The Dymola Bond Graph Library

- In this class, we shall deal with some issues relating to the construction of the ***Dymola Bond Graph Library***.
- The design principles are explained, and some further features of the ***Dymola*** modeling framework are shown.
- We shall introduce the concept of model wrapping as implemented in the bond graph library.
- An example of an electronic circuit simulation completes the presentation.

Table of Contents

Across and Through Variables

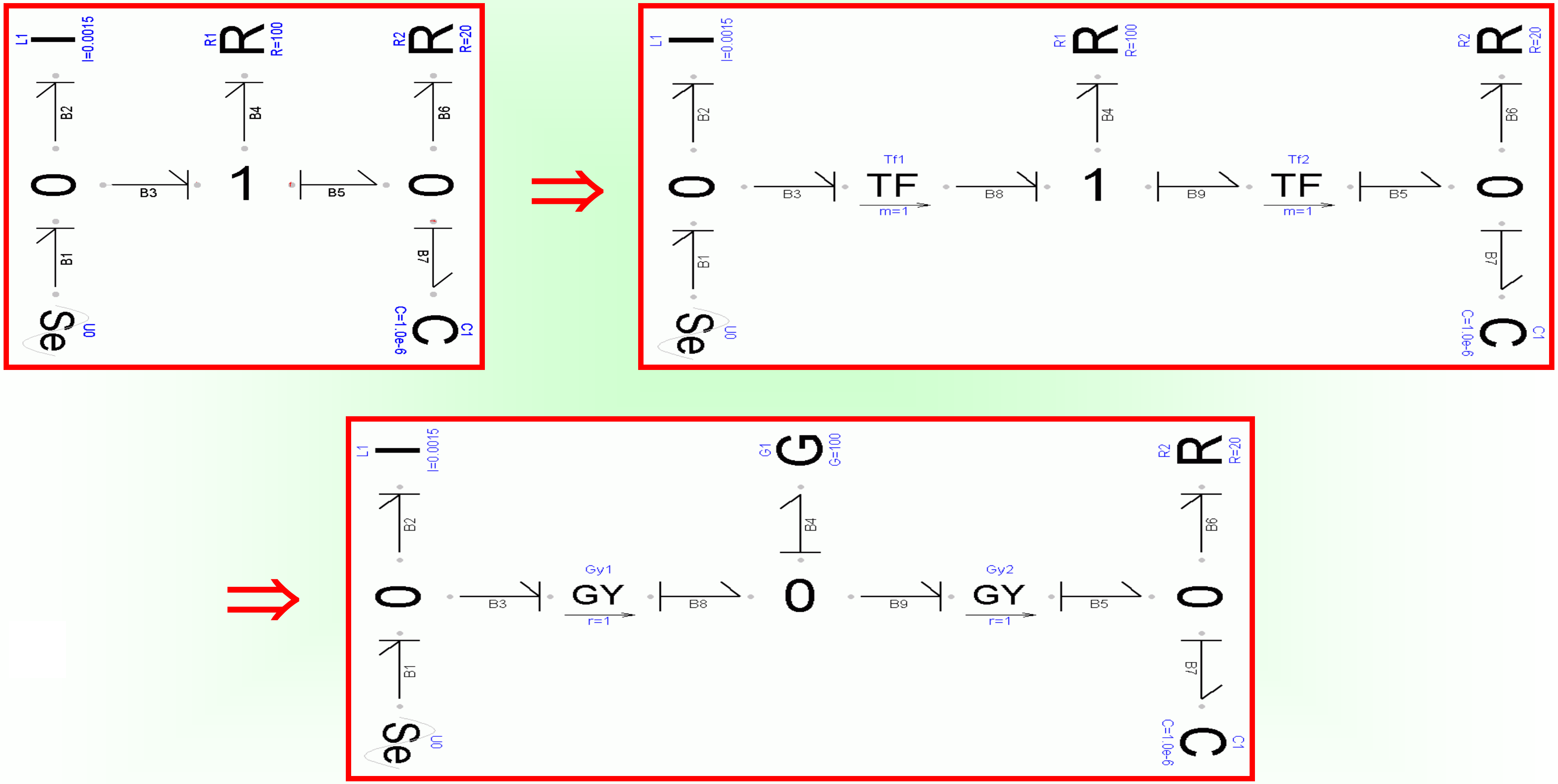
- ***Dymola*** offers two types of variables, the ***across variables*** and the ***through variables***.
- In a ***Dymola node***, across variables are set equal across all connections to the node, whereas through variables add up to zero.
- Consequently, if we equate ***across variables*** with ***efforts***, and ***through variables*** with ***flows***, ***Dymola nodes*** correspond exactly to the ***0-junctions*** of our bond graphs.

Gyro-bonds

- In my modeling book, I exploited this similarity by implementing the ***bonds*** as ***twisted wires*** (as ***null-modems***).
- By requesting furthermore that:
  - ♣ **0- and 1-junctions must always toggle. No two junctions of the same gender may be connected by a bond.**
  - ♣ **All elements must always be attached to 0-junctions, never to 1-junctions.**

both the ***0-junctions*** and the ***1-junctions*** can be implemented as ***Dymola nodes***.

Gyro-bonds II

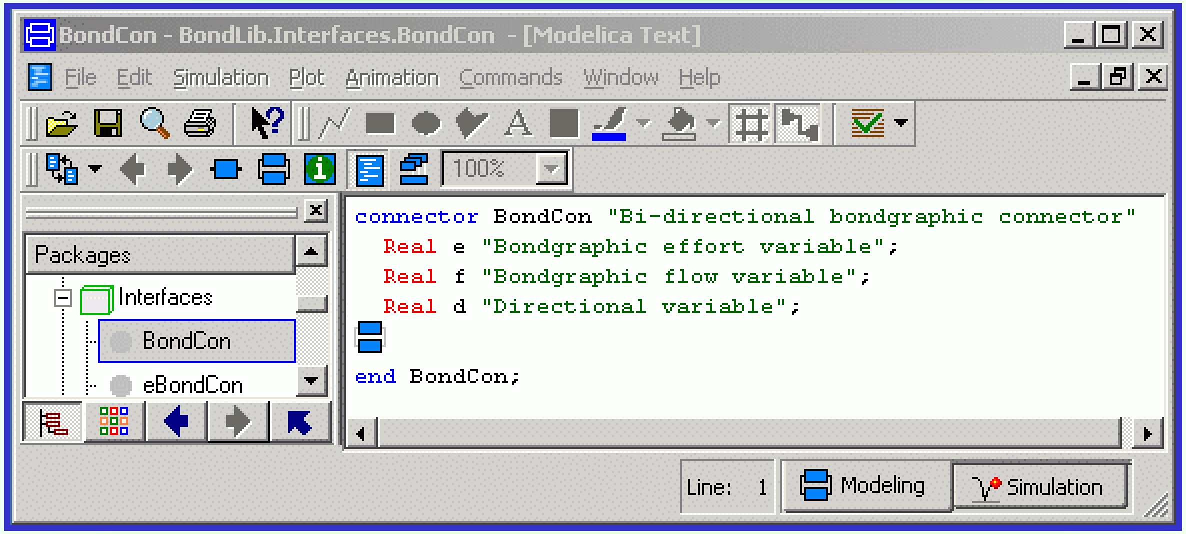


Graphical Bond Graph Modeling I

- For graphical bond-graph modeling, these additional rules may, however, be too constraining.
- For example, thermal systems often exhibit 0-junctions with many bonds attached. It must be possible to split these 0-junctions into a series of separate 0-junctions connected by bonds, so that the number of bonds attached at any one junction can be kept sufficiently small.

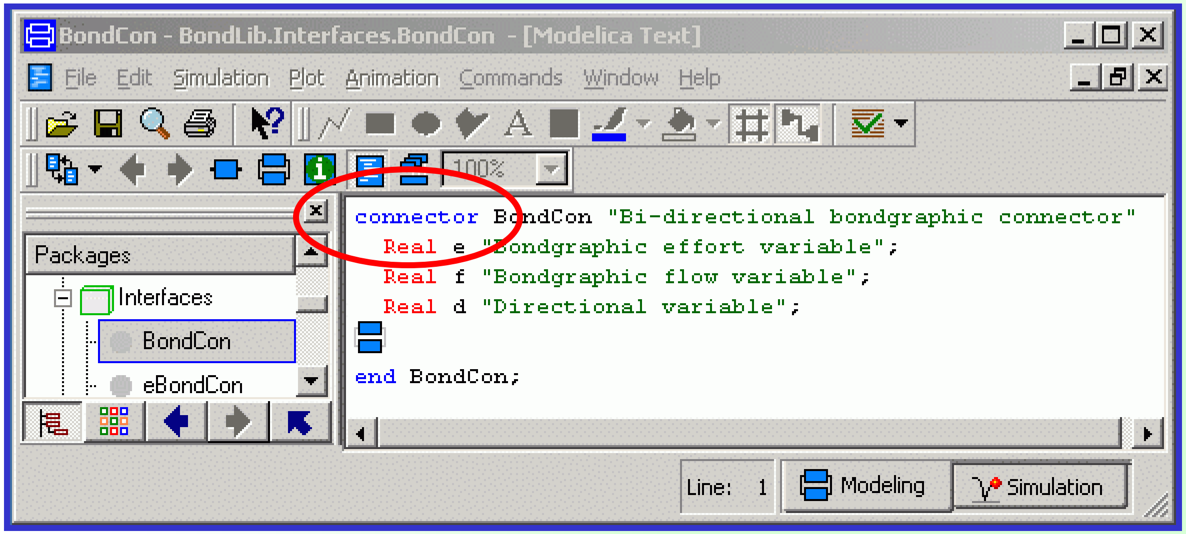
Graphical Bond Graph Modeling II

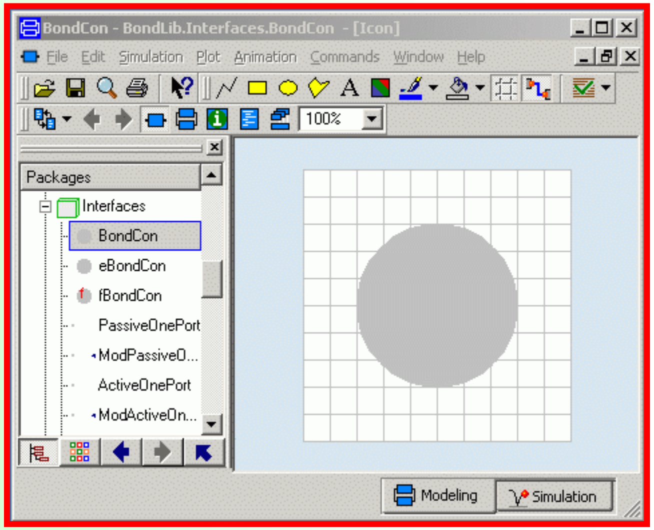
- For this reason, the graphical bond graph modeling of ***Dymola*** defines both ***efforts*** and ***flows*** as ***across variables***.



