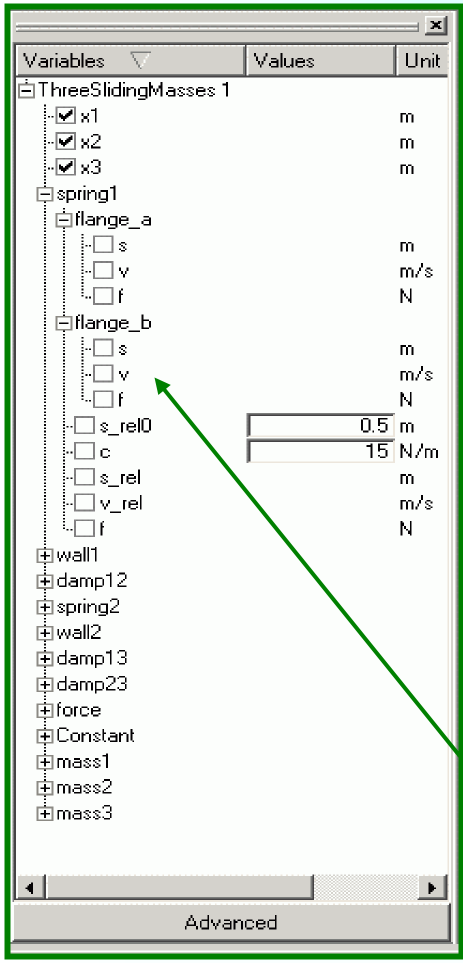


An Example II

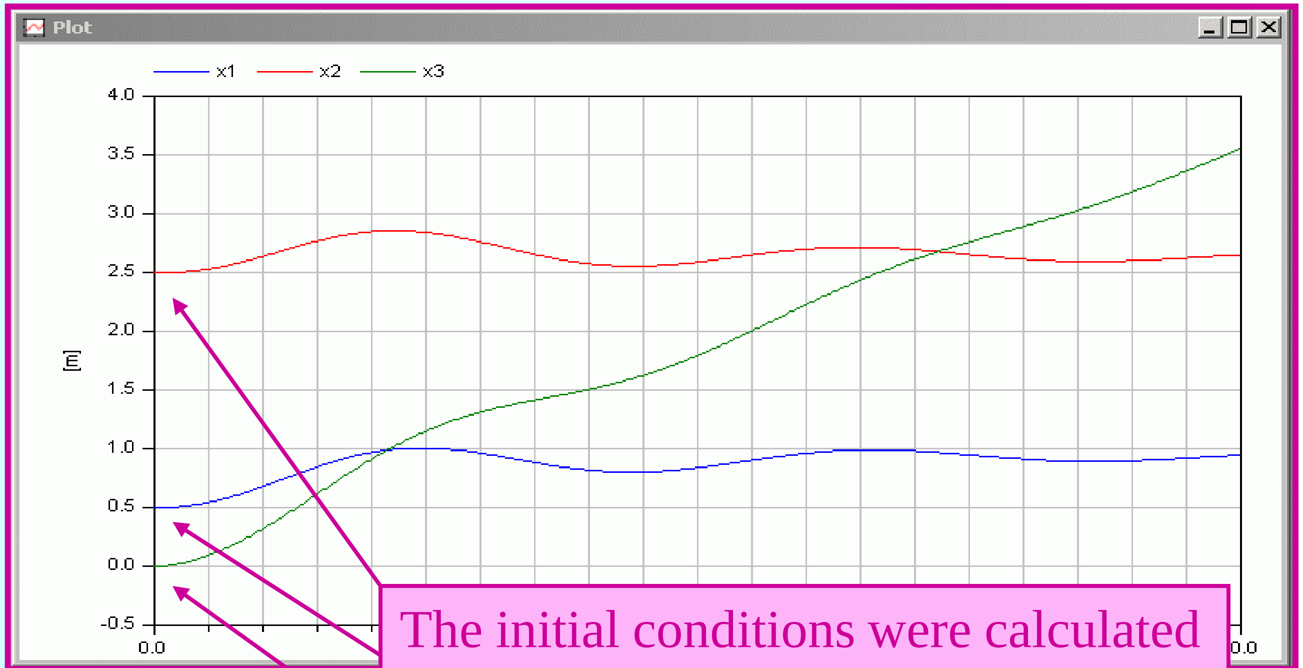
- Using the translational sub-library of the mechanics library of the ***BondLib*** library, this system can be modeled as follows:



An Example III



```
Variables        Values   Unit
⊟ThreeSlidingMasses 1
   ☑ x1                   m
   ☑ x2                   m
   ☑ x3                   m
  ⊟spring1
    ⊟flange_a
       ☐ s                m
       ☐ v                m/s
       ☐ f                N
    ⊟flange_b
       ☐ s                m
       ☐ v                m/s
       ☐ f                N
     ☐ s_rel0      0.5    m
     ☐ c            15    N/m
     ☐ s_rel              m
     ☐ v_rel              m/s
     ☐ f                  N
  ⊞wall1
  ⊞damp12
  ⊞spring2
  ⊞wall2
  ⊞damp13
  ⊞damp23
  ⊞force
  ⊞Constant
  ⊞mass1
  ⊞mass2
  ⊞mass3
            Advanced
```



The initial conditions were calculated correctly.

Due to information hiding (protected variables), only those variables external to the wrapper models are visible.

