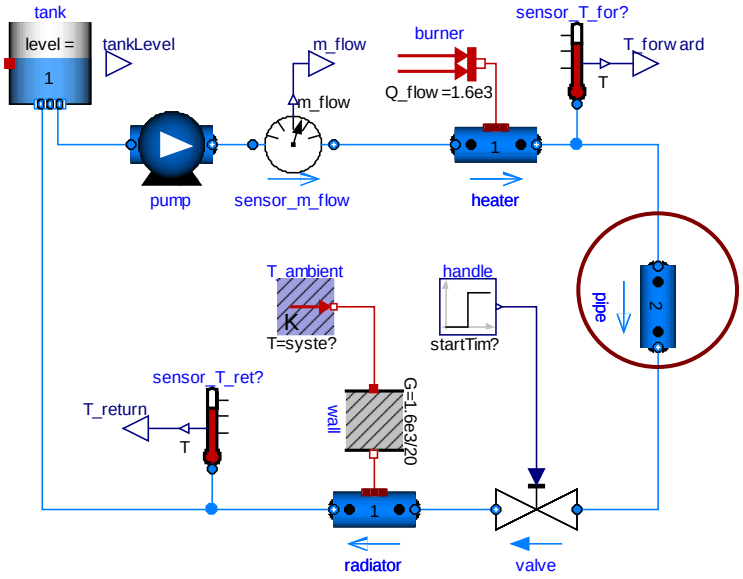
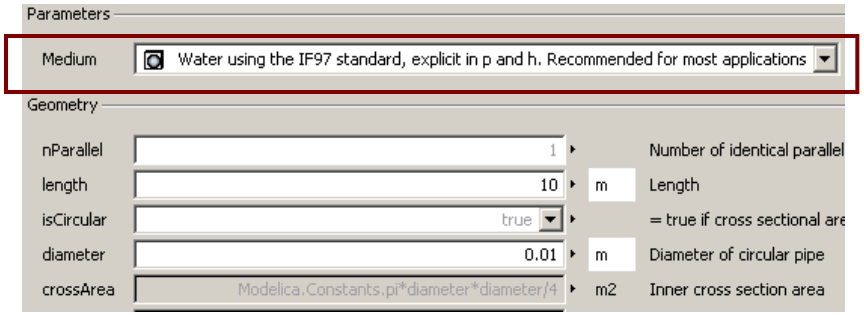
Virtual Physics
Equation-Based Modeling

TUM, December 02, 2014

Higher Level Modeling Tasks: Better Parameterization



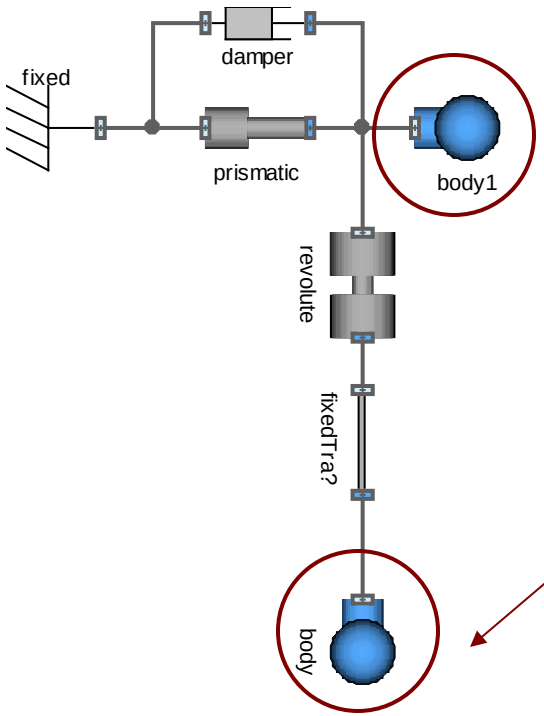


Dr. Dirk Zimmer

German Aerospace Center (DLR), Robotics and Mechatronics Centre

Outer Models

- So far, all our body components contained a parameter for the gravitational acceleration:



We have to set the gravitational acceleration twice, although it is actually a global constant.

