Open Standards and Software for Dynamic System Simulation - Modelica

Francesco Casella

francesco.casella@polimi.it

DEIB - Politecnico di Milano – Italy

Modelica Association

Open Source Modelica Consortium

Outline

Part #1: Object-Oriented Modelling & Modelica

- Principles of Equation-Based Object-Oriented Modelling (EOOM)
- The Modelica Language
- Automatic generation of executable code
- Related standards: FMI, SSP, DCP
- Case study: satellite attitude modelling and control

Outline

Part #1: Object-Oriented Modelling & Modelica

- Principles of Equation-Based Object-Oriented Modelling (EOOM)
- The Modelica Language
- Automatic generation of executable code
- Related standards: FMI, SSP, DCP
- Case study: satellite attitude modelling and control

Part #2: The Community

- The Modelica Association and its projects
- The Open Source Modelica Consortium and OMC

Principles of Equation-Based Object-Oriented Modelling

Declarative Modelling

Models should describe how a system behaves

not how the behaviour can be computed

There are no input and output variables in real life

The best formalization of a simulation model

is more easily understood by a human

not by a computer

Principle #1: Declarative Modelling

Equation-Based modular (→ Object-Oriented) description

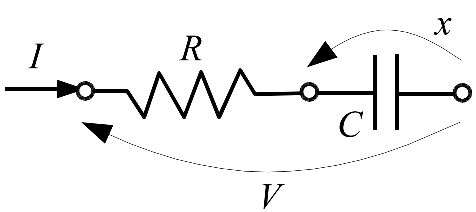
- The model of each component is described by equations
- The model is independent of the components it is connected to
- Physical connections ↔ *connection equations*

Principle #1: Declarative Modelling

Equation-Based modular (→ Object-Oriented) description

- The model of each component is described by equations
- The model is independent of the components it is connected to
- Physical connections ↔ *connection equations*

Example: RC component



$$x + RI = V$$
$$C\,\dot{x} = I$$

(DAE – declarative model)

Principle #1: Declarative Modelling

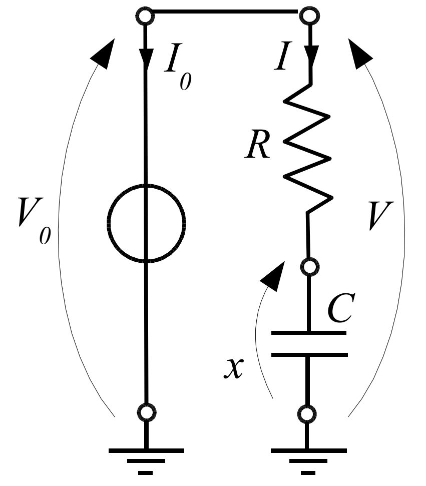
The solution work-flow is only determined at the *overall system level*

$$x + RI = V$$
$$C\dot{x} = I$$
(RC network)

$$V_0 = f(t)$$ (voltage generator)

$$V_0 = V$$ (Kirchoff's law - mesh)

$$I_0 + I = 0$$ (Kirchoff's law - node)



Principle #1: Declarative Modelling

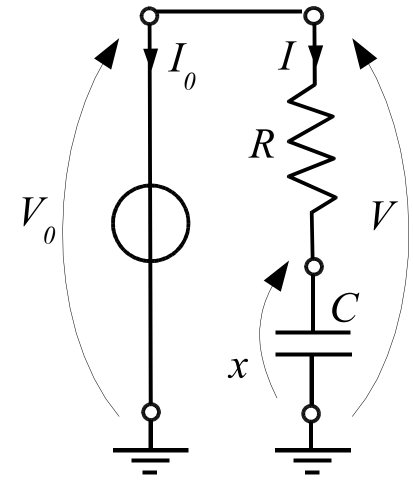
The solution work-flow is only determined at the *overall system level*

$$x + RI = V$$
$$C\dot{x} = I$$
(RC network)

$$V_0 = f(t)$$ (voltage generator)

$$V_0 = V$$ (Kirchoff's law - mesh)

