INTRODUCTION TO MODELICA

Reconfigurable models

Modelon

OVERVIEW

- Creating reconfigurable models
  - Templates and interfaces
  - Conditional components
  - Arrays of components
- Replaceable functions

CREATING RECONFIGURABLE MODELS

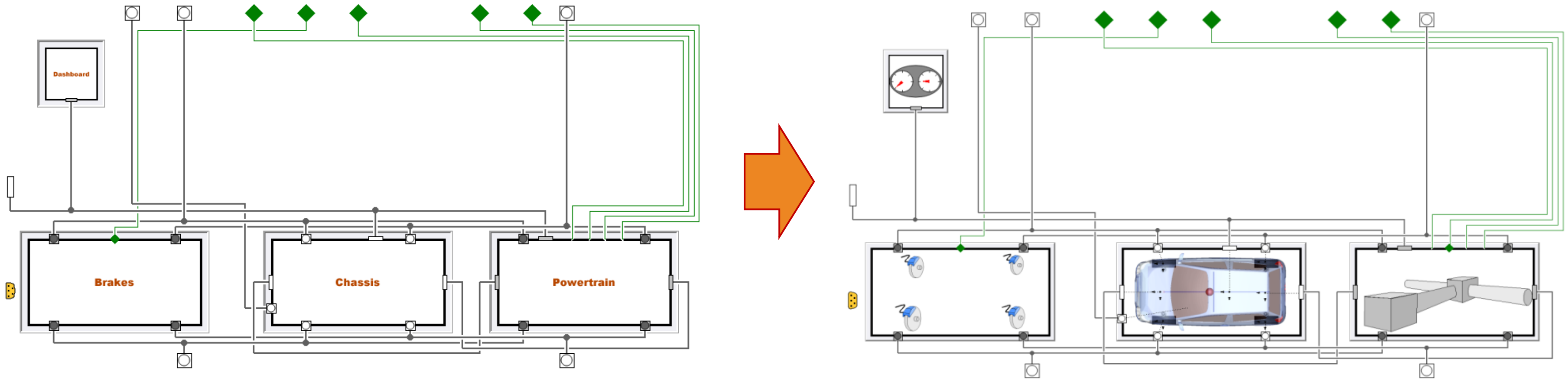
INTERFACES AND TEMPLATES

INTERFACES AND TEMPLATES

- Efficient model development requires that the models are designed so that the code can be reused:
  - Hierarchical structure allows models to be reused on different levels, components, subsystems and systems.
  - Inheritance allow properties that are common for several models to be defined only once.
  - Combining the above two allows for creation of templates that brings code reuse to a new level.
- The interfaces and templates concept was introduced in the Vehicle Dynamics Library to efficiently systematically handle complex models and the huge amount of variants.

TEMPLATES

- A template is a topology definition
- A template consist of
  - interfaces that act as placeholders
  - connections between the placeholders
- Using a template, configurations can be created by just specifying the components / subsystems
- Variant generation and maintenance is straight-forward



INTERFACES

- An interface contains
  - Connectors
  - Common parameters
- Well-defined interfaces ensures plug-compatibility: all models that share an interface will also fit in the template.
- Interfaces are used with inheritance, in Modelica the keyword ‘extends’.

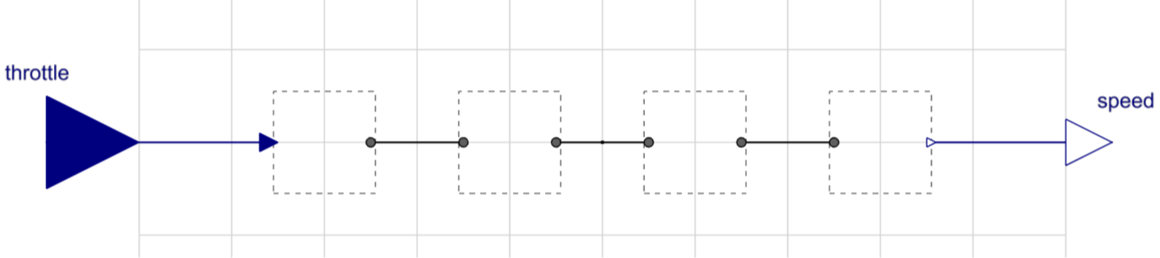
```
partial model EngineInterface
    Modelica.Blocks.Interfaces.RealInput throttle;
    Modelica.Mechanics.Rotational.Interfaces.Flange_a shaft;
end EngineInterface;
model Engine
  extends EngineInterface;
...
end Engine;
```

- Engine has all the properties of the EngineInterface.