Outer Models

- Like in a programming language, there seems to be the need for global parameters or global model variables.
- To this end, Modelica offers the concept of inner/outer models.
- A sub-model can be declared as outer.
- This means that this sub-model is not an actual component of this model but declared somewhere else in the complete system.

```
model Body

  Interfaces.Frame_a frame_a;

  parameter SI.Mass m;
  parameter SI.Inertia I;

  outer World world;

  SI.Force f[2] "force";
  SI.Position r[2] "transl. position";
  […]

  parameter Boolean
   animate = world.animation;
 […]

equation
  […]
  //Newton's law
  f = {frame_a.fx, frame_a.fy};
  f + m*world.g = m*a;
  frame_a.t = I*z;
end Body;
```

Outer Models

- Once we have declared such an outer model, we can access its parameters.
- Fortunately, the “world” model contains a parameter for gravity acceleration and for animation.

```
model Body

  Interfaces.Frame_a frame_a;

  parameter SI.Mass m;
  parameter SI.Inertia I;

  outer World world;

  SI.Force f[2] "force";
  SI.Position r[2] "transl. position";
  […]

  parameter Boolean
   animate = world.animation;
 […]

equation
  […]
  //Newton's law
  f = {frame_a.fx, frame_a.fy};
  f + m*world.g = m*a;
  frame_a.t = I*z;
end Body;
```

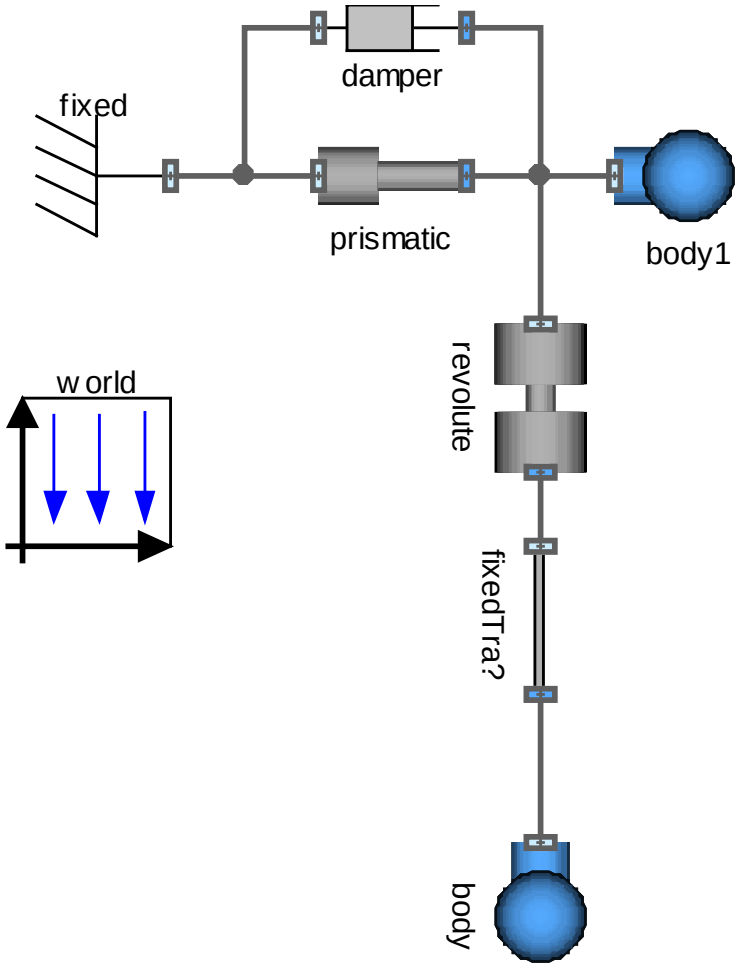
Outer Models

- Here we see the actual world model. It is a simple container for parameters of global use.
  - Gravity Acceleration
  - Animation

```
model World

  parameter SI.Acceleration
   g[2] = {0,-9.81}
   "Gravity Accleration";

  parameter Boolean
    animation = true
    "Enable Animation as default
    for components";

  annotation(
    defaultComponentPrefixes="inner",
    defaultComponentName="world
  );

end World;
```

Outer Models

- Now the body components in our system demand for an outer world model.
- Hence, we have to declare one in our system.
- Now we can globally change the gravity acceleration by setting the parameter in the world model.



