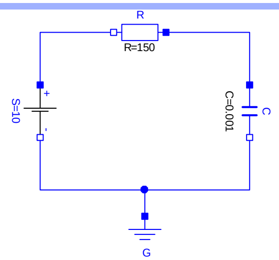
Virtual Physics
Equation-Based Modeling

TUM, October 21, 2014

Modeling in Modelica – Graphical Modeling

```
model SimpleCircui
  import SI = Mode
  parameter SI.Cap
  parameter SI.Res
  parameter SI.Vol
  SI.Current i ”Cu
 ”Capacitor Voltag
initial equation
  uC = 0;
equations
  V0-uC = R*i;
  der(uC)*C = i;
end SimpleCircuit;
```



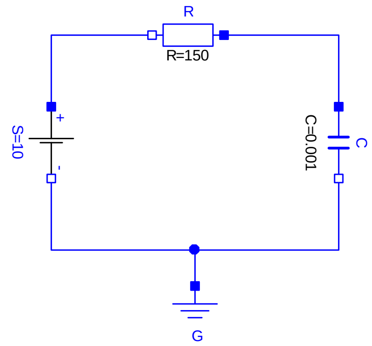
Dr. Dirk Zimmer

German Aerospace Center (DLR), Robotics and Mechatronics Centre

Outline

In this lecture, the language Modelica is officially introduced.

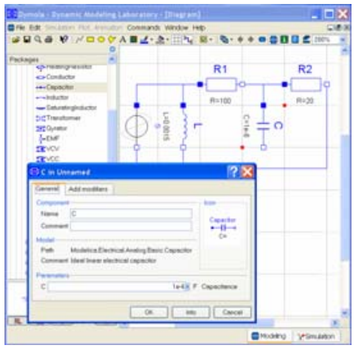
- The graphical modeling layers in Dymola
- Annotations
- Parameter GUI
- Initialization via GUI
- Modelica Blocks
- Inputs / Outputs
- Blocks and Functions



Graphical Modeling

So far, we have only looked at the textual side of modeling.

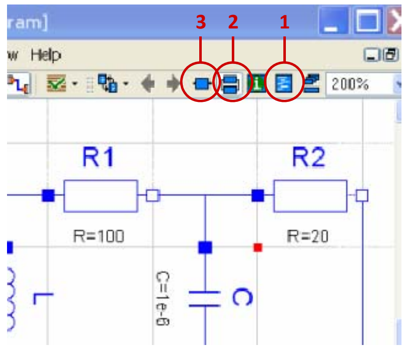
- Using a modern modeling environment like Dymola, most modeling is performed graphically.
- Textual modeling is only done for the lower level tasks.



The Modeling Layers

To this end, Dymola offers three distinct modeling layers.

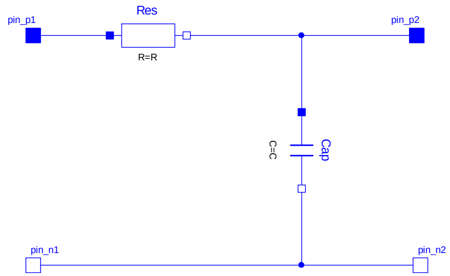
- The inner textual representation (1)
- The inner graphical representation (2)
- The outer graphical representation (3)



Inner Graphical Layer

Let us model an RC-Filter.

- We start with the inner graphical representation.
- Here we model the actual sub-circuit



Textual Layer

Let us model an RC-Filter.

- On the textual layer, we provide two parameters for the resistor and the capacitor

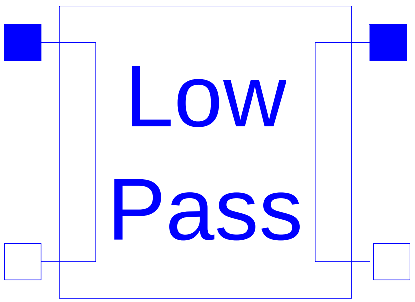
```
model RCFilter
  import SI = Modelica.SIunits;
  parameter SI.Resistance R = 100;
  parameter SI.Capacitance C = 1e-3;

  Modelica…Resistor Res(R=R);
  Modelica…Capacitor Cap(C=C);
  Modelica…NegativePin pin_n1;
  Modelica…NegativePin pin_n2;
  Modelica…PositivePin pin_p1;
  Modelica…PositivePin pin_p2;
equation
  connect(pin_p1, Res.p);
  connect(Res.n, pin_p2);
  connect(Cap.p, Res.n);
  connect(Cap.n, pin_n2);
  connect(pin_n1, pin_n2);
end RCFilter;
```


Outer Graphical Layer

Let us model an RC-Filter.

