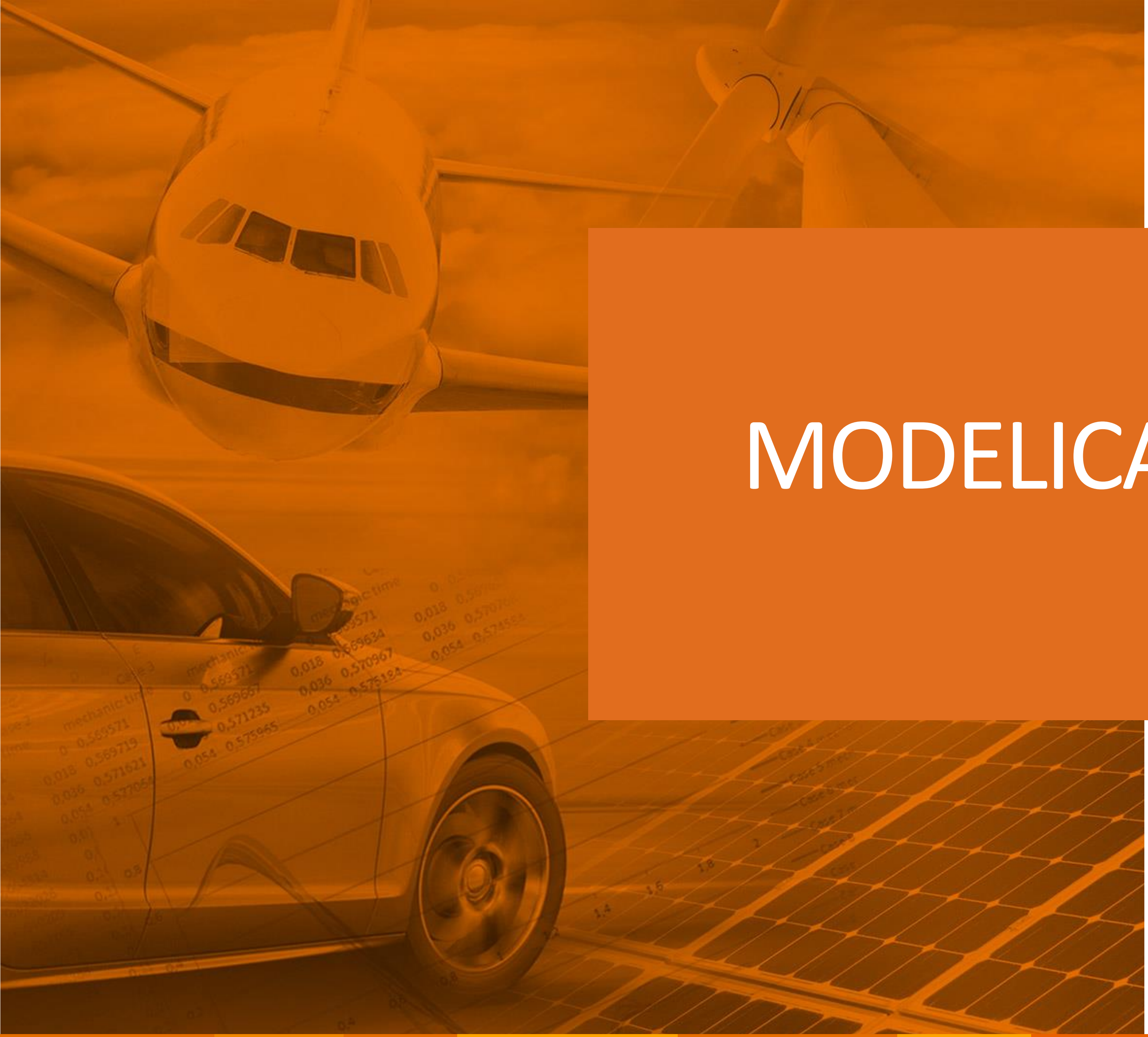


MODELICA LANGUAGE

ADVANCED FEATURES

Lecture 3.2

Modelon

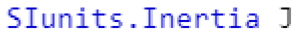
OVERVIEW

- Encapsulation
- Advanced connectors
  - Stream connectors
  - Overdetermined connectors
  - Bus connector

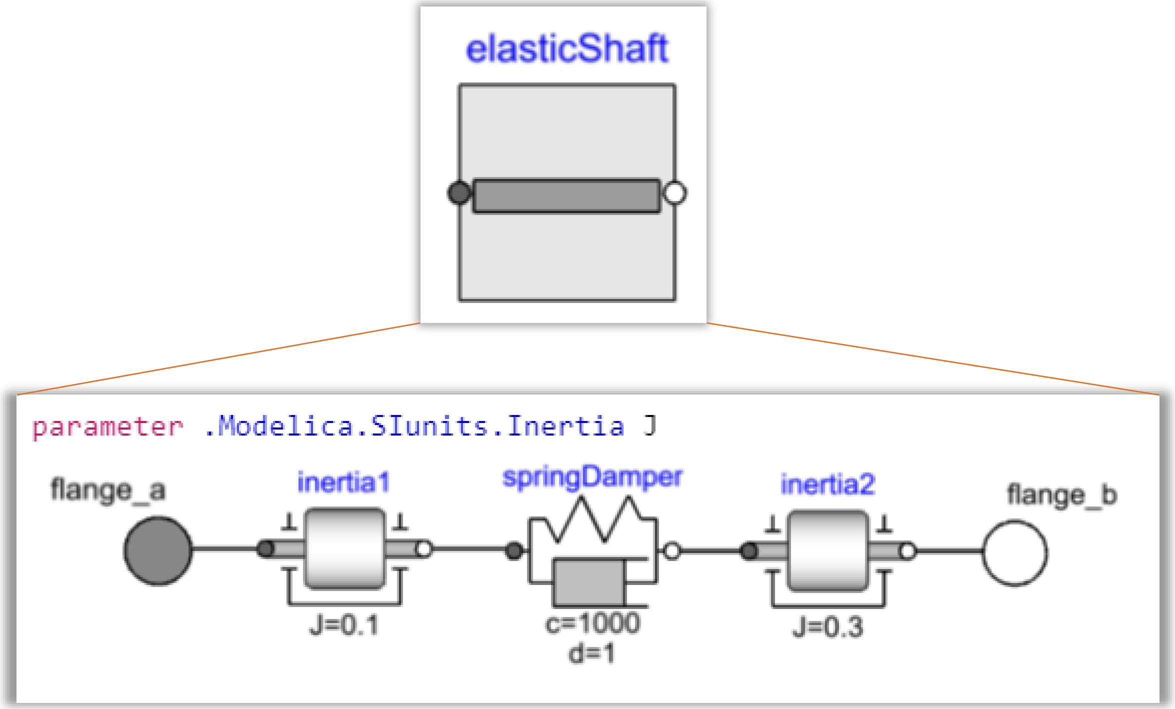
ENCAPSULATION

ENCAPSULATION

- **J** occurs several times in this model
  - J in elasticShaft
  - J in inertia1
  - J in inertia2
- These are individual instances:

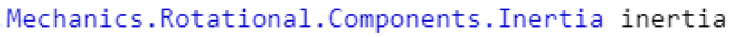


```
SIunits.Inertia J
```



```
parameter .Modelica.SIunits.Inertia J
```

- inertia1 and inertia2 are instances of a class:

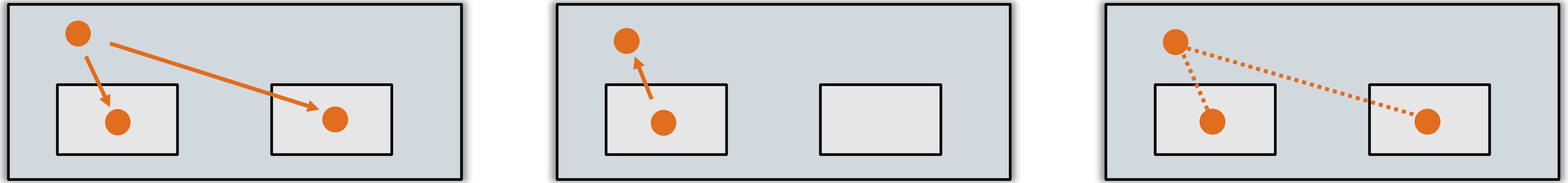


```
Mechanics.Rotational.Components.Inertia inertia
```

- All variable and parameters in inertia1 and inertia2 are encapsulated within that class.
- In order to pass information in or out of the class you need to “break” the encapsulation.

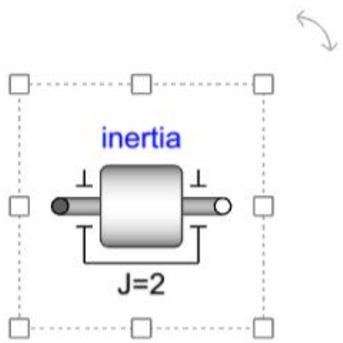
ENCAPSULATION

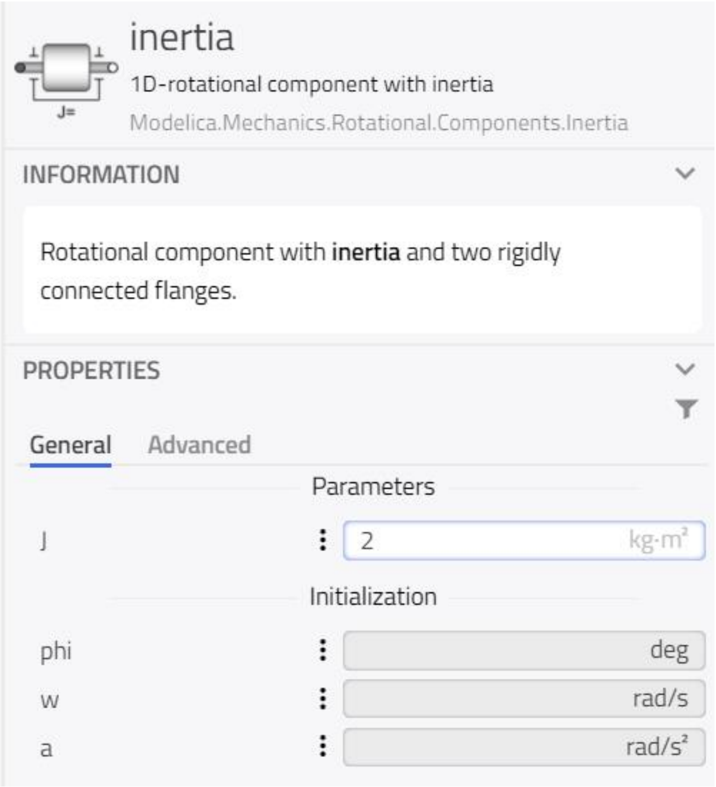
- There are three standard methods for “breaking” the encapsulation
  - Modifiers (to change an instance default parameter values)
  - Dot notation (to retrieve values from within an instance)
  - Inner/outer (creates a global object, centralized place to retrieve information)



BREAKING ENCAPSULATION: MODIFIERS

- Set properties of a component from the container level
  1. Double click on component in diagram layer of container class
  2. Fill in parameter dialog (Properties), here, parameter **J** receives value 2





inertia

1D-rotational component with inertia

Modelica.Mechanics.Rotational.Components.Inertia

INFORMATION

Rotational component with **inertia** and two rigidly connected flanges.

PROPERTIES

General Advanced

Parameters

J 2 kg·m²

Initialization

phi deg

w rad/s

a rad/s²

  - Modelica text view:

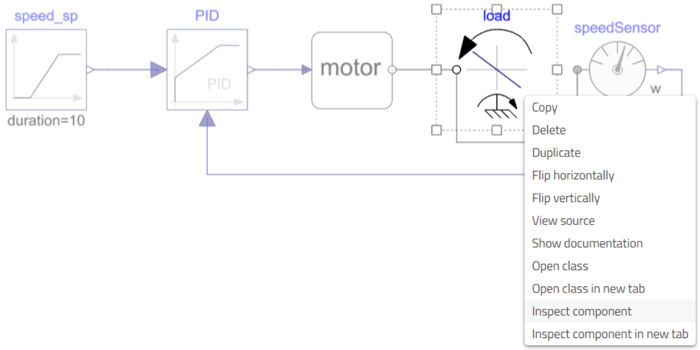
```
.Modelica.Mechanics.Rotational.Components.Inertia inertia(J = 2) annotation(•••);
```