Outer Models

- However, a normal declaration is not sufficient.
- The model must be declared as inner.
- Also, the name of the component must precisely match.
- Hence the following annotation pattern is used for most inner/outer models:
  annotation(
  defaultComponentPrefix = “inner”,
  defaultComponentName = “world”

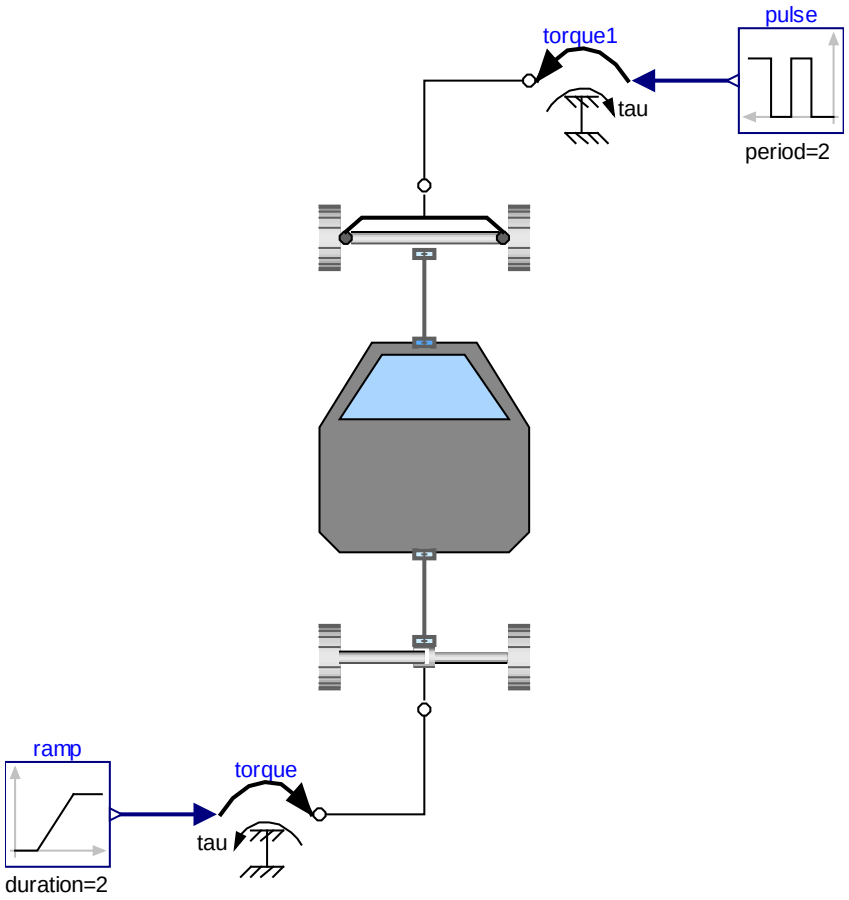
```
model CraneCrabWorld

  inner World world;

  Parts.Body2 body(I=0.1,m=0.5);
  Joints.Revolute revolute(
    initialize=true, phi_start=-2.7);
  Parts.FixedTranslation
    fixedTranslation(r={0,-1});
  Parts.Fixed fixed;
  Joints.Prismatic prismatic(
   r={1,0},initialize=true);

[…]

equation

  connect(…)
  […]

end CraneCrab Worl
```

Outer Models

- When an outer model is used, a Modelica translator will search for a component with the desired name upwards in the component hierarchy.
- When a component with matching name is found, it must be declared as inner and it must be of compatible type.
- If no component is found, a warning is issued and a default inner model is instantiated at the top level.
- Outer models can be used across several layers in the component hierarchy.