- Consequently, the ***junctions*** will have to be programmed explicitly. They can no longer be implemented as ***Dymola nodes***.

The Bond Graph Connectors I



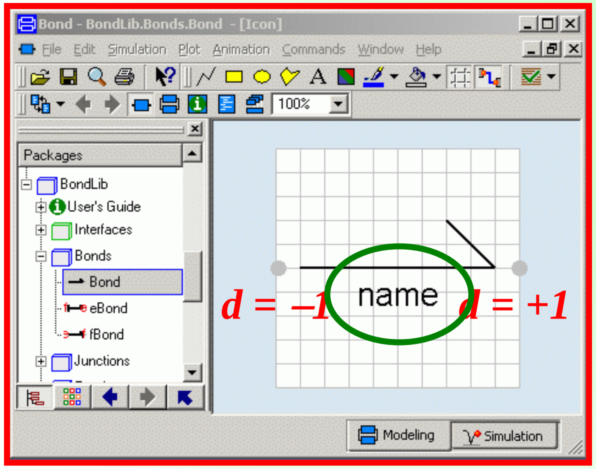


Equation window *Icon window*

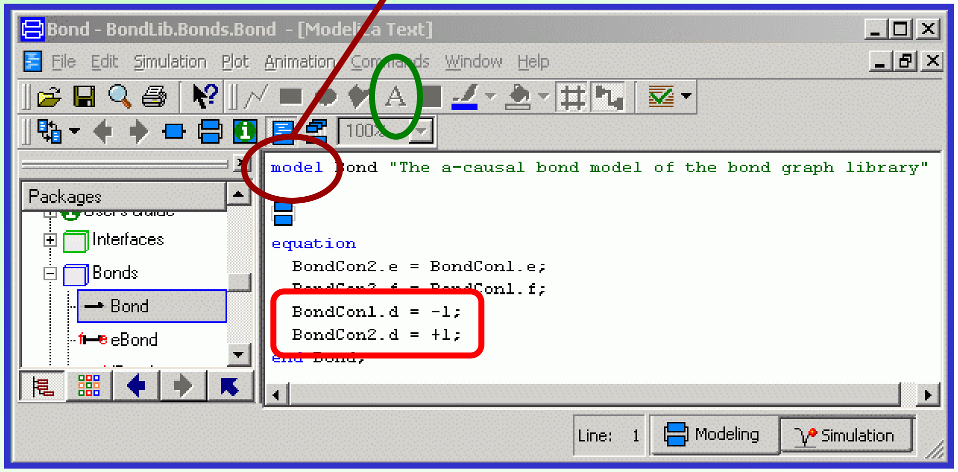
- The directional variable, ***d***, is a third across variable made available as part of the ***bond-graph connector***, which is depicted as a ***grey dot***.

The A-Causal Bond "Model"

- The model of a bond can now be constructed by dragging two of the bond-graph connectors into the diagram window. They are named ***BondCon1*** and ***BondCon2***.



Icon window

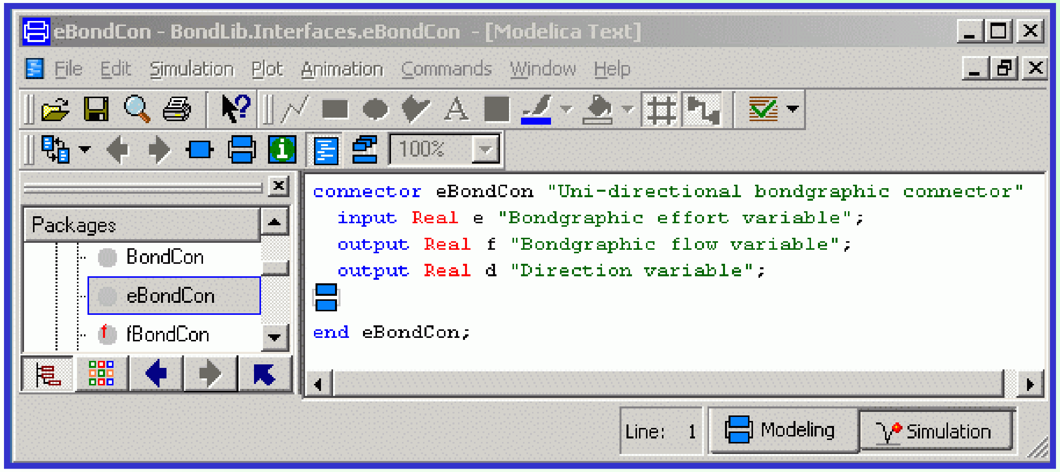


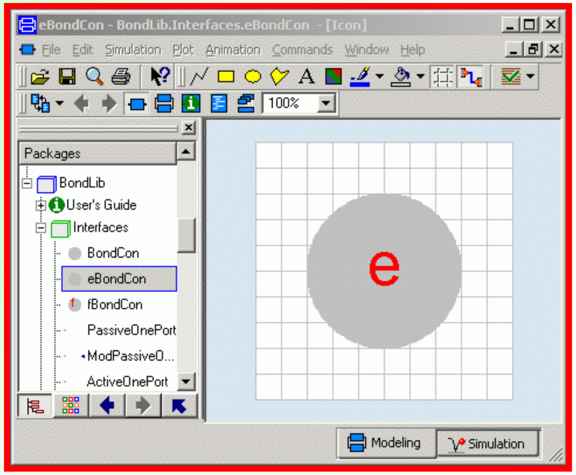
Equation window

Place the text "***%name***" in the icon window to get the name of the model displayed upon invocation.

The Bond Graph Connectors II

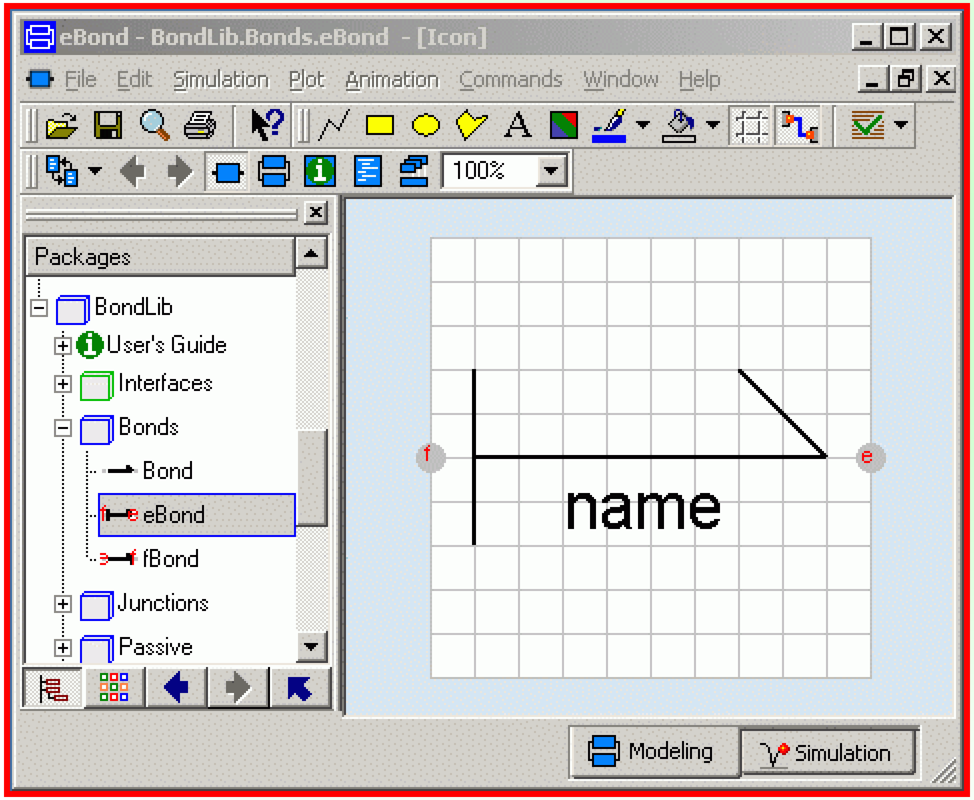
- ***Dymola*** variables are usually a-causal. However, they can be made causal by declaring them explicitly in a causal form.
- Two additional bond-graph connectors have been defined. The ***e-connector*** treats the ***effort*** as an ***input***, and the ***flow*** as an ***output***.





- The ***f-connector*** treats the ***flow*** as ***input*** and the ***effort*** as ***output***.

The Causal Bond "Blocks"



Using these connectors, causal bond blocks can be defined.

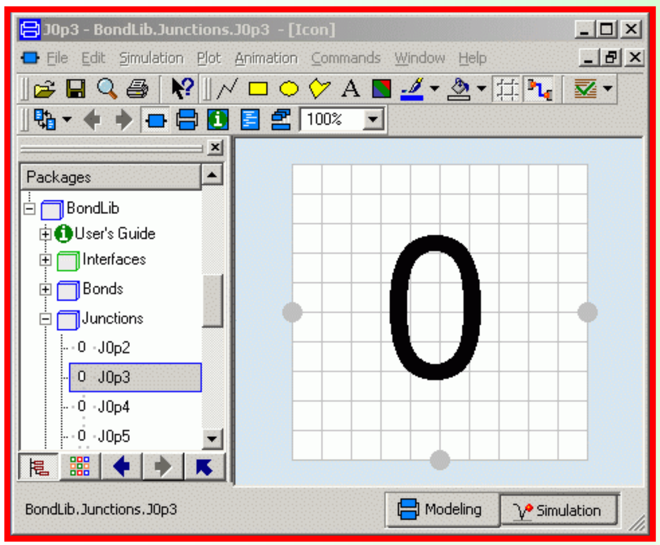
The ***f-connector*** is used at the side of the causality stroke.

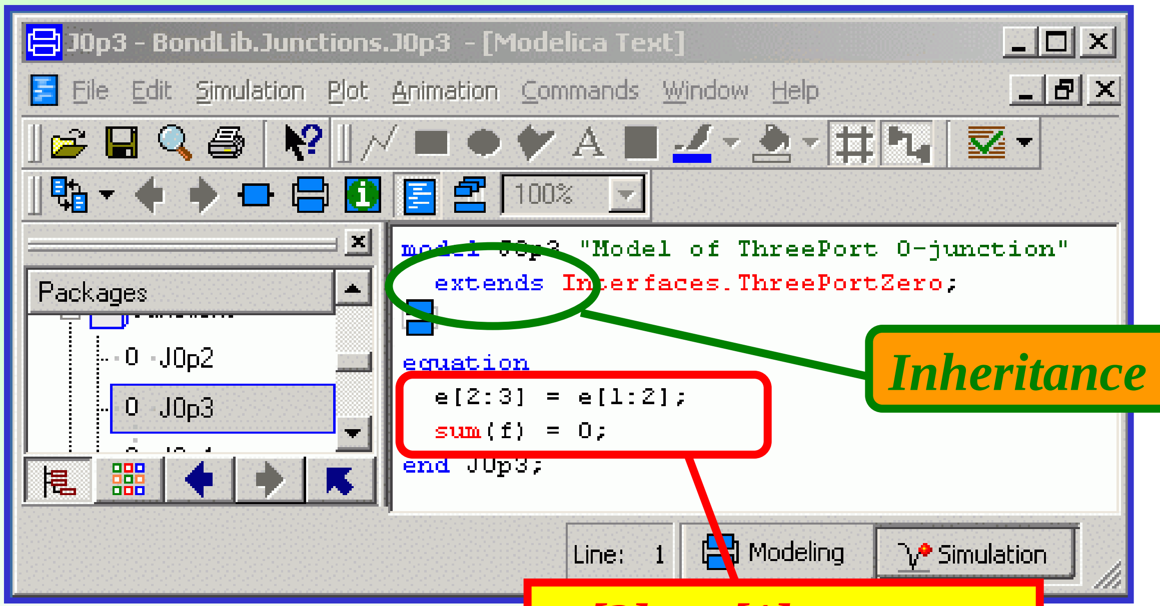
The ***e-connector*** is used at the other side.