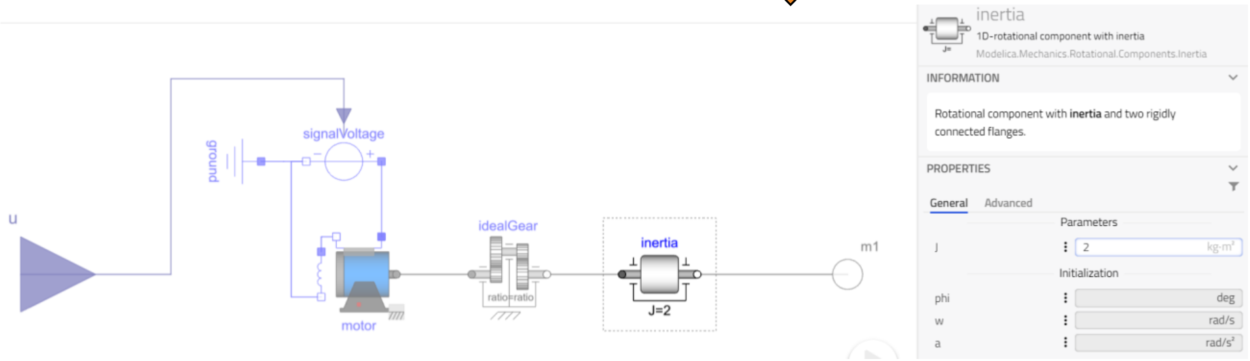
BREAKING ENCAPSULATION: MODIFIERS

- Modifications are valid only within the container class they have been performed in
  - Example: A model **ControlledMotor,** which contains a component **motor,** which contains a component **inertia**
  - Modify parameter **J** in **inertia** two levels below
    1. Navigate down one level into component **motor**, using *Inspect Component*
    2. Double-click on component **inertia** to bring up its parameter dialog
    3. set parameter **J**
  - Modelica text view:

```
model ControlledMotor
  ElectricalMotor motor(inertia(J=2));
...
```

  - Modification only valid for this specific motor component, not the motor class in general





inertia

1D-rotational component with inertia

Modelica.Mechanics.Rotational.Components.Inertia

INFORMATION

Rotational component with **inertia** and two rigidly connected flanges.

PROPERTIES

General Advanced

Parameters

J 2 kg·m²

Initialization

phi deg

w rad/s

a rad/s²

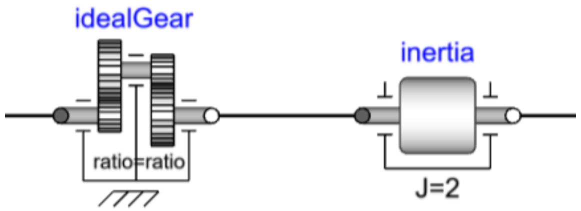
BREAKING ENCAPSULATION: DOT-NOTATION

- Retrieving components (incl. variables) further down the hierarchy is done using the component names separated by dots:
  - **`motor.inertia.flange_a.tau`** translates into: the component (or variable) **`tau`** inside the component **`flange_a`** inside the component **`inertia`** inside the component **`motor`**
  - Applies especially to variables, since they are usually determined in model equations at the base level and retrieved to be used at a higher hierarchical level
  - Example: Add up all heat flows transferred in individual channels of a heat exchanger to give the total transferred heat

```
model HeatExchanger
  Real Q_flow ”Total heat flow rate”;
  Channel ch1 ”First channel”;
  Channel ch2 ”Second channel”;
...
equation
  Q_flow = ch1.Q_flow + ch2.Q_flow;
...
end HeatExchanger;
```

BREAKING ENCAPSULATION: DOT-NOTATION

- Connections automatically create a relationship between connector variables of connected components



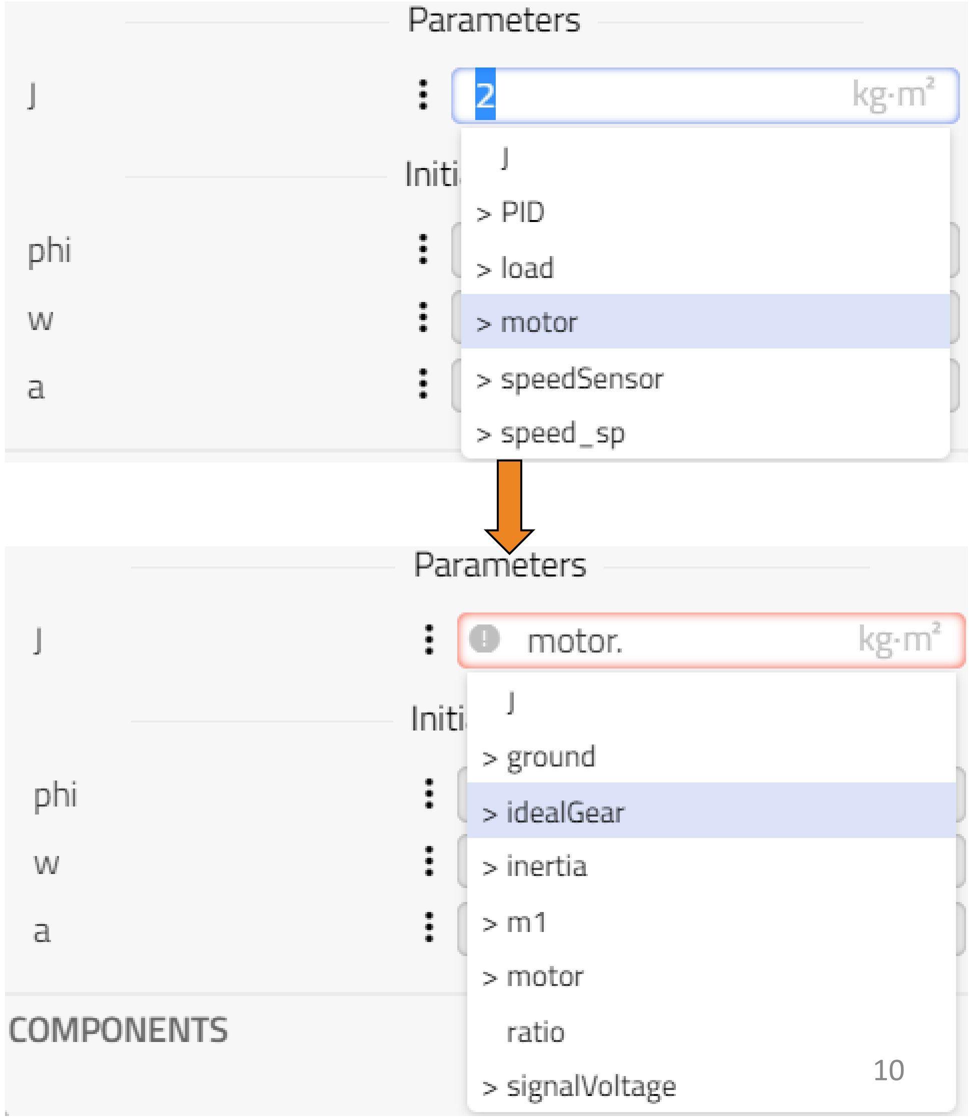
- The connector itself (e.g. **flange_a**) is a component inside a component (e.g. **inertia**)
- When drawing a connection between two components the corresponding text view is for example:

connect(idealGear.flange_b, inertia.flange_a);