Replaceable Components

- Let us wrap the simple two-track car model:



Replaceable Components

- Maybe, we want to try out different chassis.
- The best solution would be if the chassis component is a parameter of the complete car model.
- But parameters must be constant values and cannot contain time-dependent variables.
- However, it is possible to declare a model as replaceable

```
model TwoTrackCar

  replaceable VehicleComponents.SimpleChassis chassis(…)
  […]
```

Replaceable Components

- When using the TwoTrackCar model, we can now redeclare the chassis model and replace it by another one.
- Only components that have been marked as replaceable can be redeclared.
- The redeclaration can be performed in the parameter menu of Dymola as well.

```
model ExampleSystem

  TwoTrackCar myCar(redeclare VehicleComp.AdvancedChassis chassis(…),…)

  […]
```

Type System

- What models can we use in order to replace the original car model?
  The new component must be “plug-compatible” to the original one.

- How is this compatibility checked?
  Modelica is using a structural type system.

- In nominal type-systems, inheritance is often used to create sub-type hierarchies (as in C++). In Modelica, this does not matter. Inheritance is only used to generate new models out of existing ones. This can be sub-types or not.
- It is possible that two models are type-compatible although they have completely disjoint implementation paths.
- It is also possible that two models are incompatible although, they are related by inheritance (“extends”).

Structural Check

- A is a **sub-type** of B iff…
  - A is equivalent to B
  - All public elements in B are contained in A and are sub-types of their counterparts in B.

- A is **plug-compatible** to B if A is a sub-type of B and A contains no additional, public input-connectors.

- This is simplified. Reality is more complex due to conditional declarations or parameterized vector/matrix sizes.

Replaceable Models

- Another application is the replacement of whole classes of models by another one.
- If we want to exchange the wheel model, we do not want to do this component-wise for each of the four wheels but once for all.
- To this end, we can declare replaceable models.

```
model SimpleFrontAxis

  replaceable model Wheel = Wheels.DryFrictionWheelJoint;

  parameter SI.Length R =  0.25 "radius of the wheel";
  […]

  Wheel WheelJointLeft(radius=R, r = r,…);

  Wheel WheelJointRight(radius=R, r = r,…);
```

Replaceable Models

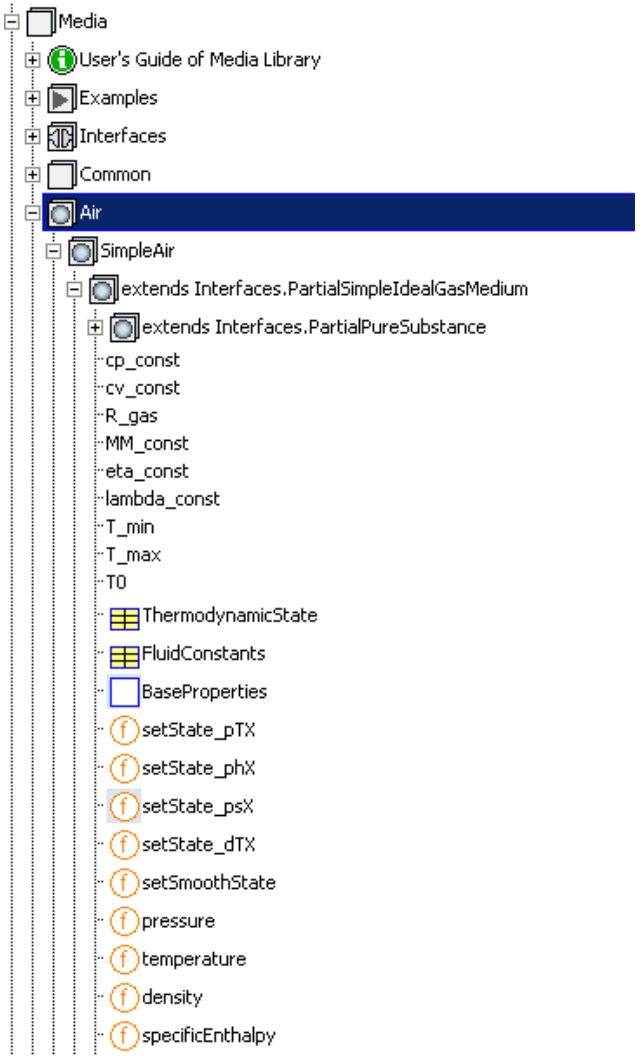
- This new model definition can then be used in order to declare new components.