The causal connectors are only used in the context of the bond blocks. Everywhere else, the normal bond-graph connectors are to be used.

The Junctions I

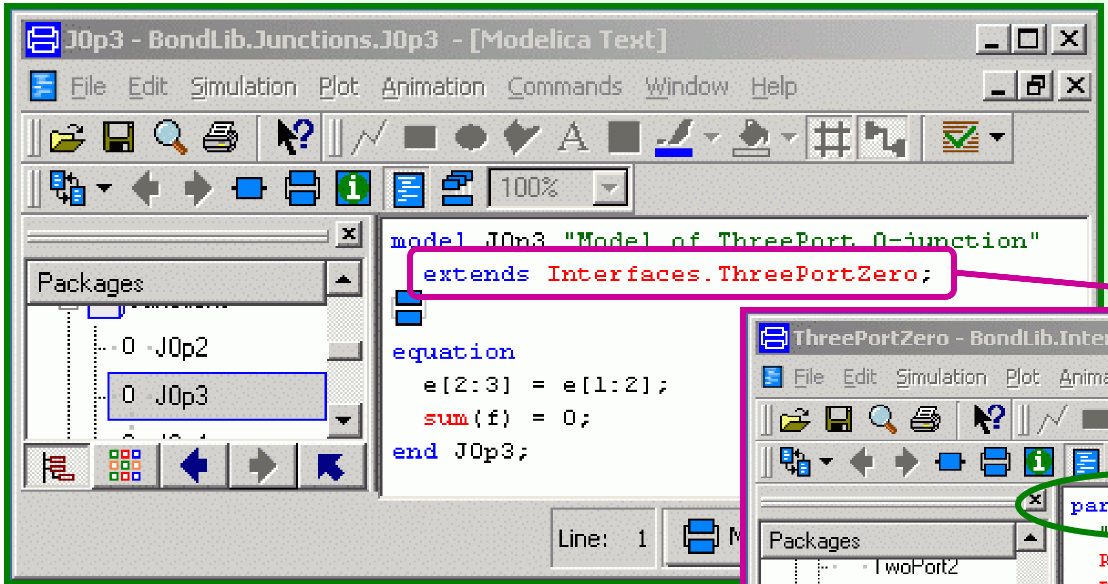
- The junctions can now be programmed. Let us look at a ***0-junction with three bond attachments***.

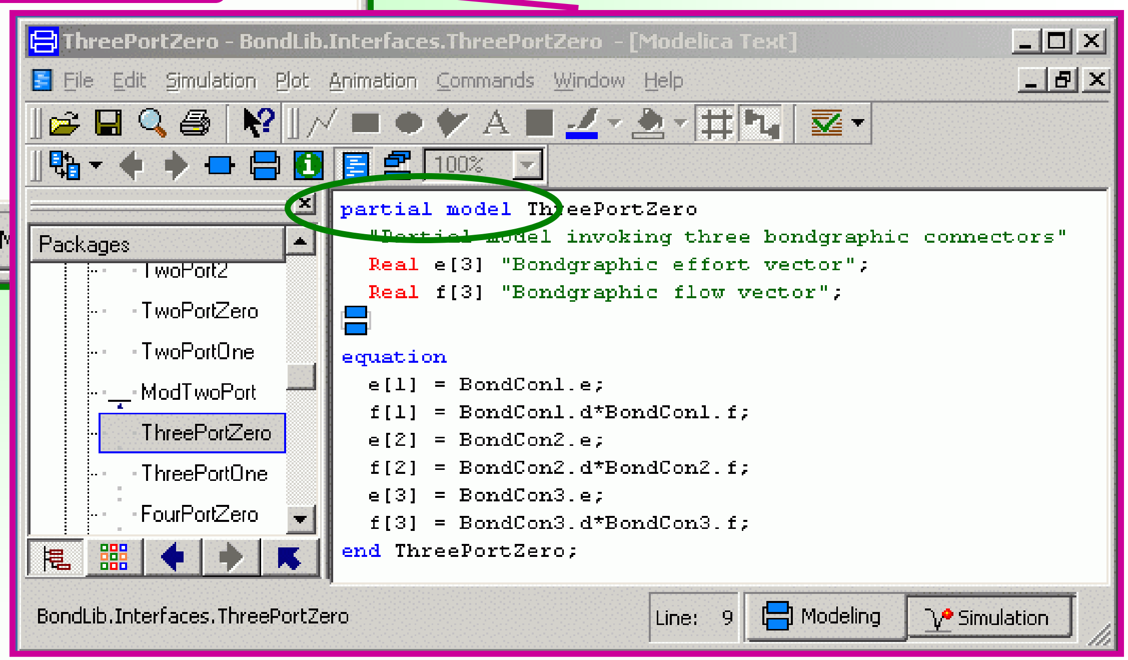




$$e[2] = e[1];$$
$$e[3] = e[2];$$
$$f[1] + f[2] + f[3] = 0;$$

The Junctions II

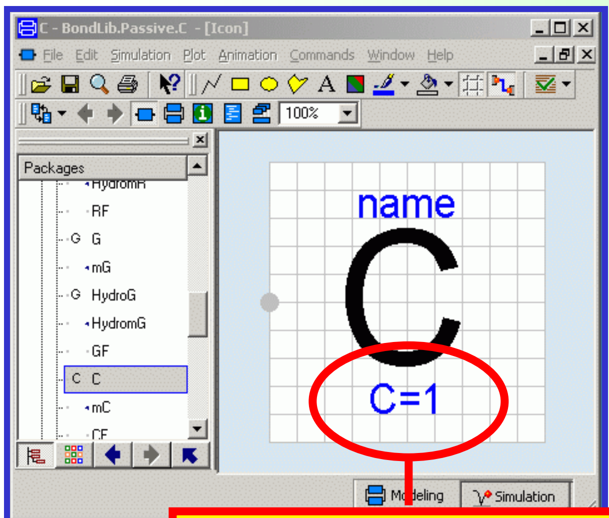




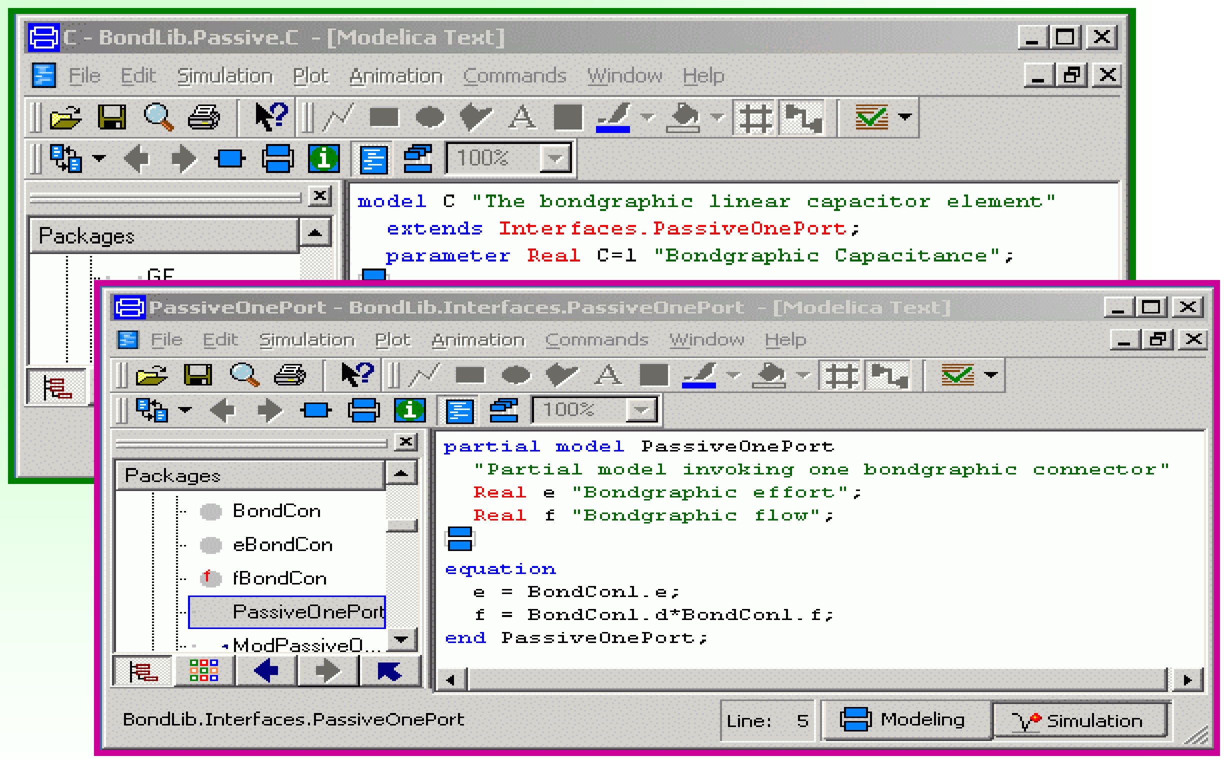
The ThreePortZero partial model drags the three bond connectors into the diagram window, and packs the individual bond variables into two vectors.

The Element Models

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



Add text " C=%C " to icon window.