The Selection of State Variables

- Until now, we have always used the "capacitor" voltages and "inductor" currents of our bond graph models as our state variables.
- Sometimes, this is not desirable. We may have specific wishes as to which variables should be used as state variables.
- In some cases (as we shall see later), the choice of state variables also influences the run-time efficiency of the generated simulation code.
- The number of generated equations may depend heavily on a wise choice of state variables.

The Selection of State Variables II

- ***Dymola*** supports the concept of selecting state variables differently from those that the system would normally choose.
- To this end, the user declares a desired state variable as follows:

```
Real x(stateSelect = StateSelect.prefer)
```

- ***Dymola*** complies with the request by means of a variant of the ***Pantelides algorithm***.

The Selection of State Variables III

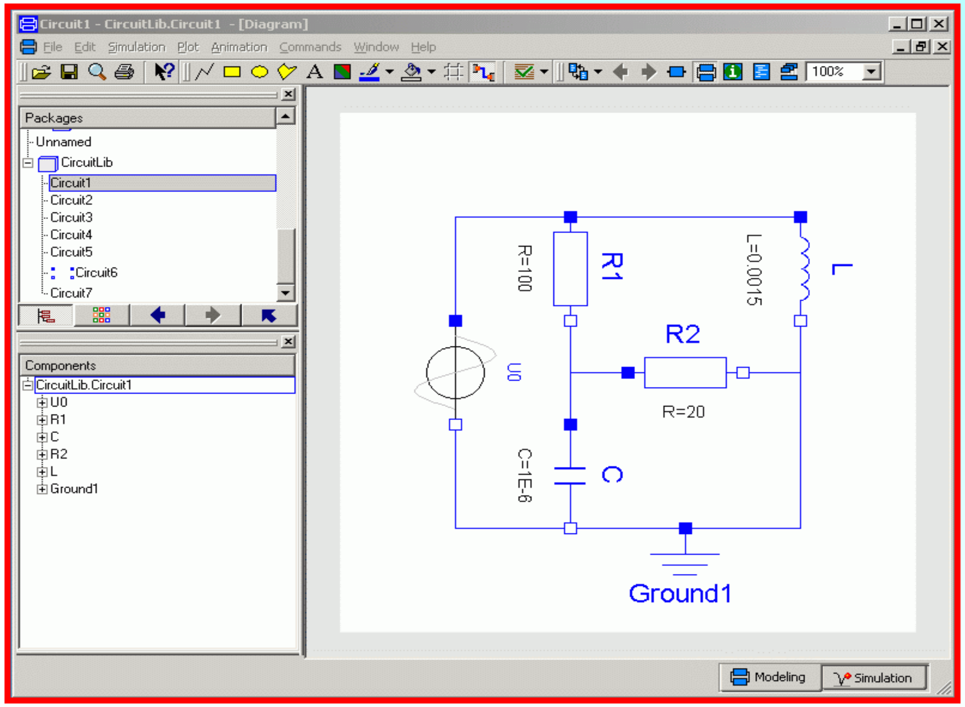
- If the desired state variable already appears in differentiated form, use it whenever possible as a state.
- If the desired state variable does not already appear in differentiated form, differentiate the equation that computes the desired state, add it to the set of equations, and create a new integrator for it.
- We now have one equation too many.
- If in the process of differentiation additional algebraic variables are being differentiated, differentiate the equations defining those variables as well, add them also to the set of equations, but don't add new integrators for them.

The Selection of State Variables IV

- In the process of these additional differentiations, new variables and new equations are added to the set, so that at the end of the process, there is still one equation too many.
- If the desired state variable is legitimate, at least one of the previous state variables occurs among the set of equations that were differentiated.
- Throw the integrator associated with one of those state variables away to once again end up with an identical number of equations and unknowns.

The Selection of State Variables V

- Given our “standard” electrical circuit.
- We have already learnt how to retrieve a causal set of equations from it.



$\Rightarrow$

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C/dt = i_C / C_1$

The Selection of State Variables VI

- We wish to derive a different set of simulation equations that uses u_{R1} as a state variable, while eliminating one of the two former state variables from the set.
- To this end, we manually implement the state selection algorithm described earlier.

The Selection of State Variables VII

Continue with this equation.

Now differentiate these two equations.

Differentiate this equation, while creating a new derivative for u_{R1}, but no new derivatives for v_1 and v_2.

Eliminate this integrator.

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C \cancel{/dt} = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

The Selection of State Variables VIII

- We can now apply the Tarjan algorithm to the new set of equations.

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

$\Rightarrow$

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

The Selection of State Variables IX

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

$\Rightarrow$

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

The Selection of State Variables X

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

$\Rightarrow$

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

The Selection of State Variables XI

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

$\Rightarrow$

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

The Selection of State Variables XII

1: $U_0 = f(t)$
2: $v_2 = u_C$
3: $u_{R2} = u_C$
4: $u_L = U_0$
5: $v_1 = U_0$
6: $i_{R2} = u_{R2} / R_2$
7: $u_{R1} = v_1 - v_2$
8: $i_{R1} = u_{R1} / R_1$
9: $i_C = i_{R1} - i_{R2}$
10: $i_0 = i_L + i_{R1}$
11: $di_L/dt = u_L / L_1$
12: $du_C = i_C / C_1$

$du_{R1}/dt = dv_1 - dv_2$
$dv_1 = dU_0$
$dv_2 = du_C$
$dU_0 = df(t)/dt$

- We ended up with 16 equations in 16 unknowns instead of the former 12 equations in 12 unknowns.
- This solution is a bit less run-time efficient.
- However, the variables that now appear differentiated in the model are the inductor current, i_L, and the resistive voltage, u_{R1}.