- In a programming language, this pattern would be represented by type-parameters (as for templates in C++)

```
model SimpleFrontAxis

  replaceable model Wheel = Wheels.DryFrictionWheelJoint;

  parameter SI.Length R =  0.25 "radius of the wheel";
  […]

  Wheel WheelJointLeft(radius=R, r = r,…);

  Wheel WheelJointRight(radius=R, r = r,…);
```

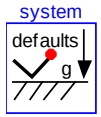
Media Models

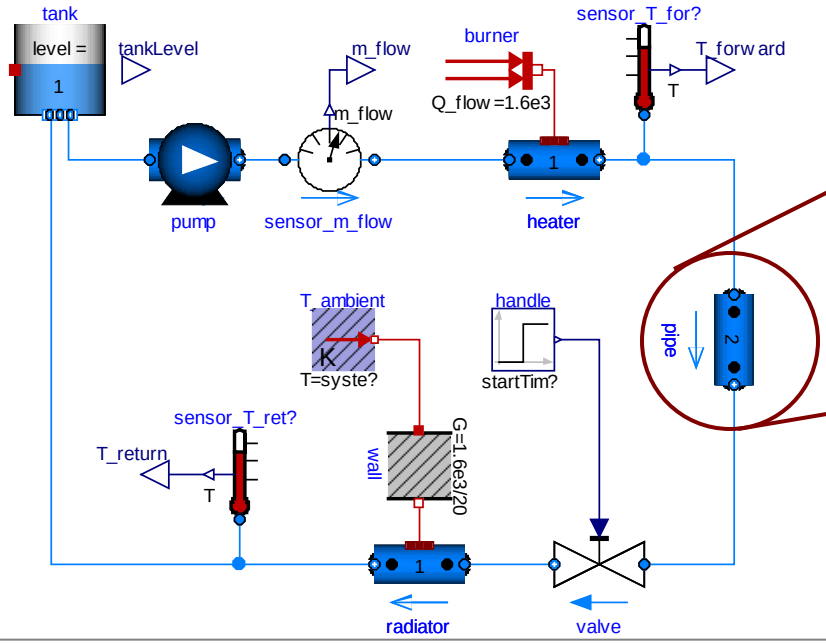
- Replaceable models are intensively used for media models.
- The thermal state of a (compressible) medium is typically stored by two variables:
  - absolute pressure
  - specific enthalpy.
- Out of these two variables other relevant variables can be computed such as:
  - temperature
  - density
  - specific entropy
  - etc…
- To this end, a package of functions and models is provided for each medium of interest.

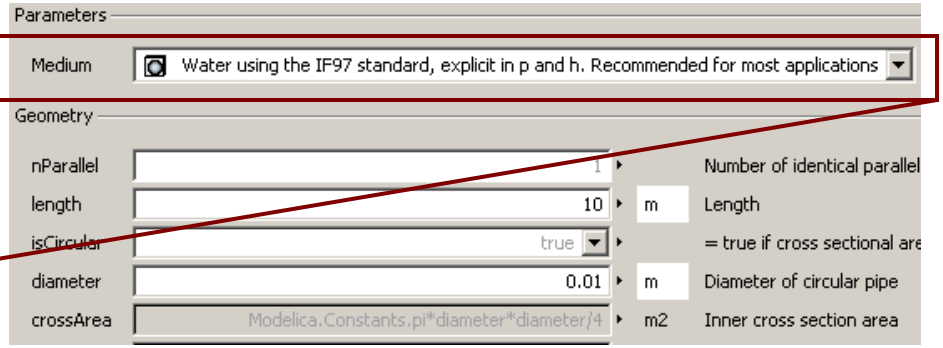


Media Models

- The fluid components (e.g. a model of a pipe) all have replaceable medium packages.
- In this way, the same components for fluid-systems can be used for different mediums.







Questions ?