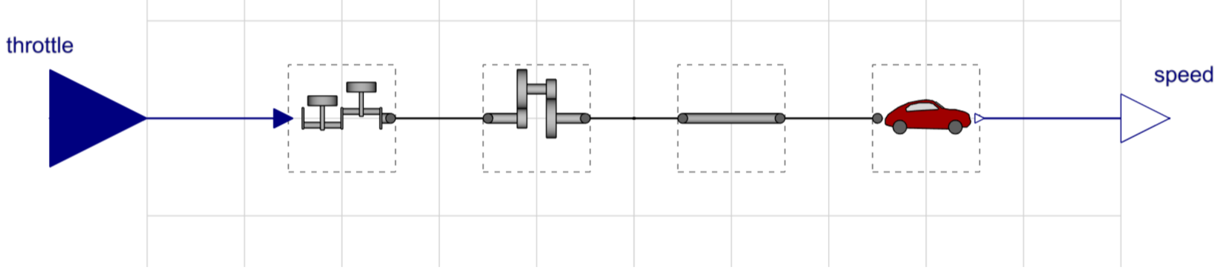
TEMPLATES

- Now if we have interfaces for each subsystem, we can design a template with replaceable components:

```
partial model StandardCar
  replaceable Interfaces.Engine engine constrainedby Interfaces.Engine engine
    annotation (...);
```

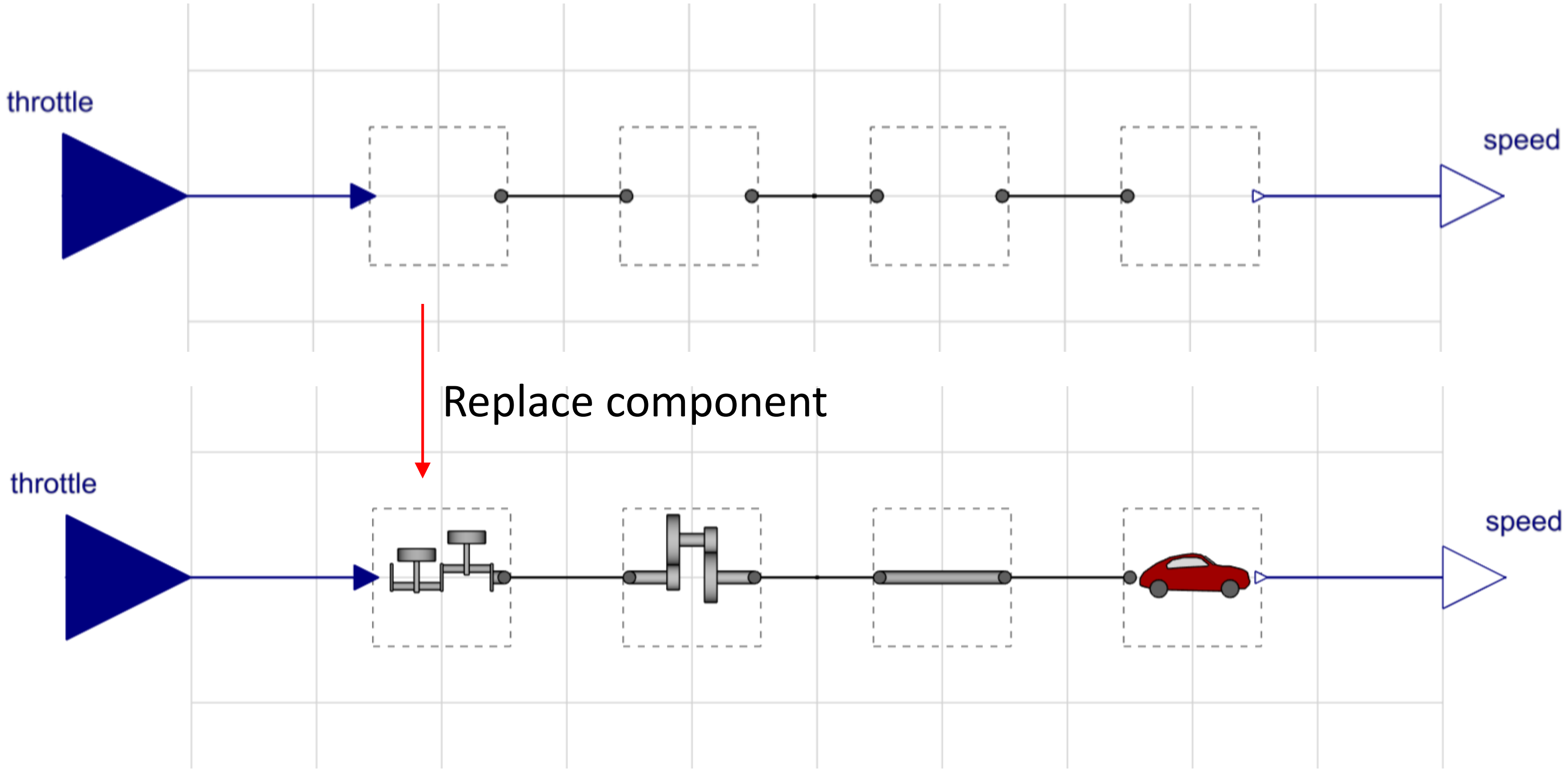



```
model Car "Actual configuration"
  extends Templates.StandardCar(
    redeclare SubSystems.BasicEngine engine,
```



SYSTEM VARIANT

- If we have a template, with well defined interfaces, we can extend that and create a specific system variant:



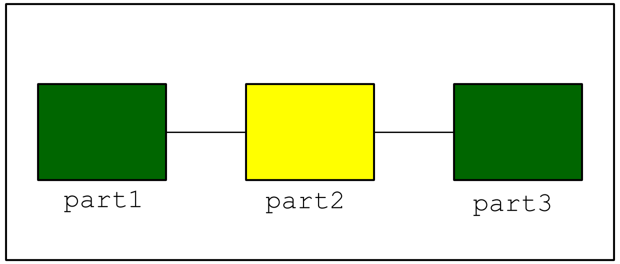
APPLICATIONS OF REPLACEABLE

The prefix *replaceable* in an element declaration allows for later modification of that element using *redeclare*.