- Dot-notation access of the connector components at the level, where the connection is made

BREAKING ENCAPSULATION: COMBINATIONS

- Insert dot-notation component reference as a modifier
- Retrieve parameter/variable from one component and use as modifier in another
  - Double-click on the parameter field to select current value
  - Navigate the appearing component tree to the correct parameter/variable
  - The reference appears in dot-notation in the parameter field
  - If names are known, the reference can be written manually
  - Bad system design, if used extensively
  - Propagate to top level and modify from there instead



Parameters

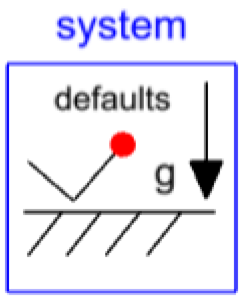
J 2 kg·m²

Parameters

J motor. kg·m²

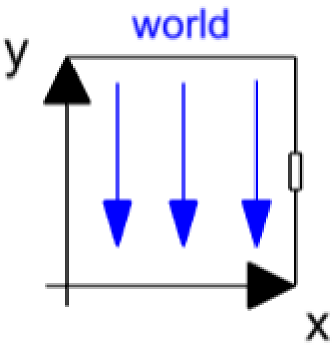
BREAKING ENCAPSULATION: INNER/OUTER

- Some type of information needs to be passed to all through the system to many components
- Examples: Ambient conditions, coordinate systems, gravity, magnetic fields
- inner/outer prefix is used in modelica to define a globally accessible object.



system component in Fluids package

- gravity, global system settings, ambient conditions



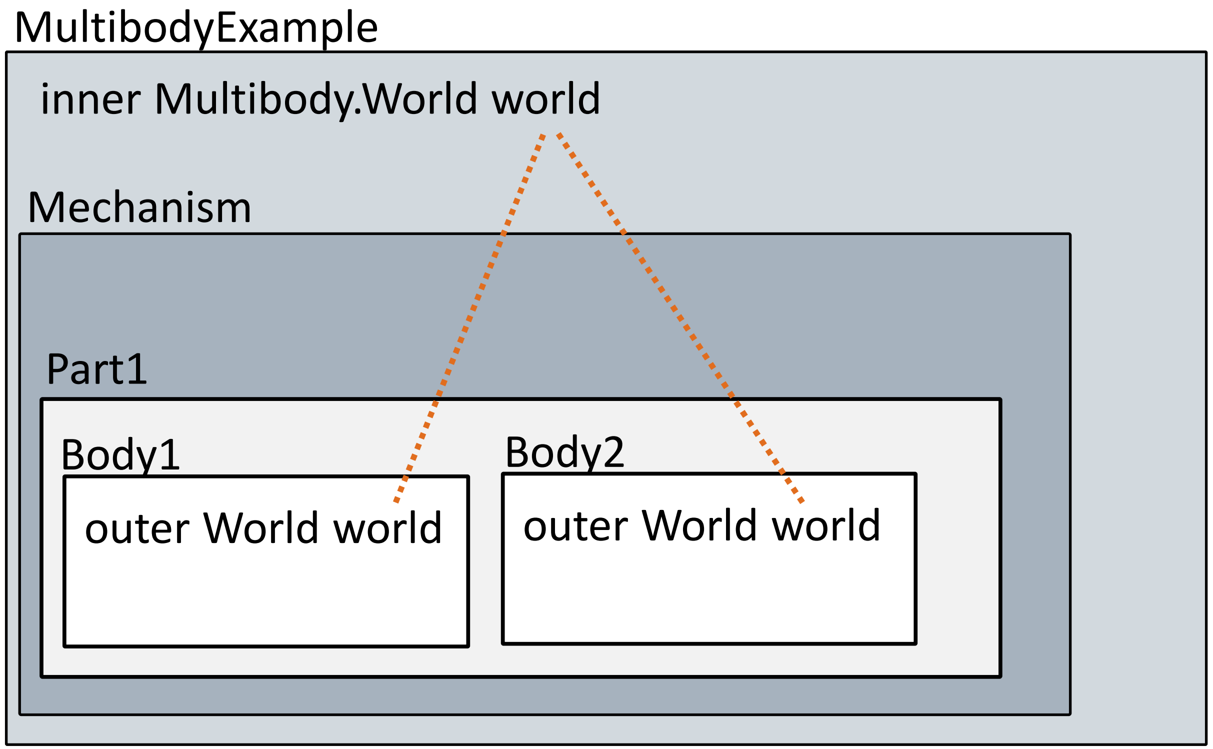
world component in Mechanics.Multibody package

- gravity, coordinate system

- **inner** keyword creates an instance which is accessible by all components at a lower hierarchical level
- Any **outer** component defined at a lower hierachical level will access the **inner** component

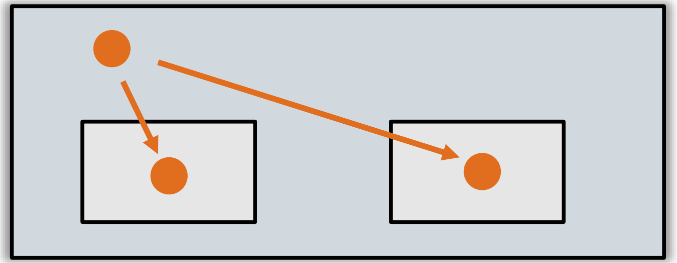
BREAKING ENCAPSULATION: INNER/OUTER

- The Body1 and Body2 instances, contains World objects with **outer** prefix, that refers to the component world with **inner** prefix in class MultibodyExample.