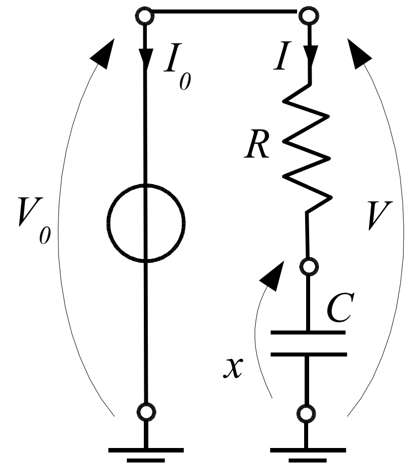
$$I_0 + I = 0$$ (Kirchoff's law - node)



$$V_0 = f(t)$$
$$V = V_0$$
$$I = \frac{V - x}{R}$$
$$I_0 = -I$$
$$\dot{x} = \frac{I}{C}$$

Principle #1: Declarative Modelling

The solution work-flow is only determined at the *overall system level*



$$x + RI = V$$
$$C\dot{x} = I$$
(RC network)

$$V_0 = f(t)$$
(voltage generator)

$$V_0 = V$$
(Kirchoff's law - mesh)

$$I_0 + I = 0$$
(Kirchoff's law - node)

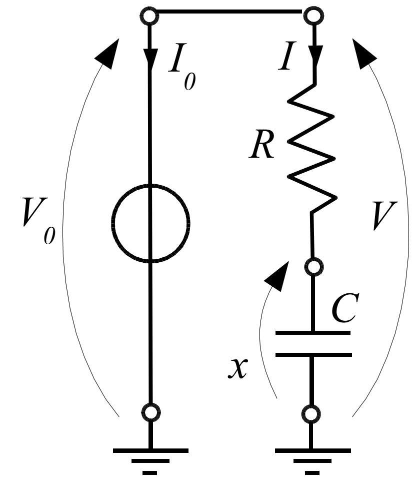
$$V_0 = f(t)$$
$$V = V_0$$
$$I = \frac{V - x}{R}$$
$$I_0 = -I$$
$$\dot{x} = \frac{I}{C}$$

```
x := x_initial
t := t_initial
loop
   V_0 = f(t)
   V := V_0
   I := (V - x)/R
   I_0 := -I
   dx_dt := I/C
   x := x + h*dx_dt
   t := t + h
end loop
```

Principle #1: Declarative Modelling

The solution work-flow is only determined at the *overall system level*

$$x + RI = V$$
$$C\dot{x} = I$$
(RC network)



$$V_0 = f(t)$$ (voltage generator)

$$V_0 = V$$ (Kirchoff's law - mesh)

$$I_0 + I = 0$$ (Kirchoff's law - node)

$$V_0 = f(t)$$
$$V = V_0$$
$$I = \frac{V - x}{R}$$
$$I_0 = -I$$
$$\dot{x} = \frac{I}{C}$$

```
x := x_initial
t := t_initial
loop
   V_0 = f(t)
   V := V_0
   I := (V - x)/R
   I_0 := -I
   dx_dt := I/C
   x := x + h*dx_dt
   t := t + h
end loop
```

performed automatically by a tool!

Principle #1: Declarative Modelling

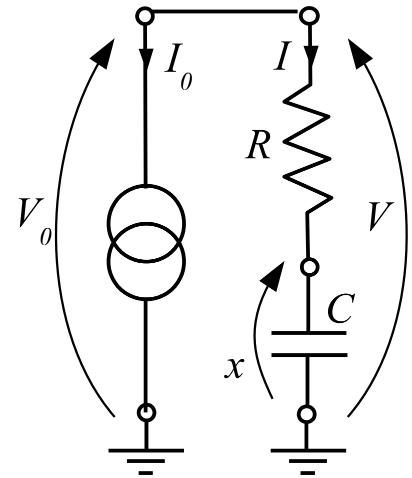
The same component can be reused in different contexts

$$x + RI = V$$
$$C\dot{x} = I$$
(RC network)

$$I_0 = f(t)$$
(**current** generator)

$$V_0 = V$$
(Kirchoff's law - mesh)

$$I_0 + I = 0$$
(Kirchoff's law - node)



$$I_0 = f(t)$$
$$I = -I_0$$
$$V = x + RI$$
$$V_0 = V$$
$$\dot{x} = \frac{I}{C}$$

```
x := x_initial
t := t_initial
loop
   I_0 := f(t)
   I  := -I_0
   V  := x + R*I
   V_0 := V
   dx_dt := I/C
   x  := x + h*dx_dt
   t  := t + h
end loop
```

```
x := x_initial
t := t_initial
loop
   V_0 = f(t)
   V  := V_0
   I  := (V - x)/R
   I_0 := -I
   dx_dt := I/C
   x  := x + h*dx_dt
   t  := t + h
end loop
```