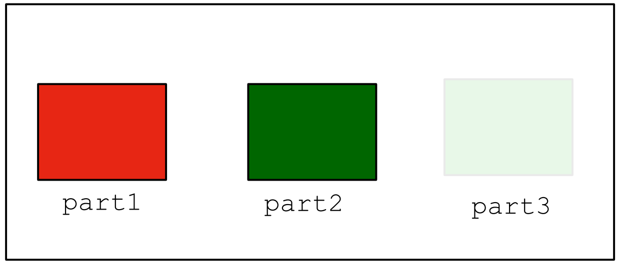
- **Replaceable components**
  - applies to exactly 1 component (used for template design in Vehicle Dynamics Library as just illustrated)
- **Replaceable classes**
  - applies to many components at once, sub-model may be propagated (used e.g. for heat transfer in Air Conditioning Library)
- **Replaceable packages**
  - many functions and models can be replaced consistently at once (e.g. medium properties)

REPLACEABLE COMPONENTS

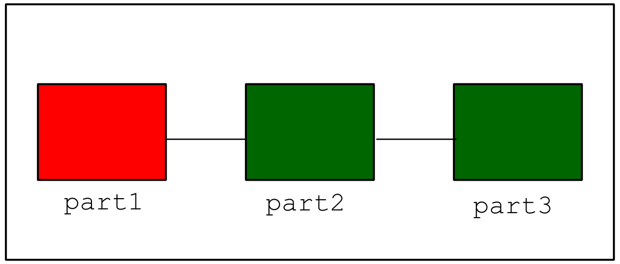
```
model M1
  replaceable GreenModel  part1(p=2);
  replaceable YellowModel part2;
  replaceable GreenModel  part3;
  connect(…);
end M1;
```




```
model M2
extends M1(redeclare RedModel part1,
           redeclare GreenModel part2);
```

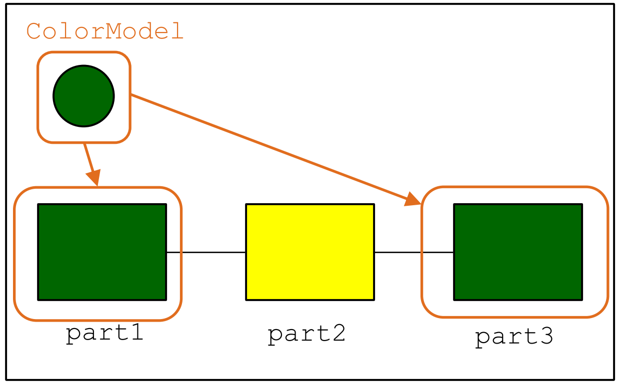


```
model M "equivalent to M2"
  RedModel   part1(p=2);
  GreenModel part2;
  GreenModel part3;
  connect(…);
end M;
```

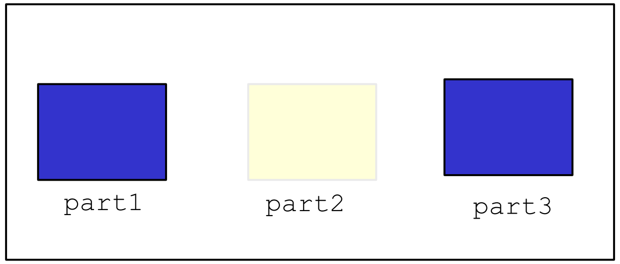


REPLACEABLE CLASSES

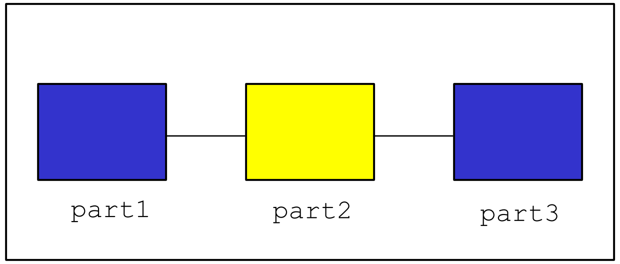
```
model M1
  replaceable model ColorModel=GreenModel;
  ColorModel  part1(p=2);
  YellowModel part2;
  ColorModel  part3;
  connect(…);
end M1;
```




```
model M2
extends M1 redeclare model
ColorModel=BlueModel);
```



```
model M "equivalent to M2"
  BlueModel   part1(p=2);
  YellowModel part2;
  BlueModel part3;
  connect(…);
end C;
```



CREATING RECONFIGURABLE MODELS

CONDITIONAL DECLARATIONS

CONDITIONAL COMPONENTS

- When switching on and off a feature, or switching between a limited amount of models, conditional components is an alternative.
- Typical example is visualization
  - For real-time purposes, and other cases when simulation speed is prioritized, visualization primitives can be removed with a parameter.