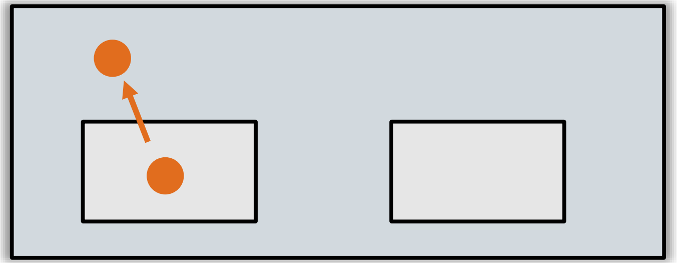


BREAK ENCAPSULATION: SUMMARY

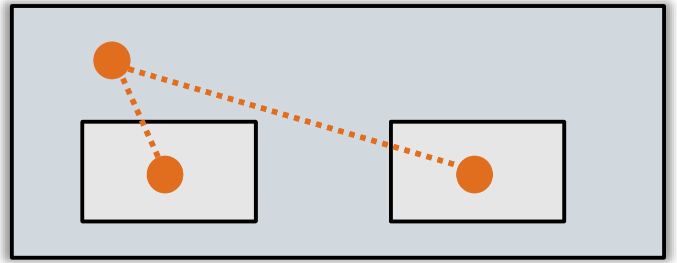
- Modifiers
  - set a variable/parameter
  - one declaration can be passed on to several sub-components
  - especially used for parameter propagation from top level of a component to sub-components



- dot-notation
  - retrieve a variable/parameter
  - access sub-component variables and components from higher level container classes



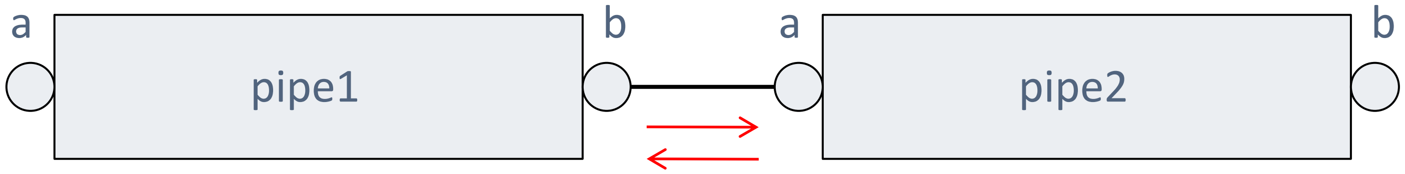
- inner/outer declarations
  - automatic equality from matching component names regardless of number of component levels



STREAM CONNECTORS

STREAMS

- Consider a system with a fluid, which flows from one component to another, both directions are possible.



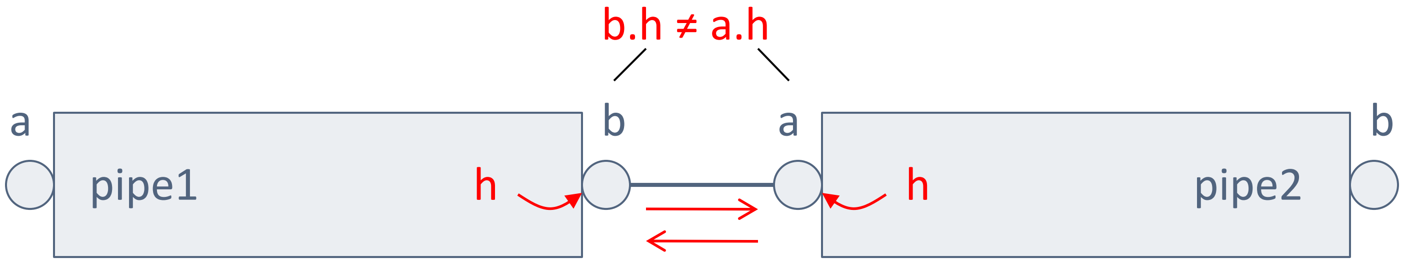
- Pressure and mass flow rate are a potential/flow variable pair, since a pressure potential drives the fluid flow.
- But what can we match properties with, that are transported by the flow, like specific enthalpy or concentrations? We could add “false” partners, like enthalpy flow rate or species flow rates.
- What happens to the properties at the connection point if the flow switches direction? They would need to switch discontinuously, not nice for model robustness.

STREAMS

- Solution
  - Stream variables in the connector

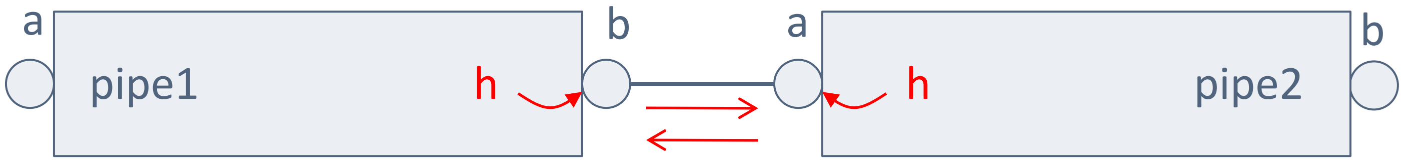
```
connector FluidPort
  Modelica.SIunits.Pressure p;
  flow Modelica.SIunits.MassFlowRate m_flow;
  stream Modelica.SIunits.SpecificEnthalpy h;
end FluidPort;
```

- one flow variable, one potential variable and an arbitrary number of stream variables
- **no** equations are generated for the stream variables, when connected



STREAMS

- If no additional equation is generated, how is then information passed across a connection?
- Operators, that can be applied to stream variables in a connector
  - **`inStream()`** – accesses the connector variable “on the other side”
    - independent of flow direction
    - e.g. inside **`pipe1`** the expression **`inStream(b.h)`** will yield the value of **`pipe2.a.h`**
  - **`actualStream()`** – accesses the upstream connector variable
    - discontinuously switching with flow direction, but yields continuous expression, if used in combination with the flow variable
    - e.g. in the energy balance of pipe1 or pipe2:
  - **`dU/dt=a.m_flow*actualStream(a.h)+b.m_flow*actualStream(b.h)`**
- This is covered in detail in Thermo-Fluid Modeling Course



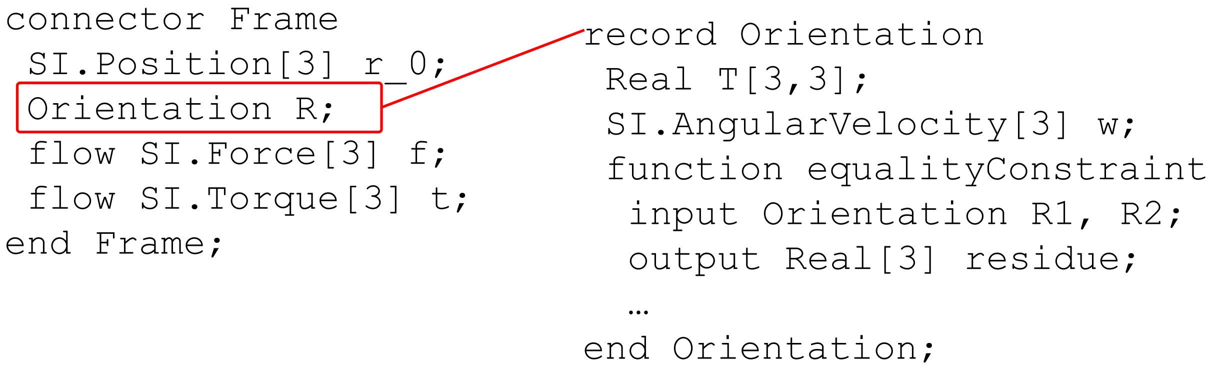
OVERDETERMINED CONNECTORS

OVERDETERMINED CONNECTORS

- Overdetermined connectors contain more connector variables than degrees of freedom needed (e.g. in the MultiBody Frame connector).
- Instead of using 3 rotational variables to describe frame orientation the connector stores the complete rotation matrix and the angular velocity vector. (3 vs 12 variables)
- This was introduced as a more efficient implementation
- To be able to compare two frames there is a need to implement a constraint function that compares two rotation objects each with 12 variables and returns a residual with length 3.