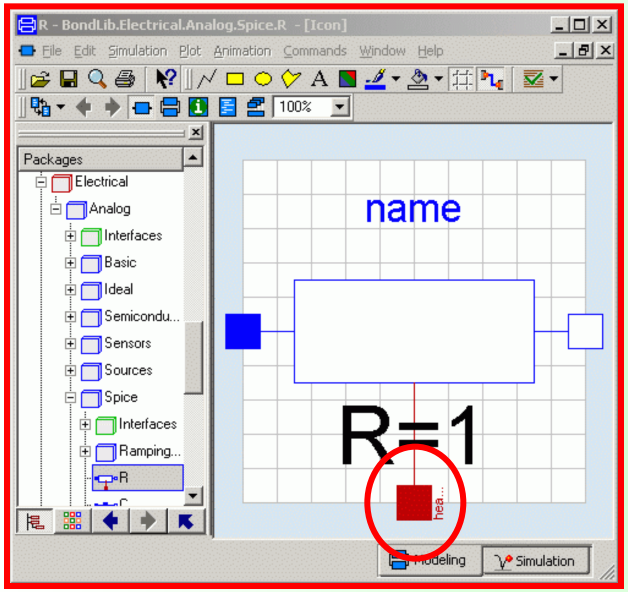
Model Wrapping

- Although it is possible to model physical systems manually down to the bond graph level, this may not always be convenient.
- The bond graph interface is the lowermost graphical interface that is still fully object-oriented.
- The interface is important as it keeps the distance between the lowermost graphical layer and the equation layer as small as possible.
- Higher level graphical layers can be built easily on top of the bond graph layer for enhanced convenience.

The Bond Graph Electrical Library

- It is possible to wrap any other object-oriented graphical modeling paradigm around the bond graph methodology.
- This was done with the analog electrical library that forms part of the standard library of Modelica.
- A new analog electrical library was created as part of the bond graph library.
- In this new library, the bottom layer graphical models were wrapped around a yet lower level bond graph layer.

The Wrapped Resistor Model



Icon window

The Spice-style resistor model has a ***thermal port*** carrying the heat generated by the resistor.

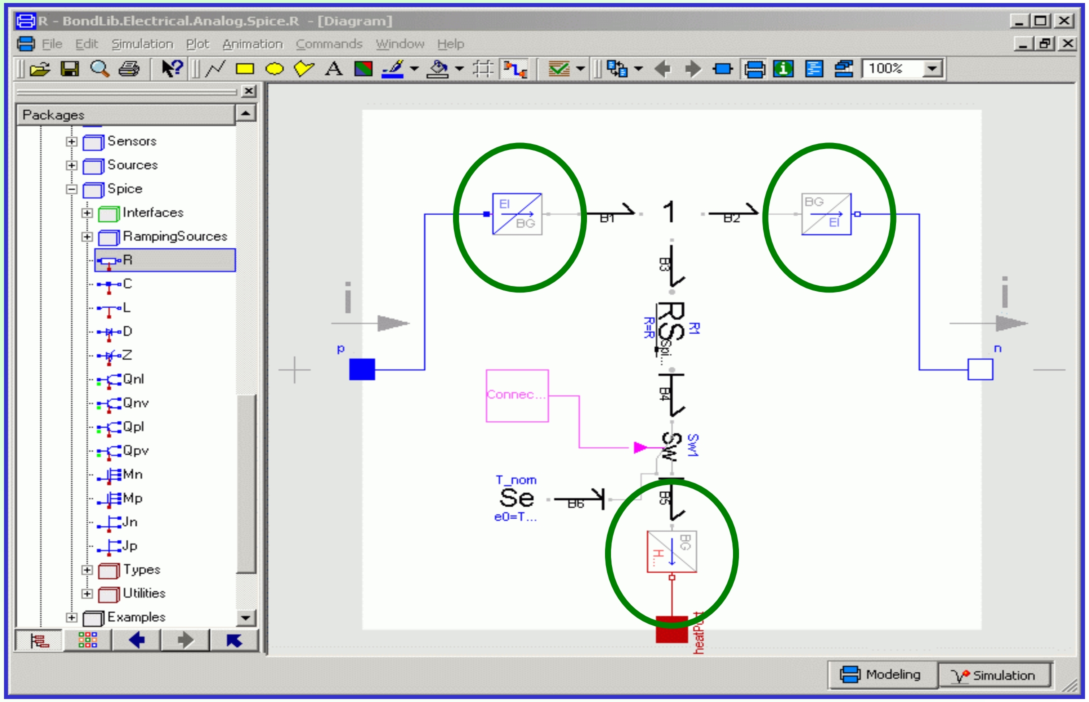
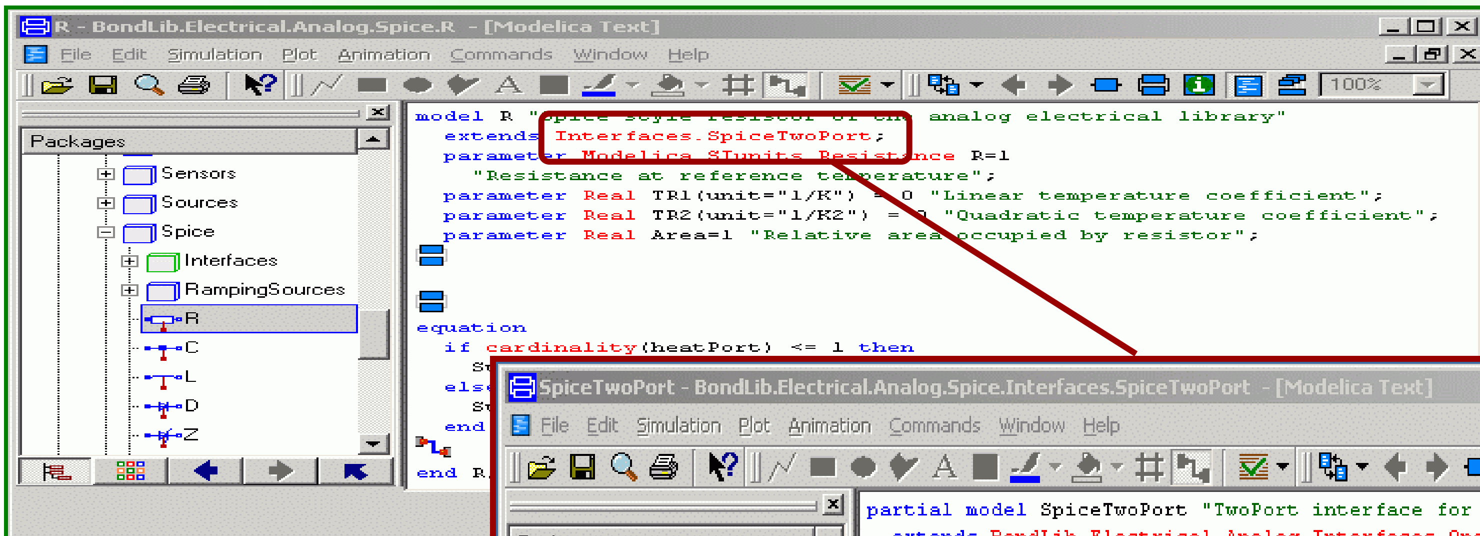
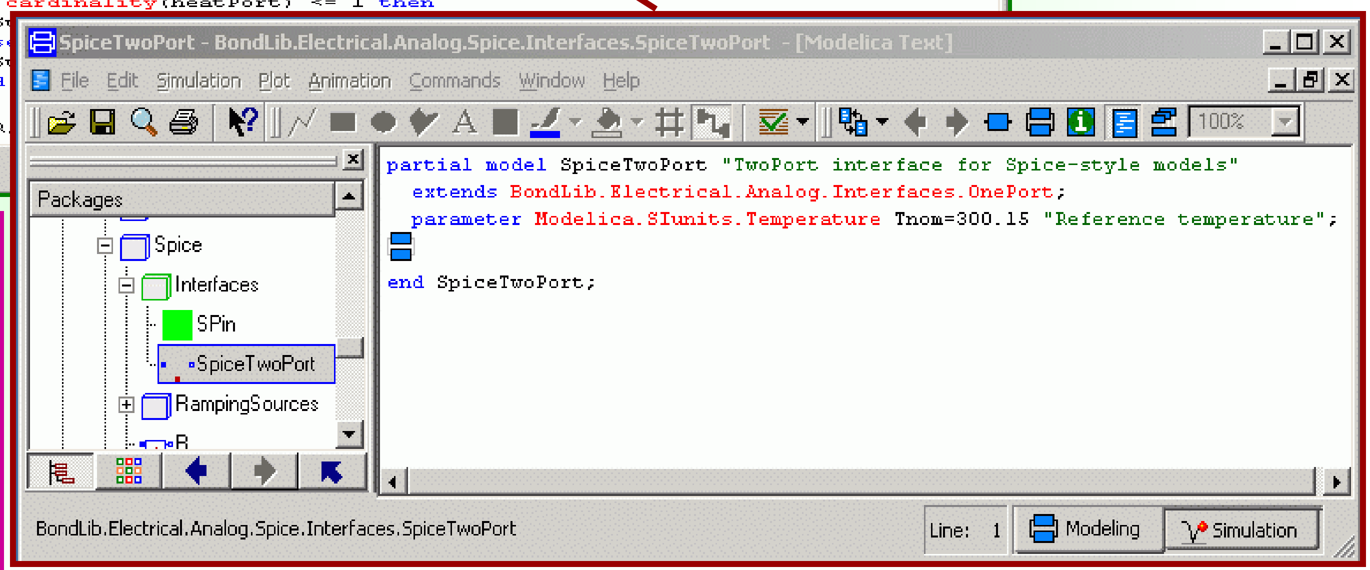


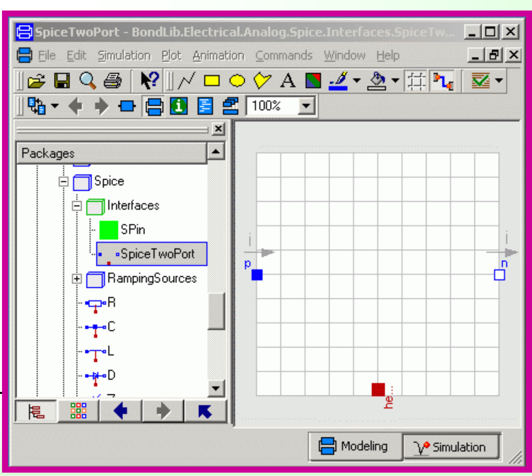
Diagram window

The ***wrapper models*** convert the connectors between the three domains: electrical, thermal, and bond graph.