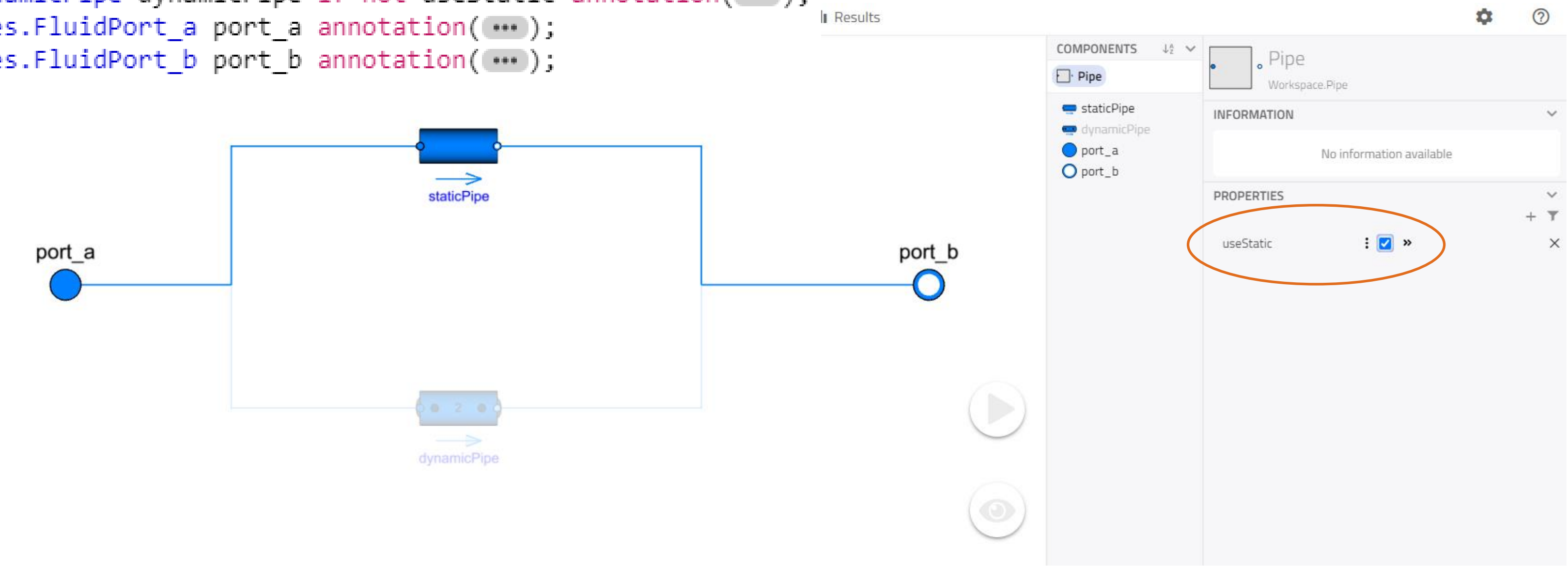
```
model Body
  parameter Boolean visualize = true;
  visualizer vis(...) if visualize;
  ...
```

- This construction can also be used to switch between some predefined alternatives.

```
model Filter
  parameter Integer order(min=1,max=3) = 1;
  FirstOrder filter1(...) if order==1;
  SecondOrder filter2(...) if order==2;
  ...
```

CONDITIONAL COMPONENTS

```
model Pipe
    parameter Boolean useStatic = true "true if static pipe is used";
    .Modelica.Fluid.Pipes.StaticPipe staticPipe if useStatic annotation(•••);
    .Modelica.Fluid.Pipes.DynamicPipe dynamicPipe if not useStatic annotation(•••);
    .Modelica.Fluid.Interfaces.FluidPort_a port_a annotation(•••);
    .Modelica.Fluid.Interfaces.FluidPort_b port_b annotation(•••);
```



Connect statements to unused components are automatically removed

REPLACEABLE VS. CONDITIONAL

- Replaceable
  - Unlimited/not predefined set of choices
- Conditional
  - Choice is controlled with parameter that can be propagated
- There is overlap
  - For a limited and predefined set of choices
  - When both effects are wanted

```
replaceable Sine source if enable_source;
```