OVERDETERMINED CONNECTORS

- The redundant information needs to be presented in the connector as a record together with the constraint function:



```
connector Frame
 SI.Position[3] r_0;
 Orientation R;
 flow SI.Force[3] f;
 flow SI.Torque[3] t;
end Frame;
```

```
record Orientation
 Real T[3,3];
 SI.AngularVelocity[3] w;
 function equalityConstraint
  input Orientation R1, R2;
  output Real[3] residue;
  …
end Orientation;
```

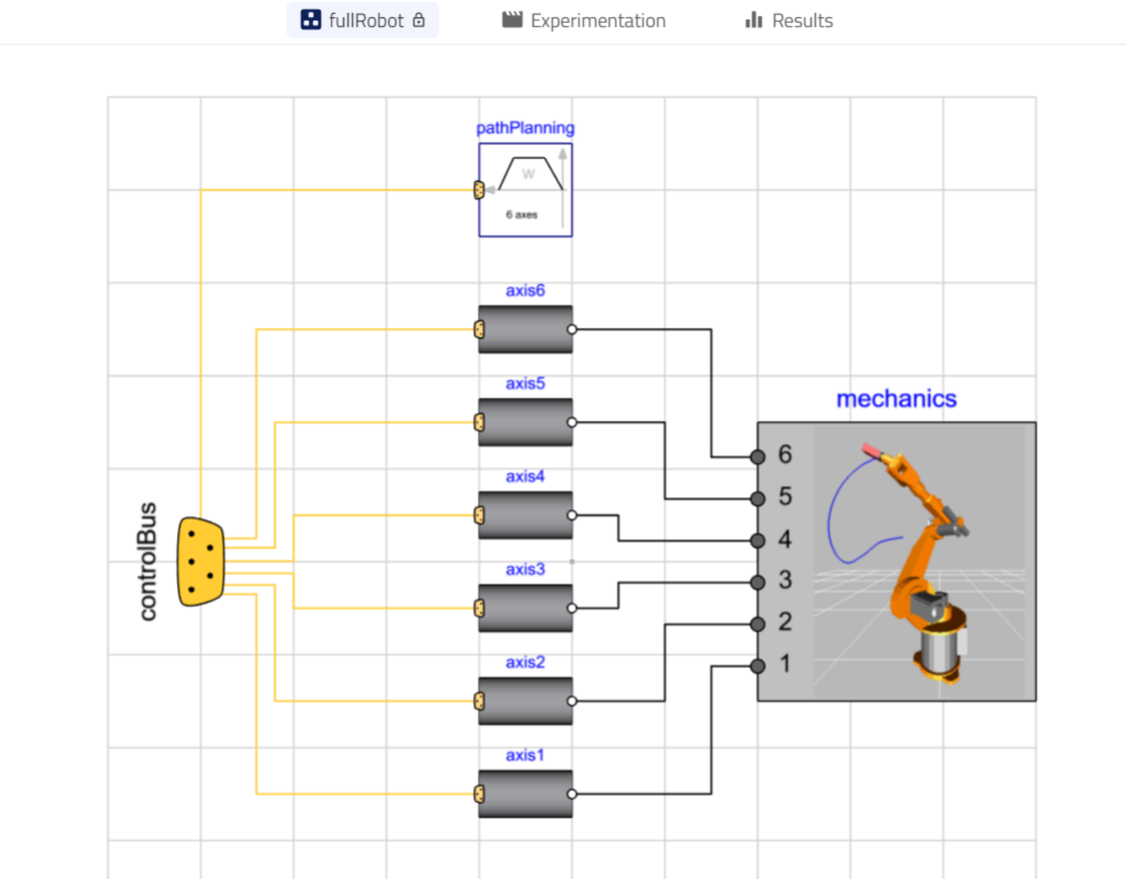
- The output of the equalityConstraint function is a Real[3] which matches the size of the torque. (The position and force match as they are)
- Note that such functions tend to generate nonlinear systems of equations.
- This is covered in detail in the Mechanics Modeling Course

MODELICA

SIGNAL BUS - EXPANDABLE CONNECTOR

EXPANDABLE CONNECTOR

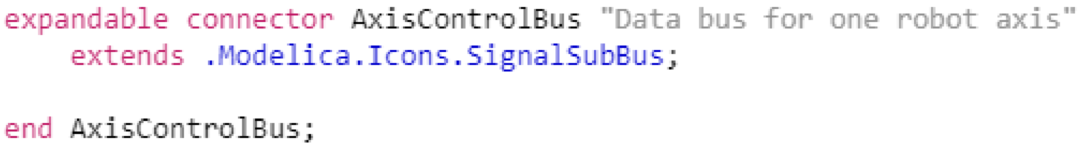
- Connector that will change its content depending on what’s connected to it.
- Used to define signal buses



EXPANDABLE CONNECTOR

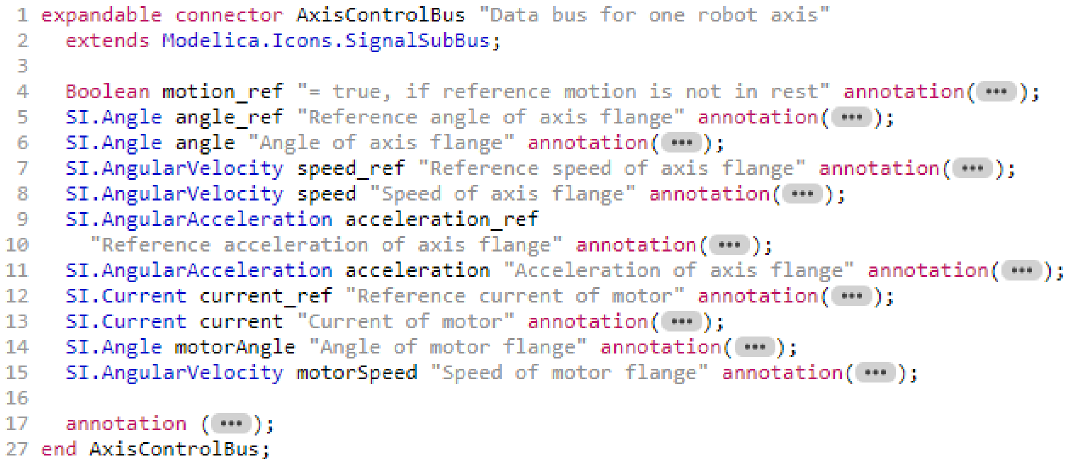
Uses Modelica keyword `expandable`

Can be empty:



```
expandable connector AxisControlBus "Data bus for one robot axis"
    extends .Modelica.Icons.SignalSubBus;

end AxisControlBus;
```