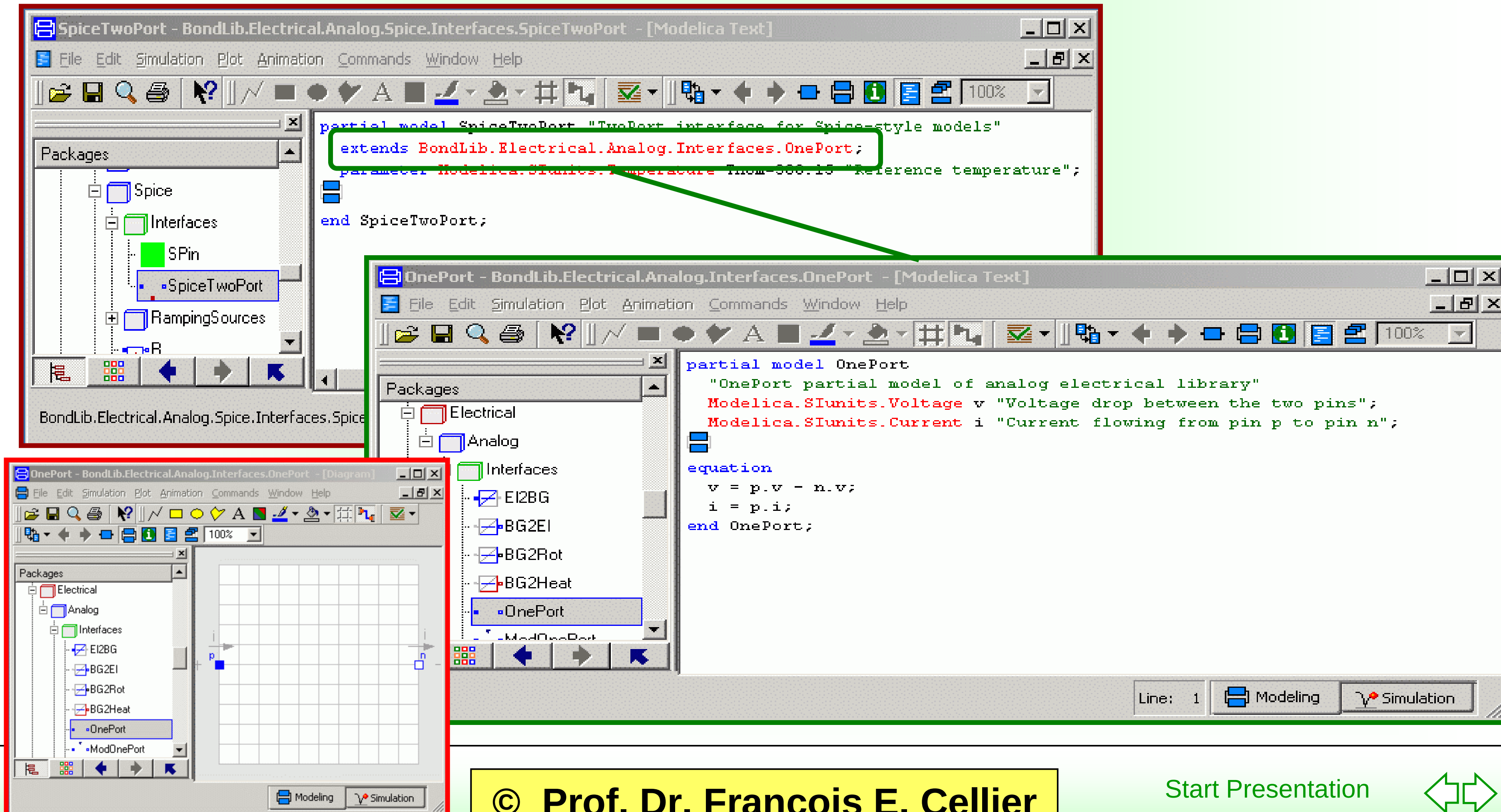
The Wrapped Resistor Model II



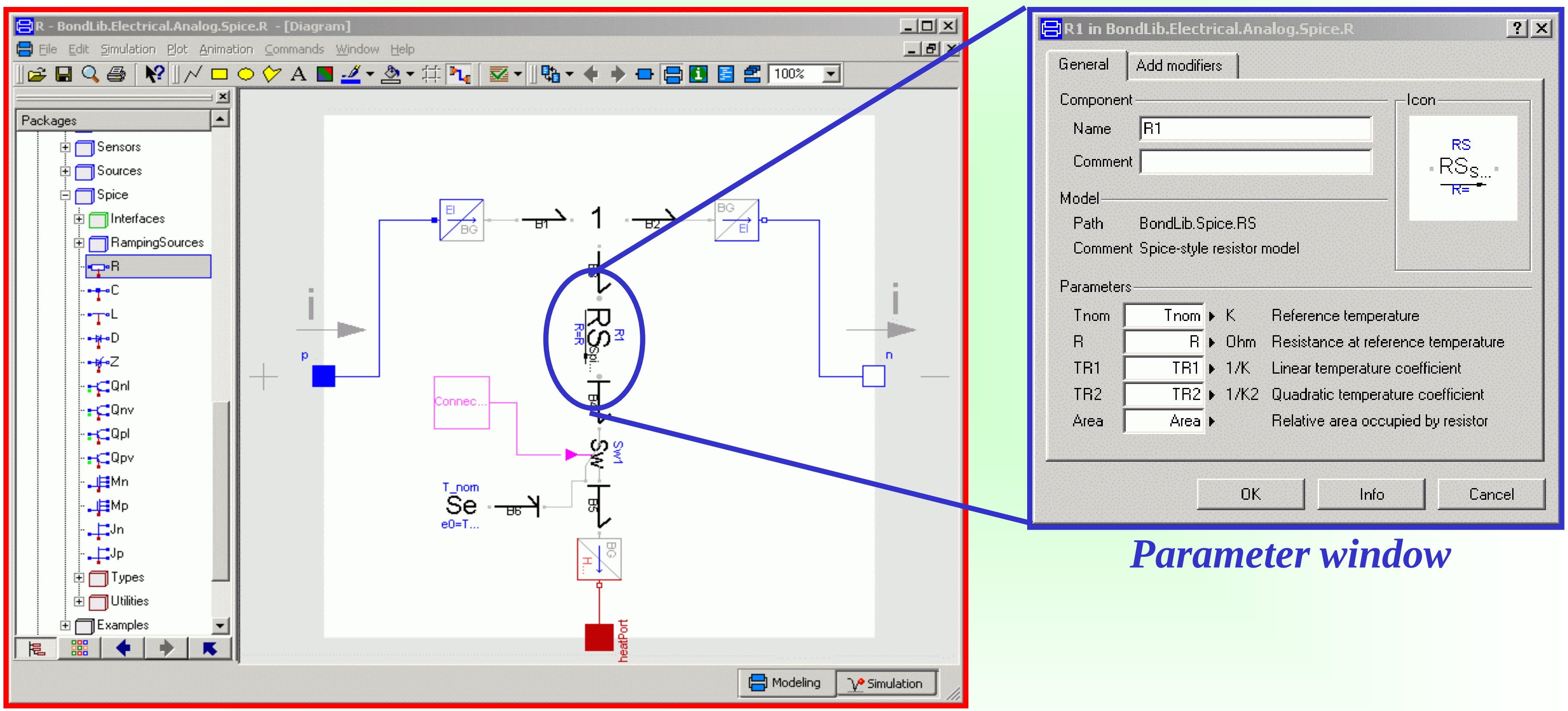




The Wrapped Resistor Model III



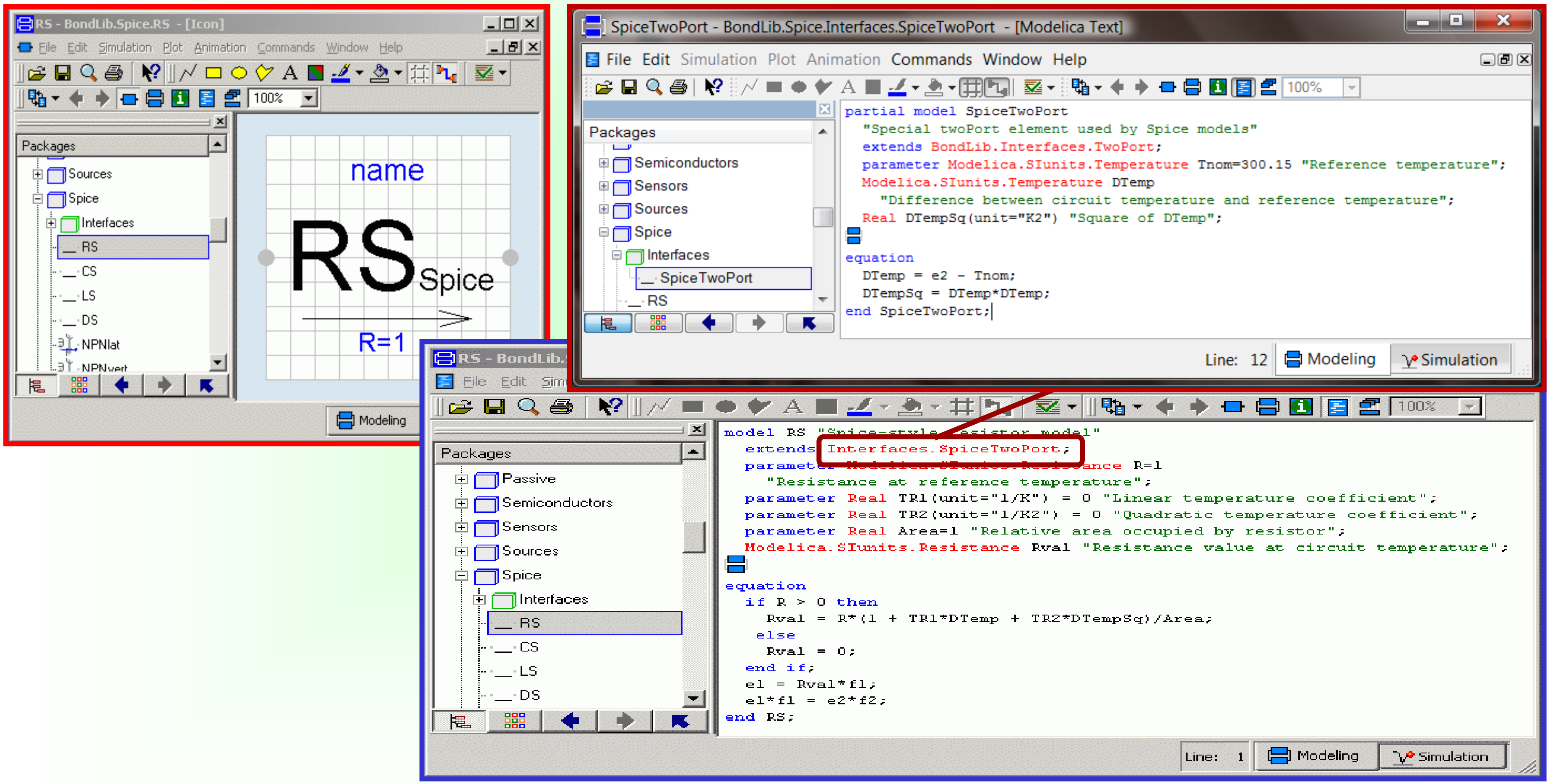
The Wrapped Resistor Model IV



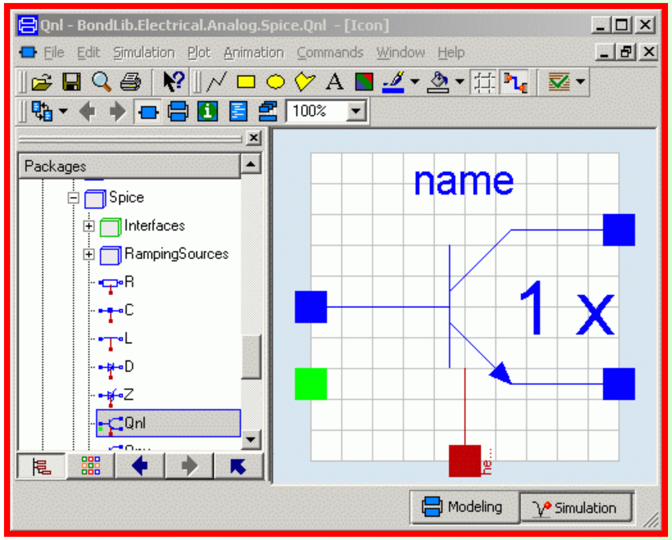