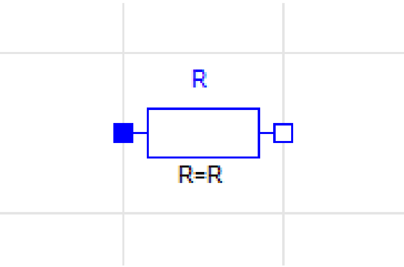
CREATING RECONFIGURABLE MODELS

ARRAYS OF COMPONENTS

ARRAYS OF COMPONENTS

To define an array:

```
Modelica.Electrical.Analog.Basic.Resistor R[10]
```



- Access an array element:

  ```
  R[i]
  ```

- Access anything inside a particular array element

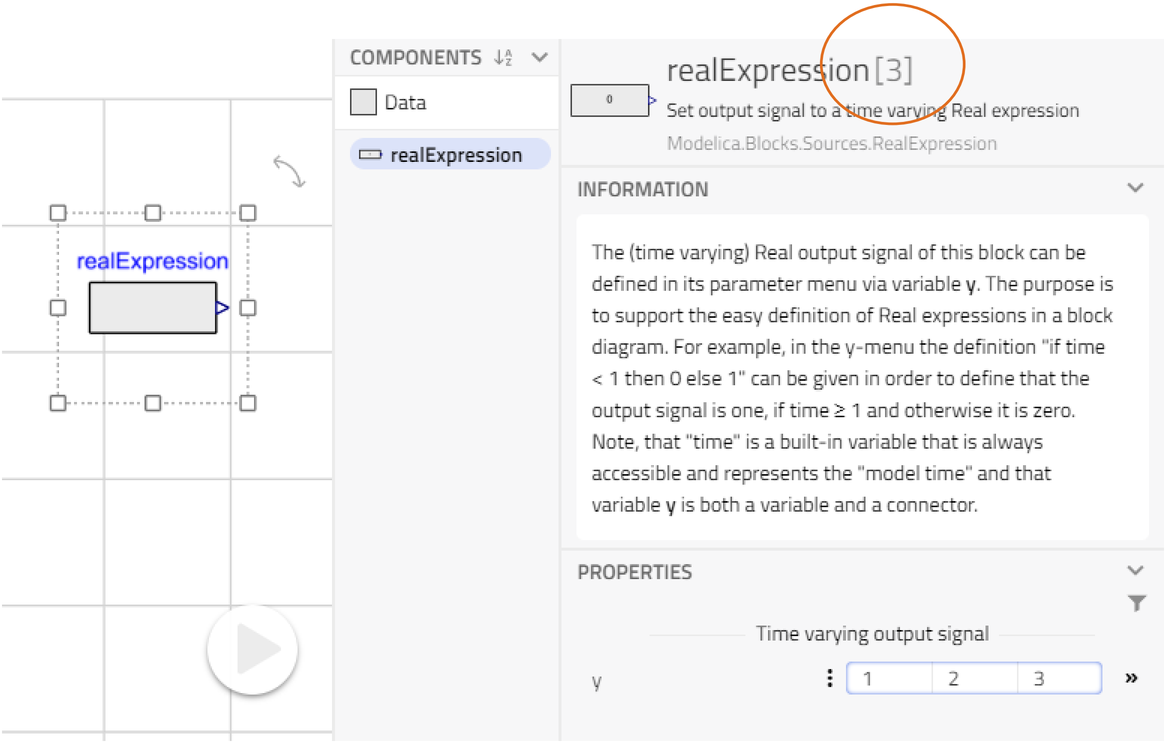
  ```
  R[i].R    (parameter R ”Resistance”)
  ```

- Setting parameter values to all array elements at once

  ```
  Electrical.Analog.Basic.Resistor R[10](each R=100);
  Electrical.Analog.Basic.Resistor R[10](R={1,2,3,4,5,6,7,8,9,10});
  ```