Or have predefined set signals:



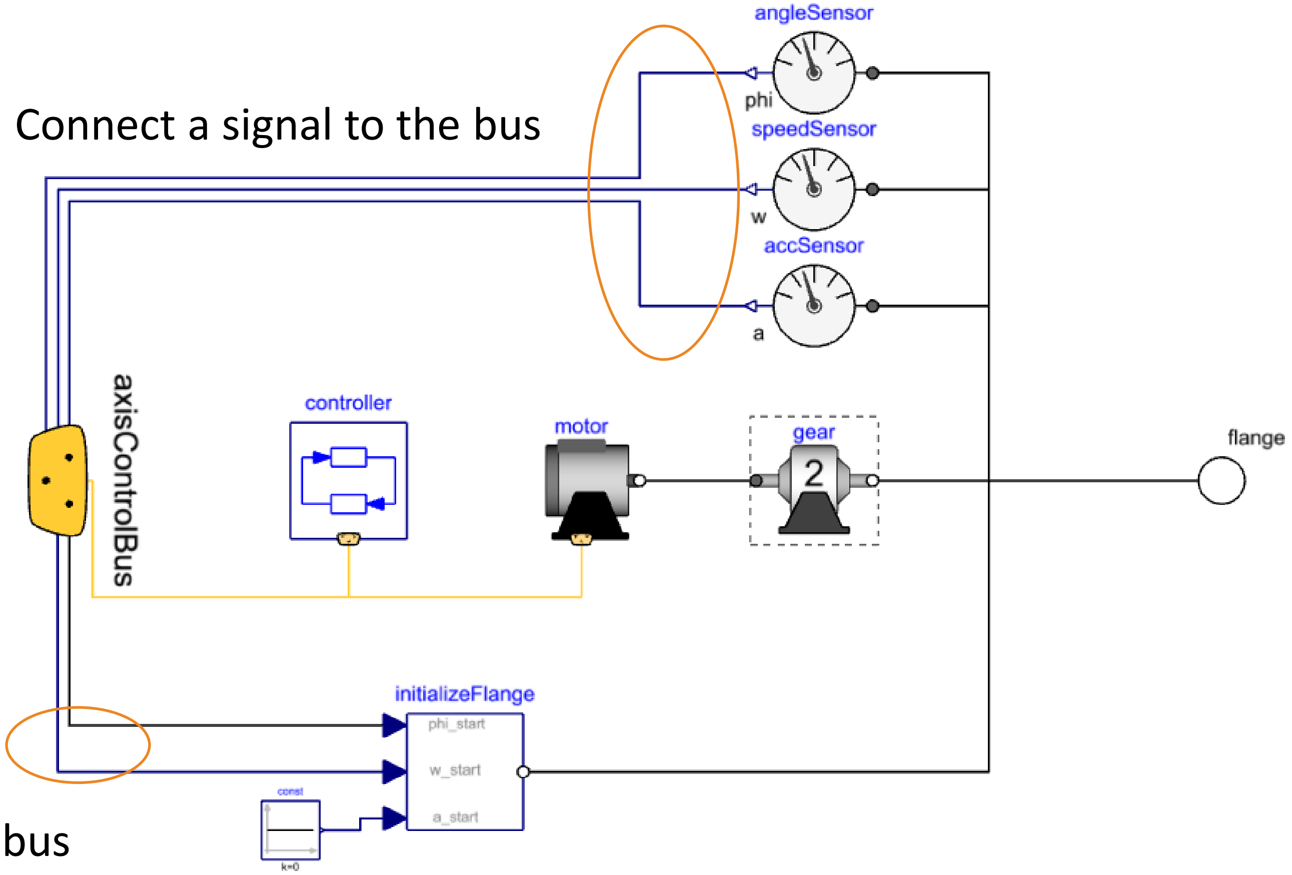
```
expandable connector AxisControlBus "Data bus for one robot axis"
  extends Modelica.Icons.SignalSubBus;

  Boolean motion_ref "= true, if reference motion is not in rest" annotation(•••);
  SI.Angle angle_ref "Reference angle of axis flange" annotation(•••);
  SI.Angle angle "Angle of axis flange" annotation(•••);
  SI.AngularVelocity speed_ref "Reference speed of axis flange" annotation(•••);
  SI.AngularVelocity speed "Speed of axis flange" annotation(•••);
  SI.AngularAcceleration acceleration_ref
    "Reference acceleration of axis flange" annotation(•••);
  SI.AngularAcceleration acceleration "Acceleration of axis flange" annotation(•••);
  SI.Current current_ref "Reference current of motor" annotation(•••);
  SI.Current current "Current of motor" annotation(•••);
  SI.Angle motorAngle "Angle of motor flange" annotation(•••);
  SI.AngularVelocity motorSpeed "Speed of motor flange" annotation(•••);

  annotation (•••);
end AxisControlBus;
```

Predefined signals guide the user!