Parameter window

Diagram window

The Wrapped Resistor Model V



The Bipolar Junction Transistor



Icon window

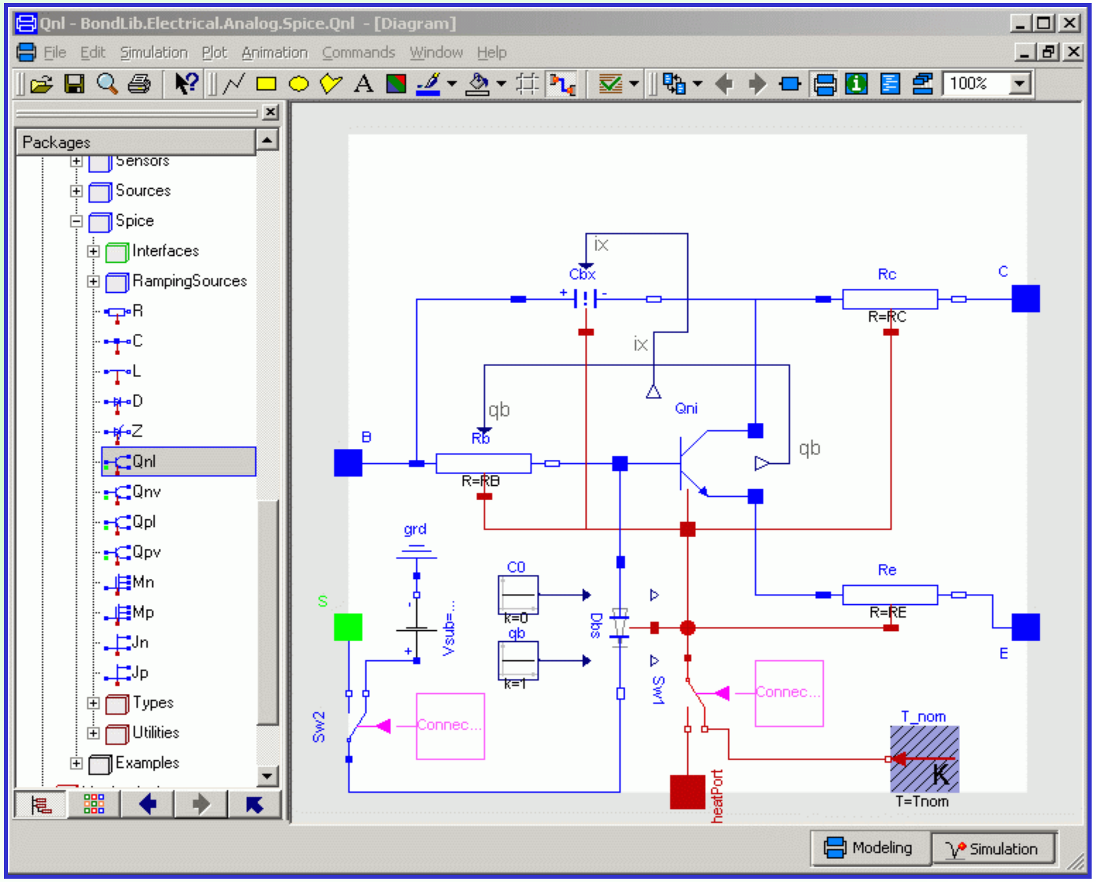
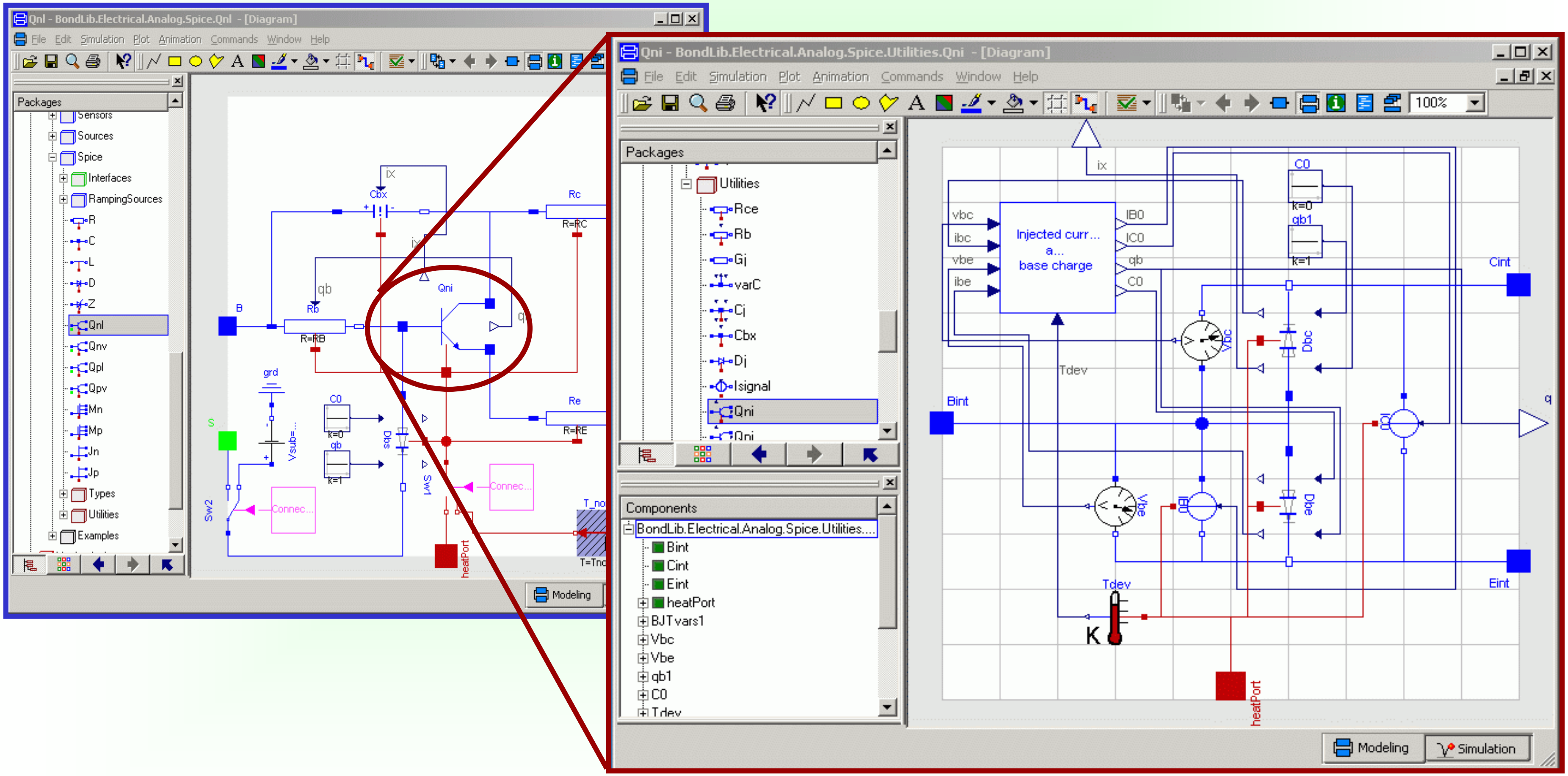
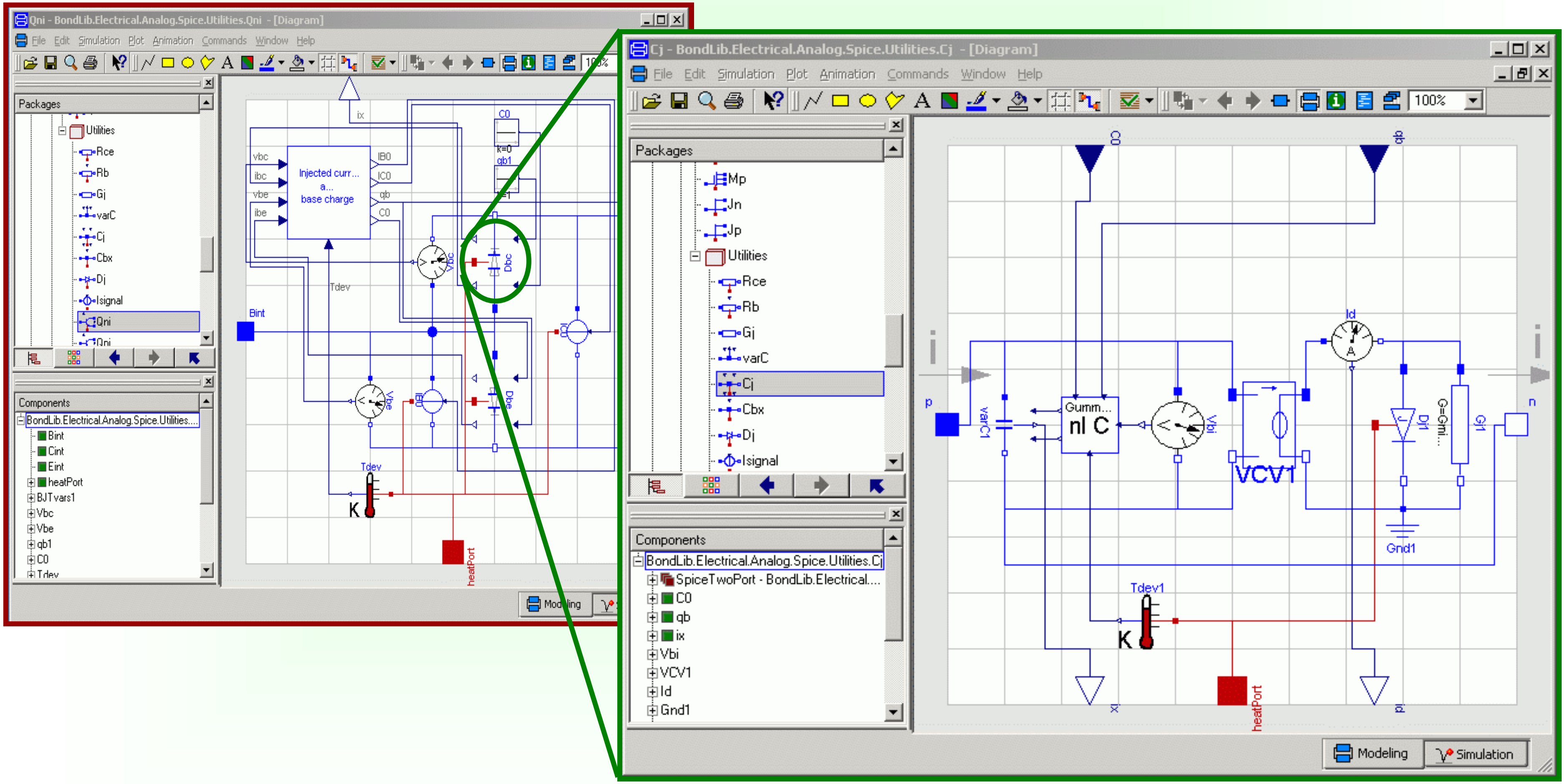