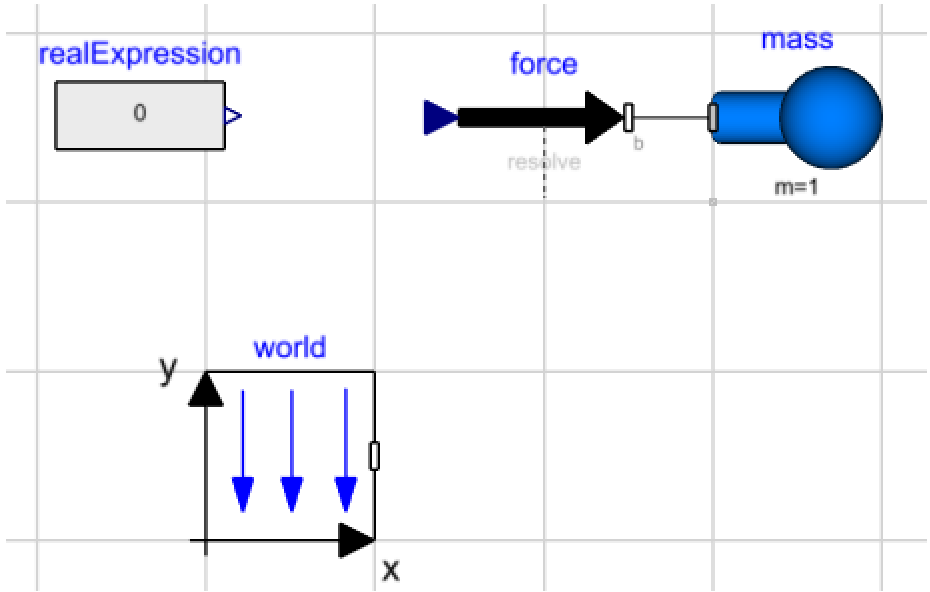
ARRAYS OF COMPONENTS

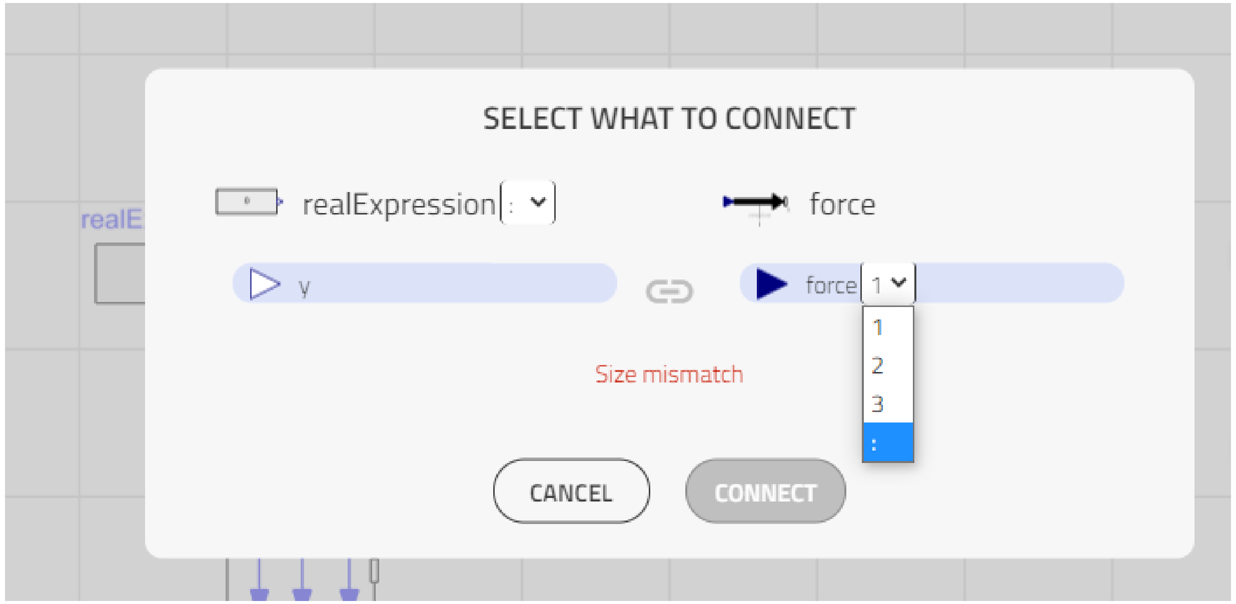
- See array size in component name
- Edit size in code layer



ARRAYS OF COMPONENTS

- Connecting arrays promts a special interface:



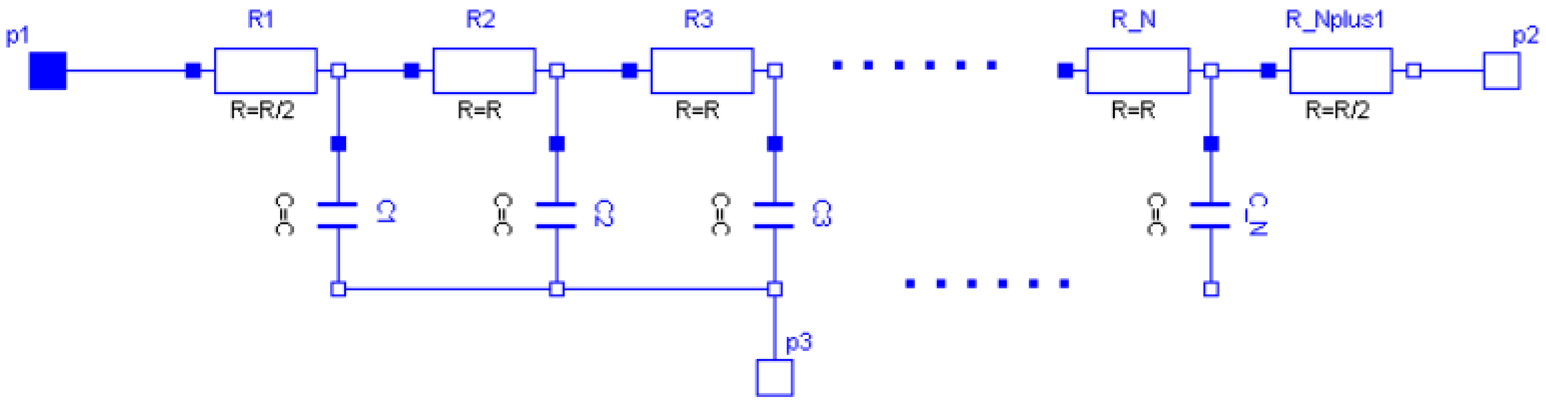


- [:] is used when you select all components
- You can choose individual connectors
- UI feeds back if size is wrong

ARRAYS OF COMPONENTS

- Arrays of components works just as arrays of real numbers
- Allows for e.g. discretization of PDE like problems as an electrical line with losses
- Example: Modelica.Electrical.Analog.Lines.ULine

```
Modelica.Electrical.Analog.Basic.Resistor R[10]
equation
 for i in 1:9 loop
   connect(R[i].p, R[i+1].n); // connecting the resistors
   ...
 end for;
```



ARRAYS OF COMPONENTS

- Access an array element:

```
R[i]
```

- Access anything inside a particular array element

```
R[i].R    (parameter R "Resistance")
```