CONNECTIONS

- Causality is defined when connecting components:

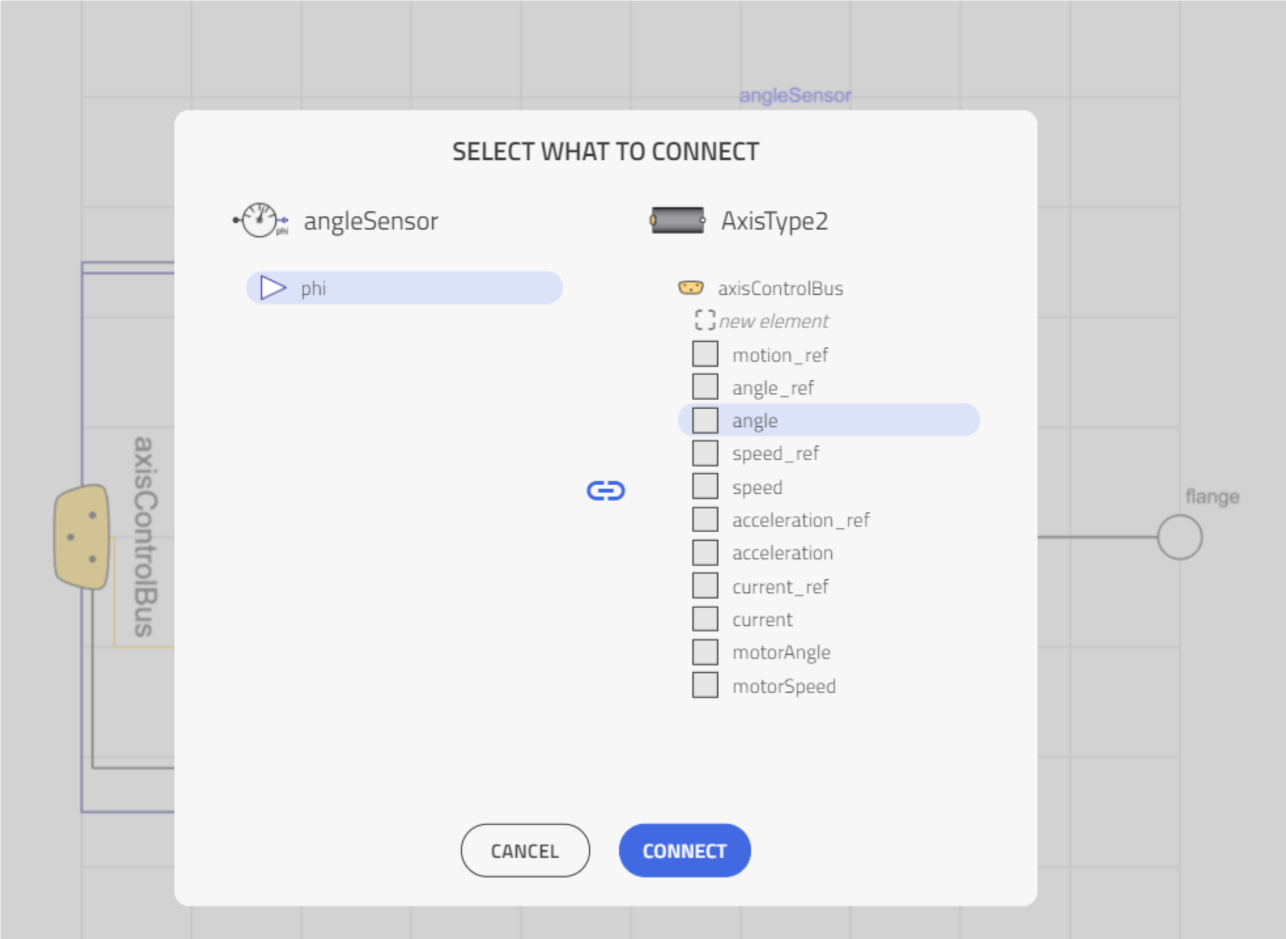
Connect a signal to the bus



Retrieve a signal from the bus

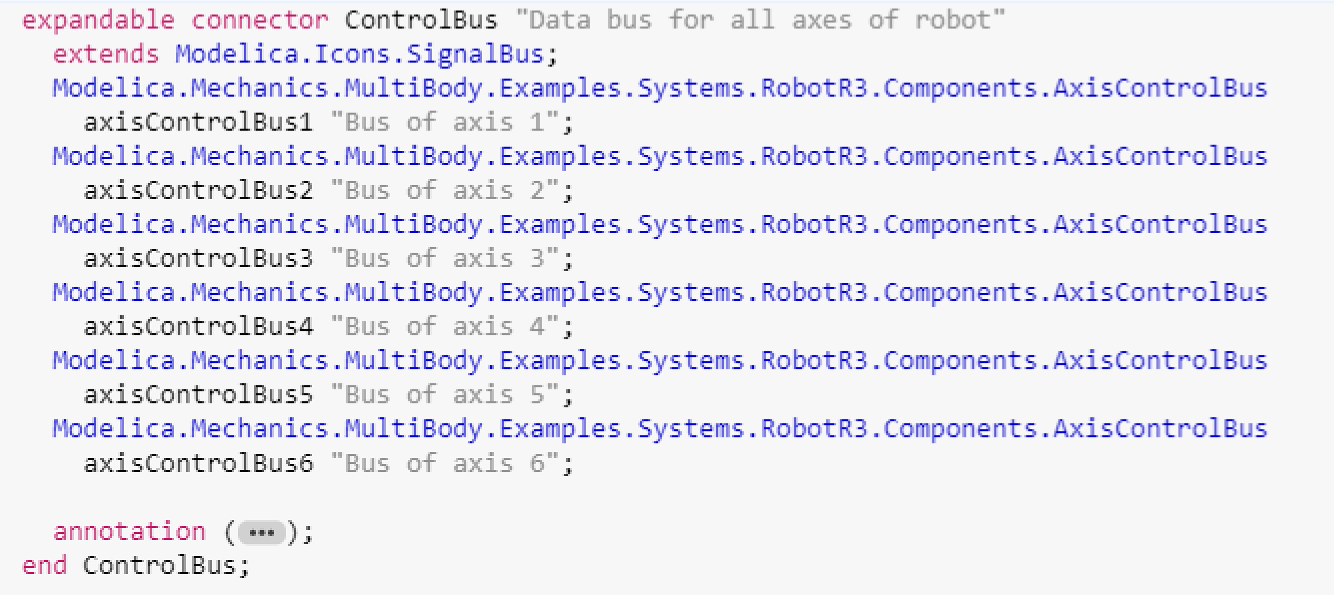
CONNECTIONS

- Connecting angleSensor to the bus:



HIERARCHICAL BUSES

- A bus connector can be composed of several sub-buses



```
expandable connector ControlBus "Data bus for all axes of robot"
  extends Modelica.Icons.SignalBus;
  Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus
    axisControlBus1 "Bus of axis 1";
  Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus
    axisControlBus2 "Bus of axis 2";
  Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus
    axisControlBus3 "Bus of axis 3";
  Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus
    axisControlBus4 "Bus of axis 4";
  Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus
    axisControlBus5 "Bus of axis 5";
  Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.Components.AxisControlBus
    axisControlBus6 "Bus of axis 6";

  annotation (•••);
end ControlBus;
```

- They don’t have to be of same type

WORKSHOP 3.2

- In this workshop you will:
  - Create a model of a solar collector
  - Create a connector and parameter interface
  - Implement equations in the code editor
  - Test the component
  - Integrate the solar collector in a system model