- The outer graphical representation already contains the connectors
- Now we design a suitable symbol for our model.
- Now it is ready to be used.



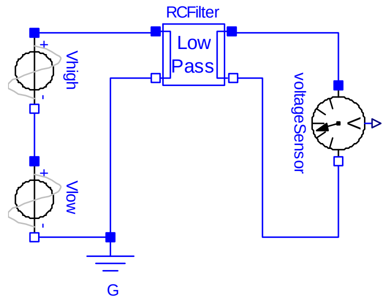
Application Example

Let us model an RC-Filter.

- Here is an application of our RC-Filter component.
- The parameters can be set by clicking on the component.

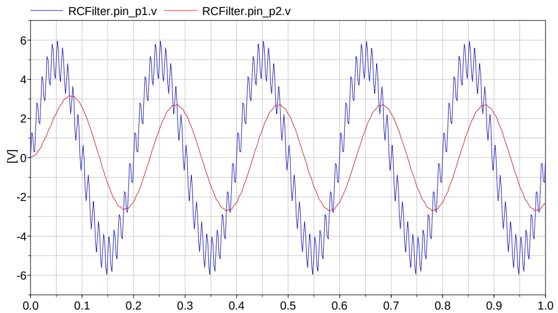
C = 0.01

R = 5



Application Example

Simulation Result



Annotations


```
model RCFilter
  import SI = Modelica.SIunits;
  parameter SI.Resistance R = 100;
  parameter SI.Capacitance C = 1e-3;

  Modelica…Resistor Res(R=R) a;
  Modelica…Capacitor Cap(C=C) a;
  Modelica…NegativePin pin_n1 a;
  Modelica…NegativePin pin_n2 a;
  Modelica…PositivePin pin_p1 a;
  Modelica…PositivePin pin_p2 a;
equation
  connect(pin_p1, Res.p) a;
  connect(Res.n, pin_p2) a;
  connect(Cap.p, Res.n) a;
  connect(Cap.n, pin_n2) a;
  connect(pin_n1, pin_n2) a;
end RCFilter;
```

- How is the graphical information stored within the model.
- Modelica uses annotations for this purpose.
- Dymola typically hides annotations and represents them by the symbol: **a**
- The visibility of annotations can be enabled in the Dymola Editor.

Annotations


```
annotation(Icon(graphics={
  Rectangle(
    extent={{-80,80},{80,-80}},
    lineColor={0,0,255},
    fillColor={255,255,255},
    fillPattern=FillPattern.Solid),
  Line(
    points={{-90,60},{-60,60},
            {-60,-60},{-90,-60}},
    color={0,0,255},
    smooth=Smooth.None),
  Line( points={{90,60},{60,60},
                {60,-60},{90,-60}},
    color={0,0,255},
    smooth=Smooth.None),
  Text(extent={{-60,60},{60,2}},
     lineColor={0,0,255},
     textString="Low"),
   …
```

- How is the graphical information stored within the model.
- Modelica uses annotations for this purpose.
- Dymola typically hides annotations and represents them by the symbol: **a**
- The visibility of annotations can be enabled in the Dymola Editor.

Annotations


```
 annotation(
Documentation(info=
  "<html>
 <p><h4>RC-Lowpass</h4></p>
 <p>This is a basic model of an
    RC-Lowpass filter.</p>
  </html>")
);


 parameter SI.Resistance
   R = 1 annotation(
    Dialog(
     group="RCSpecification“
     )
    );
```

- Annotations are also used to store the HTML-documentation of the model
- Also the the look of the Parameter GUI can be determined by annotations.

Code distribution

Robotics and Mechatronics Centre

Following classifications of aspects seems appropriate for Modelica

Physical modeling: The modeling of the physical processes that are based on differential-algebraic equations (DAEs).

System hints: The supply of hints or information for the simulation-system.

3D Visualization: Description of corresponding 3D-entities that enable a visualization of the models.

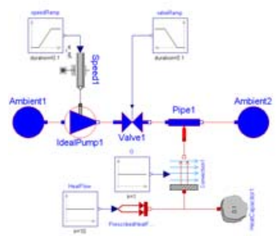
GUI-Representation: Description of an icono-graphic representation for the graphical user-interface (GUI) of the modeling environment.