Diagram window

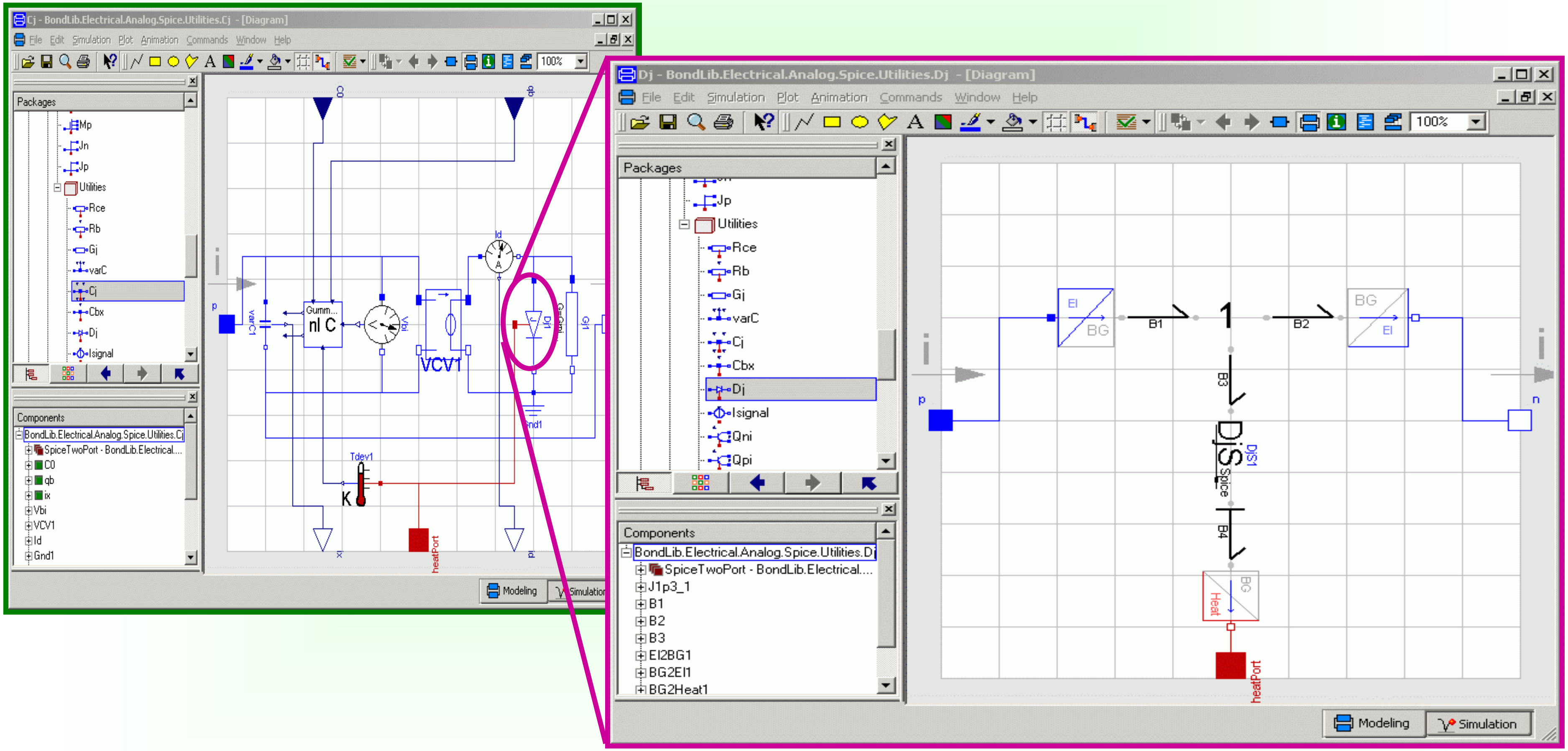
The Bipolar Junction Transistor II



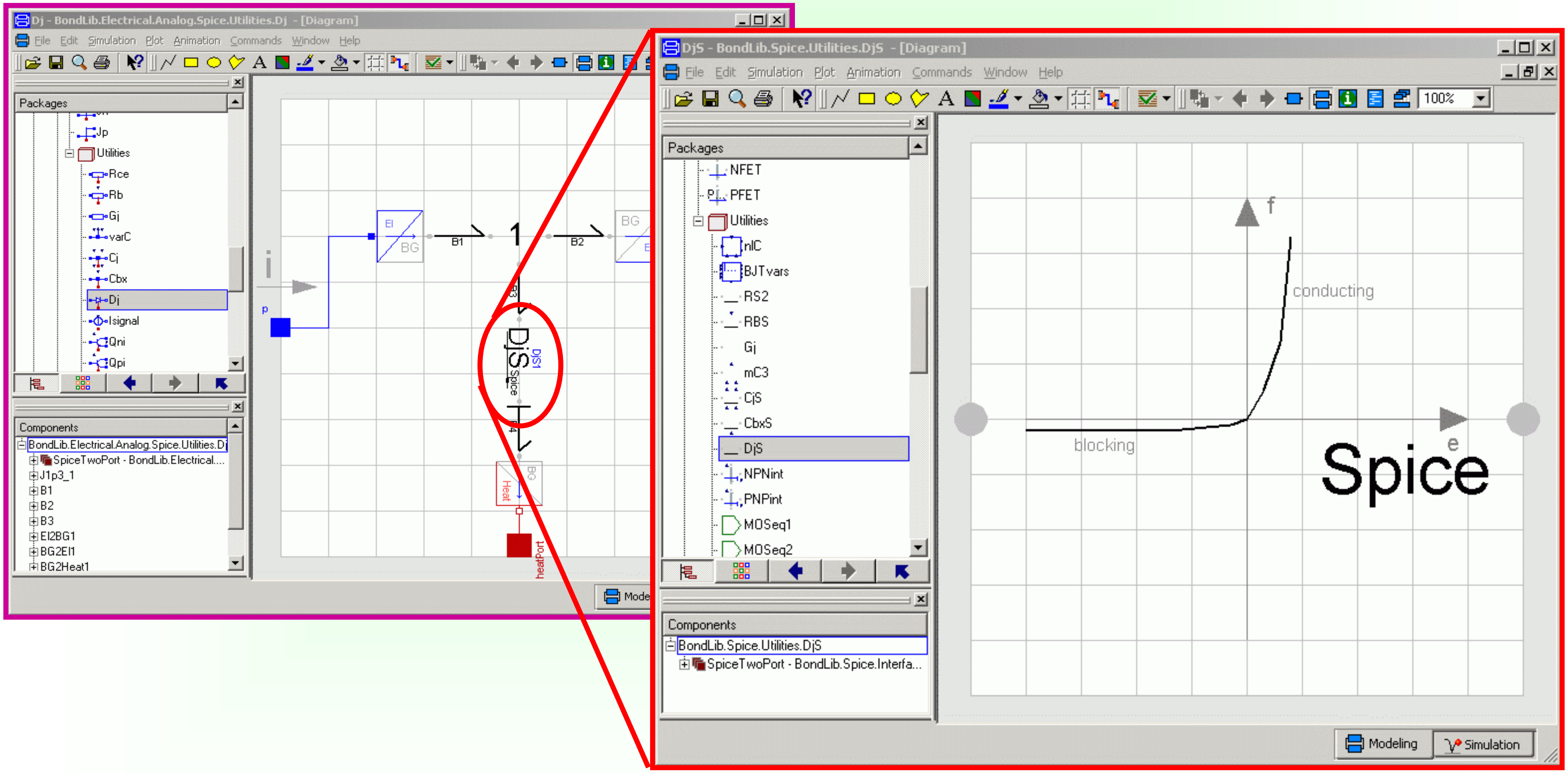
The Bipolar Junction Transistor III



The Bipolar Junction Transistor IV



The Bipolar Junction Transistor V

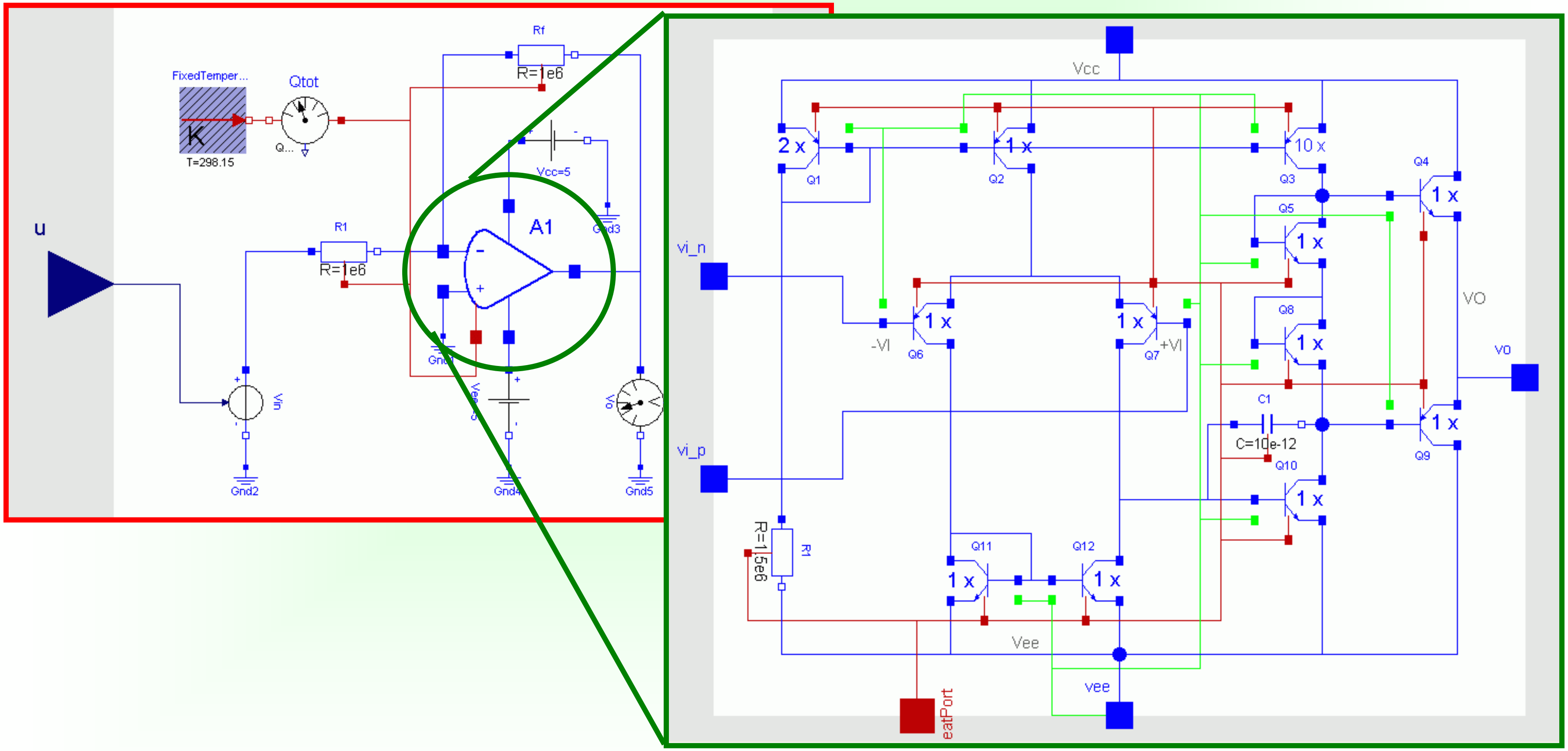


The Bipolar Junction Transistor VI

```
model DjS "Spice-style junction diode model for bipolar transistors"
  extends Interfaces.SpiceTwoPort;
  constant Modelica.SIunits.Entropy k=Modelica.Constants.k
    "Boltzmann's constant";
  constant Modelica.SIunits.ElectricCharge q=1.6021892e-19 "Electron charge";
  constant Real GapC1=7.02E-4 "First bandgap correction factor Silicon";
  constant Real GapC2=1108.0 "Second bandgap correction factor Silicon";
  parameter Modelica.SIunits.Current IS=1e-16
    "Saturation current at reference temperature";
  parameter Modelica.SIunits.Voltage EG=1.16
    "Energy gap for temperature effect on saturation current";
  parameter Real N=1 "Current emission coefficient";
  parameter Real XTI=3 "Saturation current temperature exponent";
  parameter Real Area=1 "Relative area occupied by diode";
  parameter Integer Level=2
    "Transistor modeling level (Ebers-Moll = 1; Gummel-Poon = 2)";
  parameter Real EMin=-100 "if x < EMin, the exp(x) function is linearized";
  parameter Real EMax=40 "if x > EMax, the exp(x) function is linearized";
  Modelica.SIunits.Voltage Vt "Thermal voltage";
  Modelica.SIunits.Current ISval "Saturation current at device temperature";
  Modelica.SIunits.Energy EGval "Activation energy at device temperature";

protected
  parameter Real ExMin = exp(EMin);
  parameter Real ExMax = exp(EMax);
  Real RTemp "Temperature quotient";
  Real et;

equation
  /* Compute thermal voltage as function of temperature */
  Vt = k*e2/q;
  et = e1/(N*Vt);
```

```
protected
  parameter Real ExMin = exp(EMin);
  parameter Real ExMax = exp(EMax);
  Real RTemp "Temperature quotient";
  Real et;

equation
  /* Compute thermal voltage as function of temperature */
  Vt = k*e2/q;
  et = e1/(N*Vt);

  /* Compute temperature dependence of saturation current */
  RTemp = e2/Tnom;
  EGval = EG - GapC1*e2*e2/(e2 + GapC2);
  ISval = IS*exp((RTemp - 1)*EGval/Vt + XTI*ln(RTemp));

  /* Compute diode characteristic */
  if Level==2 then
    /* Gummel-Poon model */
    f1 = ISval*Area*(if et < EMin then ExMin*(et - EMin + 1) - 1 else
                     if et > EMax then ExMax*(et - EMax + 1) - 1 else exp(et) - 1);
  else
    /* Ebers-Moll model */
    f1 = ISval*(if et < EMin then ExMin*(et - EMin + 1) - 1 else
                if et > EMax then ExMax*(et - EMax + 1) - 1 else exp(et) - 1);
  end if;

  /* Compute heat flow */
  f2 = 0;
end DjS;
```

Inverter Circuit



Inverter Circuit II

Messages - Dymola

Syntax Error | Translation | Dialog Error | Simulation

```
Translation of BondLib.Examples.ModelicaSpice.HifiOperationalAmplifier.Experiment:
DAE having 30217 scalar unknowns and 30217 scalar equations.