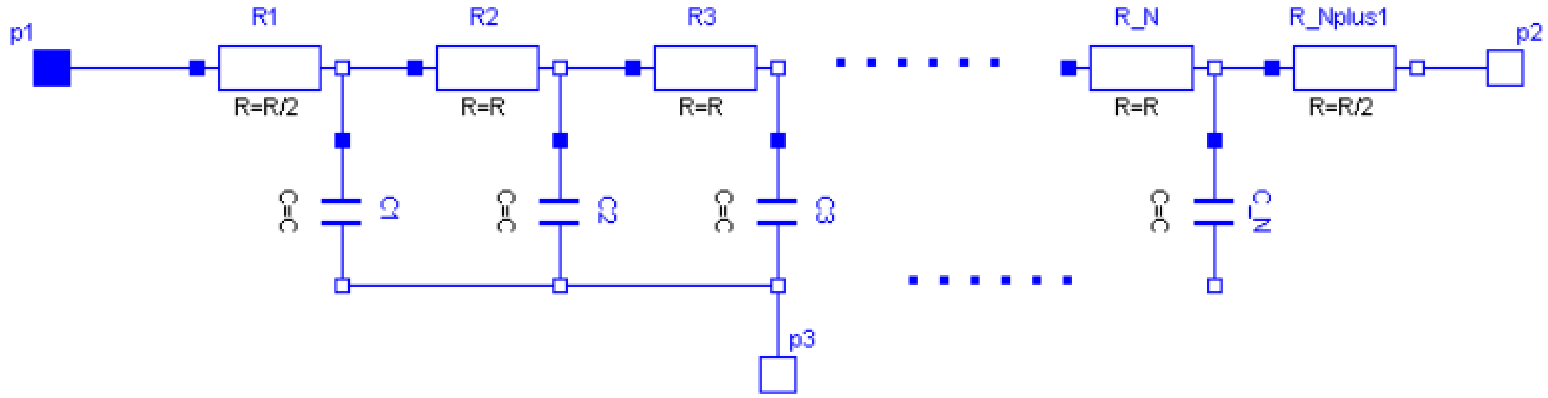
- Setting parameter values to all array elements at once

```
Modelica.Electrical.Analog.Basic.Resistor R[10](each R=100);
```

ARRAYS OF COMPONENTS

- Principle architecture of Modelica.Electrical.Analog.Lines.ULine

```
model ULine "Lossy RC Line"
  import Modelica.Electrical.Analog;
  Analog.Interfaces.Pin p, n;
  parameter Integer N(final min=1) = 1 "Number of lumped segments";
  parameter Real r = 1 "Resistance per meter";
  parameter Real c = 1 "Capacitance per meter";
  parameter Real L = 1 "Length of line";
protected
  Analog.Basic.Resistor  Res[N + 1](each R=r*L/(N + 1));
  Analog.Basic.Capacitor C[N]     (each C=c*L/N);
  Analog.Basic.Ground    g;
equation
  connect(p, Res[1].p);
  for i in 1:N loop
    connect(Res[i].n, Res[i + 1].p);
    connect(Res[i].n, C[i].p);
    connect(C[i].n, g);
  end for;
  connect(Res[N + 1].n, n);
end ULine;
```



CREATING RECONFIGURABLE MODELS

REPLACEABLE FUNCTIONS

REPLACEABLE FUNCTIONS

- Example:
  Need to calculate the volume of an object

  Case 1: a sphere
  $V = 4*pi*r^3/3$

  Case 2: cylinder
  $V = pi*r^2*L$

Solution: Use replaceable functions!

- Let you reconfigure input data and calculation of data used in a model

Modelon

REPLACEABLE FUNCTIONS

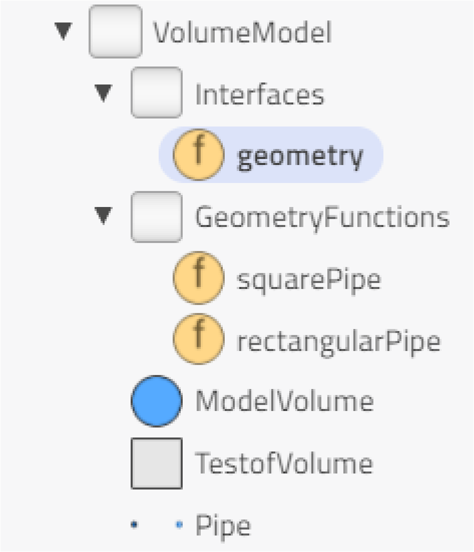
- Define a replaceable function:

```
model ModelVolume
  replaceable function V_cal = VolumeModel.GeometryFunctions.squarePipe
    constrainedby VolumeModel.Interfaces.geometry annotation (choicesAllMatching);
protected
  parameter Modelica.SIunits.Volume V=V_cal();
  annotation (•••);
end ModelVolume;
```

- By setting a constraining class, we make sure that only functions returning a calculated volume can be used.
- Using choicesAllMatching is good modelica practice but is handled automatically by Modelon Impact.