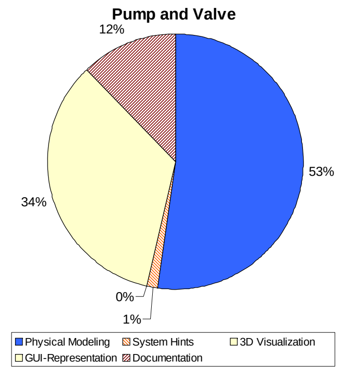
Documentation: Additional documentation that addresses to potential users or developers.

Code distribution

Robotics and Mechatronics Centre

- Modelica.Thermal. FluidHeatFlow.Examples. PunpAndValve

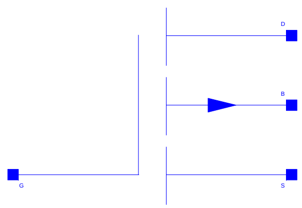


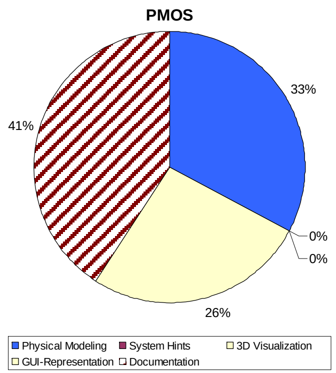


Code distribution

Robotics and Mechatronics Centre

- Modelica.Electrical. Analog.Semiconductors. PMOS

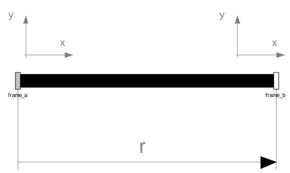


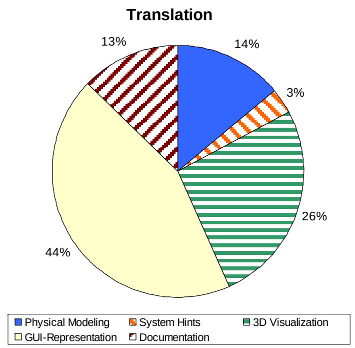


Code distribution

Robotics and Mechatronics Centre

- Modelica.Mechanics. MultiBody.Parts. FixedTranslation





Initialization

Robotics and Mechatronics Centre

```
model RCFilter
  import SI = Modelica.SIunits;
  parameter SI.Resistance R = 100;
  parameter SI.Capacitance C = 1e-3;
  parameter Boolean initialize
    = false;
  parameter Real vC0;
  Modelica…Resistor Res(R=R);
  Modelica…Capacitor Cap(C=C);
  Modelica…NegativePin pin_n1;
  …
 initial equation
if initialize then
  Cap.v = vC0;
end if;

equation
  connect(pin_p1, Res.p);
  connect(Res.n, pin_p2);
  …
end RCFilter;
```

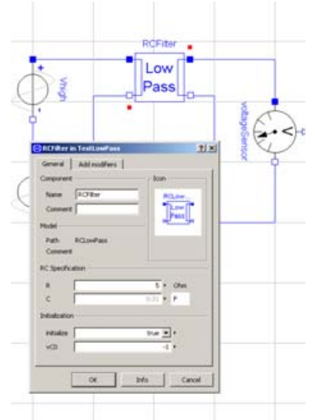
The set of initial conditions depends on the circuit structure. Hence, they must be stated globally for each new system.

To enable a convenient formulation of the initial conditions, parameters are often offered.

We use our RC-Circuit as an example.

Initialization

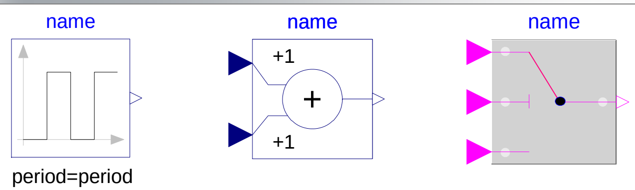
Robotics and Mechatronics Centre



- Within an electric circuit, the modeler can select the components he wants to initialize.
- Not all combinations are valid!
- This is a topic that will be discussed intensively in future lectures.