STATISTICS

Original Model
  Number of components: 4205
  Variables: 33551
    Constants: 372 (372 scalars)
    Parameters: 4672 (4672 scalars)
    Unknowns: 28507 (30217 scalars)
    Differentiated variables: 37 scalars
  Equations: 21789
    Nontrivial : 11996


Translated Model
  Constants: 10989 scalars
  Free parameters: 886 scalars
  Parameter depending: 4208 scalars
  Inputs: 0
  Outputs: 3 scalars
  Continuous time states: 37 scalars
  Time-varying variables: 1659 scalars
  Alias variables: 17708 scalars
  Assumed default initial conditions: 39
   LogDefaultInitialConditions=true; gives m
  Number of mixed real/discrete systems of equations: 0
  Sizes of linear systems of equations: {19, 5, 13, 7, 5, 5, 21, 12, 19, 5, 19, 5}
  Sizes after manipulation of the linear systems: {3, 0, 3, 2, 0, 0, 5, 2, 3, 0,
    4, 0}
  Sizes of nonlinear systems of equations: { }
  Sizes after manipulation of the nonlinear systems: { }
  Number of numerical Jacobians: 0

Finished
// experiment StopTime=0.0001
Finished
```

Initial number of equations

Final number of equations

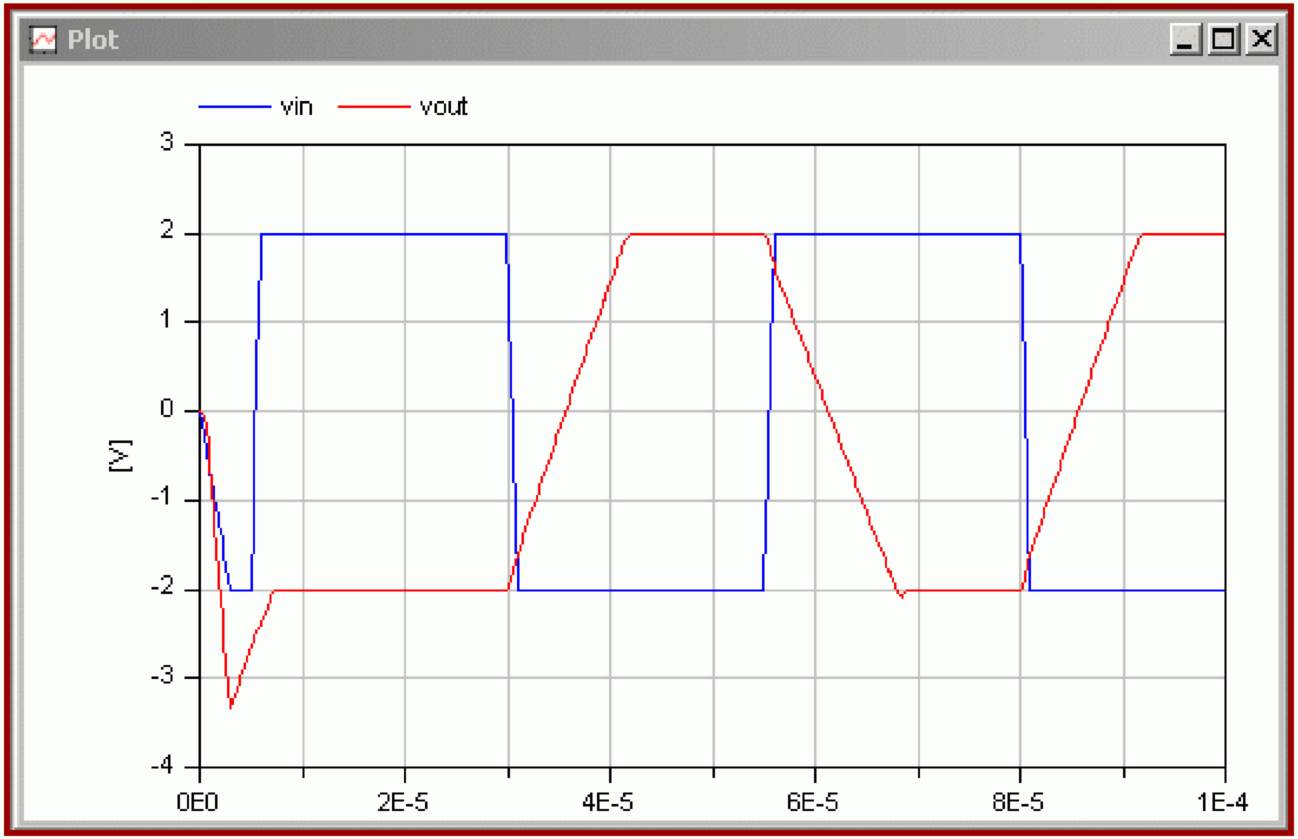
Messages - Dymola

Syntax Error | Translation | Dialog Error | Simulation

```
Log-file of program ./dymosim
(generated: Thu Oct 25 14:08:31 2007)

dymosim started
... "dsin.txt" loading (dymosim input file)
... "Experiment.mat" creating (simulation result file)

Integration started at T = 0 using integration method DASSL
(DAE multi-step solver (dassl/dasslrt of Petzold modified by Dynasim))
Integration terminated successfully at T = 0.0001
   CPU-time for integration      : 10.7 seconds
   CPU-time for one GRID interval: 21.4 milli-seconds
   Number of result points       : 626
   Number of GRID   points       : 501
   Number of (successful) steps  : 2721
   Number of F-evaluations       : 47708
   Number of H-evaluations       : 3639
   Number of Jacobian-evaluations: 1138
   Number of (model) time events : 9
      ber of (U) time events     : 0
      ber of state    events     : 58
      ber of step     events     : 0
   Minimum integration stepsize  : 1.88e-013
   Maximum integration stepsize  : 6.29e-006
   Maximum integration order     : 5
Calling terminal section
... "dsfinal.txt" creating (final states)
```

Simulation Time

Simulation Results



References

- Cellier, F.E. and R.T. McBride (2003), "Object-oriented modeling of complex physical systems using the Dymola bond-graph library," *Proc. ICBGM'03, Intl. Conf. Bond Graph Modeling and Simulation*, Orlando, FL, pp. 157-162.

- Cellier, F.E. and A. Nebot (2005), "The Modelica Bond Graph Library," *Proc. 4th Intl. Modelica Conference*, Hamburg, Germany, Vol.1, pp. 57-65.

- Cellier, F.E., C. Clauß, and A. Urquía (2007), "Electronic Circuit Modeling and Simulation in Modelica," *Proc. 6th Eurosim Congress*, Ljubljana, Slovenia, Vol.2, pp. 1-10.

- Cellier, F.E. (2007), *The Dymola Bond-Graph Library*, Version 2.3.