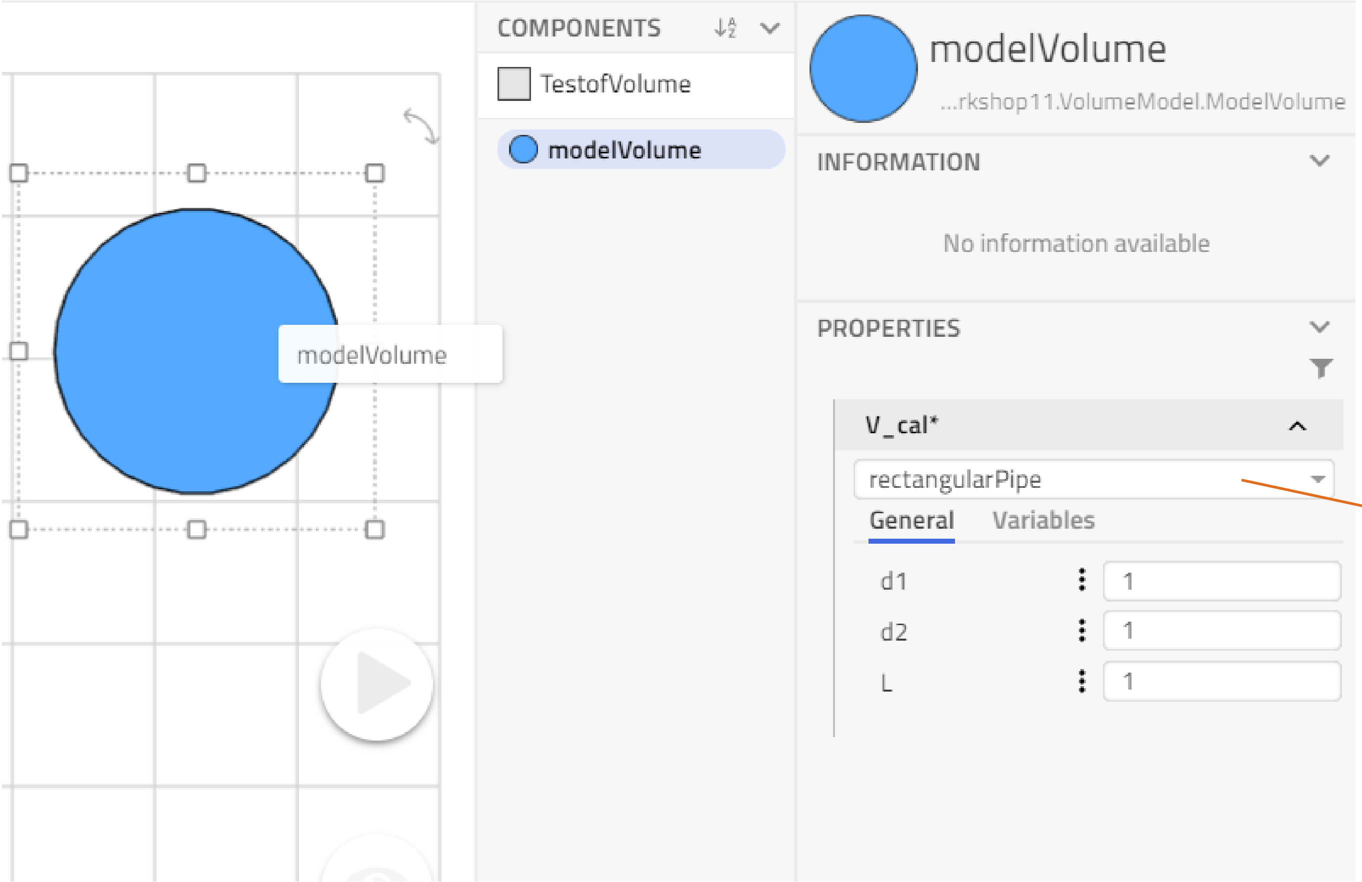
REPLACEABLE FUNCTIONS

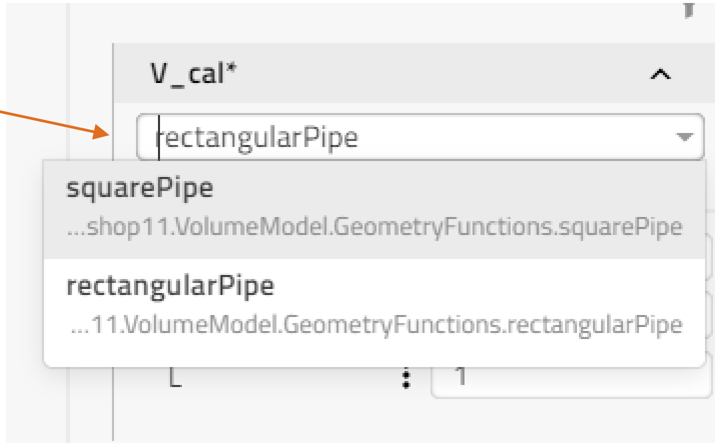
- Define the interface class and functions




```
package Interfaces
  partial function geometry "interface function to calculate volume"
    output Modelica.SIunits.Volume V "volume";
  end geometry;

end Interfaces;

package GeometryFunctions
  function squarePipe
    extends VolumeModel.Interfaces.geometry;
    input Real d=1 annotation (•••);
    input Real L=1 annotation (•••);
  algorithm
    V := L*d*d;
  end squarePipe;

  function rectangularPipe
    extends VolumeModel.Interfaces.geometry;
    input Real d1=1 annotation (•••);
    input Real d2=1 annotation (•••);
    input Real L=1 annotation (•••);
  algorithm
    V := L*d1*d2;
  end rectangularPipe;
end GeometryFunctions;
```

REPLACEABLE FUNCTIONS





WORKSHOP 4.1

In this workshop you will:

- Create a system architecture based on templates and interfaces
- Use component arrays to create a discretized model