The Modelica Blocks

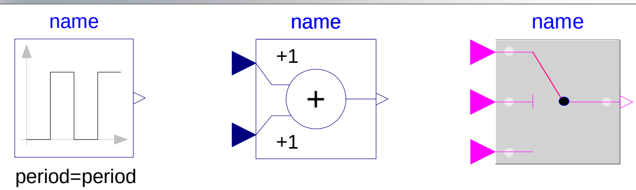
Robotics and Mechatronics Centre



- Not all modeling work represents physical processes.
- Often we want to model signals. This can include simple algebraic computations or elaborate control loops.
- Modelica offers the Modelica.Blocks Library for this purpose.

The Modelica Blocks

Robotics and Mechatronics Centre



- Modelica Blocks features a variety of models.
- There are various signal sources and algebraic and logic elements
- Also a number of control elements is ready to be used.

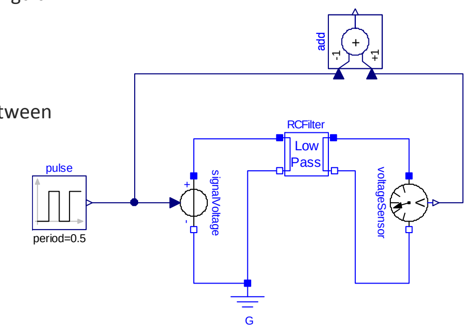
The Modelica Blocks

- Blocks can interact with physical models by the means of…
- …Sensors…
- … and Sources

Application Example

Here we use Block models..

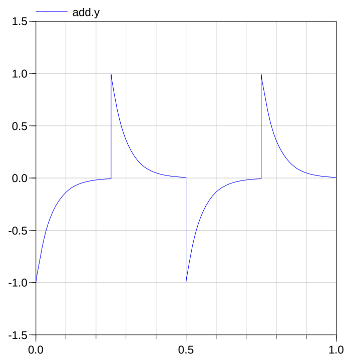
- …to describe an rectangular source voltage signal
- …and to compute the difference voltage between input and output.



Application Example

Here we use Block models..

- …to describe an rectangular source voltage signal
- …and to compute the difference voltage between input and output.



The Block Class


```
block Add

  RealInput u1;
  RealInput u2;
  RealOutput y;

  parameter Real k1=+1;
  parameter Real k2=+1;

equation

  y = k1*u1 + k2*u2;

end Add;
```

- Blocks use different connectors.
- There are input connectors and output connectors.
- Any input must be connected to an output.
- An output can be connected to an arbitrary number of matching inputs.
- The input-output relation does NOT impose a computational causality. It might be that the input is computed, given the desired output.

The Block Class

```
block Add

  RealInput u1;
  RealInput u2;
  RealOutput y;

  parameter Real k1=+1;
  parameter Real k2=+1;

equation

  y = k1*u1 + k2*u2;

end Add;
```

- A block is simply a model that has only input and output connectors.
- When locally checking a block, all inputs are assumed to be known and all outputs represent unknowns.
- Blocks may define state-variables So does the integrator block.

Defining Functions

```
function fak

  input Integer n;
  output Integer y;

algorithm

  y := 1;

  while n>1 loop

    y := y*n;
    n := n-1;

  end while;

end fak;
```

- A function is similar to a block.
- Functions have an arbitrary number of inputs and outputs.
- The order of declaration does matter since this determines the way the function is called.
- In contrast to blocks, functions cannot define state-variables. Also parameter declarations are not allowed in functions

Defining Functions

```
function fak

  input Integer n;
  output Integer y;

algorithm

  y := 1;

  while n>1 loop

    y := y*n;
    n := n-1;

  end while;

end fak;
```

- The computation of the function is typically expressed within an algorithm section.
- Auxiliary variables (non-input/output) must be declared protected.
- The algorithm section simply expresses a sequence of computations as in imperative programming languages. There exist even loop statements.
- Modelica functions must be pure, this means they shall not contain side-effects. (There are exceptions)

Defining Functions

```
[…]

 z = sin(phi)*g
 z = der(w)
 w = der(phi)

[…]
```

- The function may now be used within the equations section of a model.
- This is not a direct function call, since the simulator will finally determine if and how many times the function will be called.
- This is also the reason why the function must (or should) be free of side-effects.

Conclusions

Let us conclude by a few general remarks

- Most higher-level modeling is performed graphically.
- Annotations are used to store the corr. information.
- Physical modeling is extended by blocks and functions.
- Blocks are often used to design a controller.
- Algorithmic parts are supported by means of functions.

Outlook

- Next lecture, we are going to examine the compilation of Modelica Models.

Questions?