Modularity

Models interact through physical ports

their behaviour depends explicitly on the port variables

not on the actual connected components

A model can be internally described

as the connection of other models

Principle #2: Modularity

- Physical ports: coupled effort and flow variables
  - Electrical systems: *Voltage and Current*
  - 1D Mechanical systems (Trans): *Displacement and Force*
  - 1D Mechanical systems (Rot): *Angle and Torque*
  - Hydraulic systems: *Pressure and Flow*
  - Thermal Systems: *Temperature and Thermal Power Flow*
  - …

Principle #2: Modularity

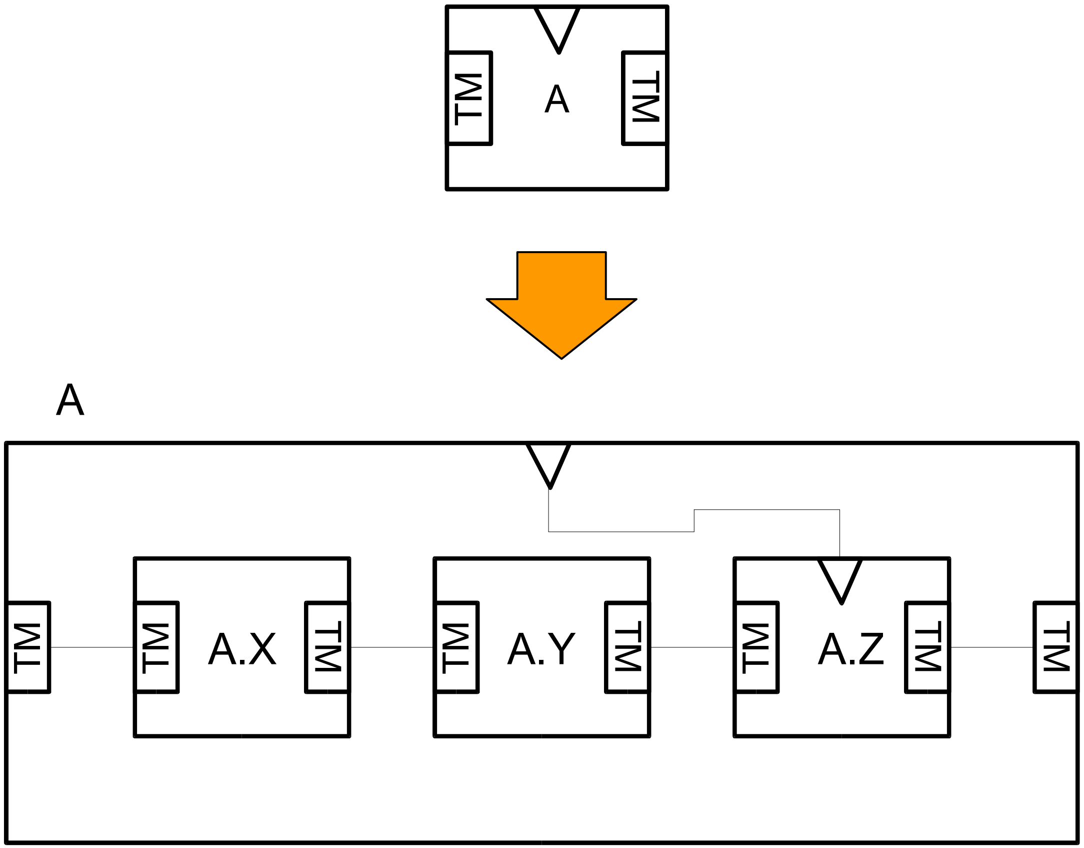
- Physical ports: coupled effort and flow variables
  - Electrical systems: *Voltage and Current*
  - 1D Mechanical systems (Trans): *Displacement and Force*
  - 1D Mechanical systems (Rot): *Angle and Torque*
  - Hydraulic systems: *Pressure and Flow*
  - Thermal Systems: *Temperature and Thermal Power Flow*
  - …

- Connection of *N* ports ↔ Connection *equations*

$e_1 = e_2 = \ldots = e_N$ (Same voltage / displacement / angle / pressure)

$\sum f_j = 0$ (Currents / Forces / Torques / Flows sum to zero)

Principle #2: Modularity



Inheritance

Parent-Child (“is-a”) relationships

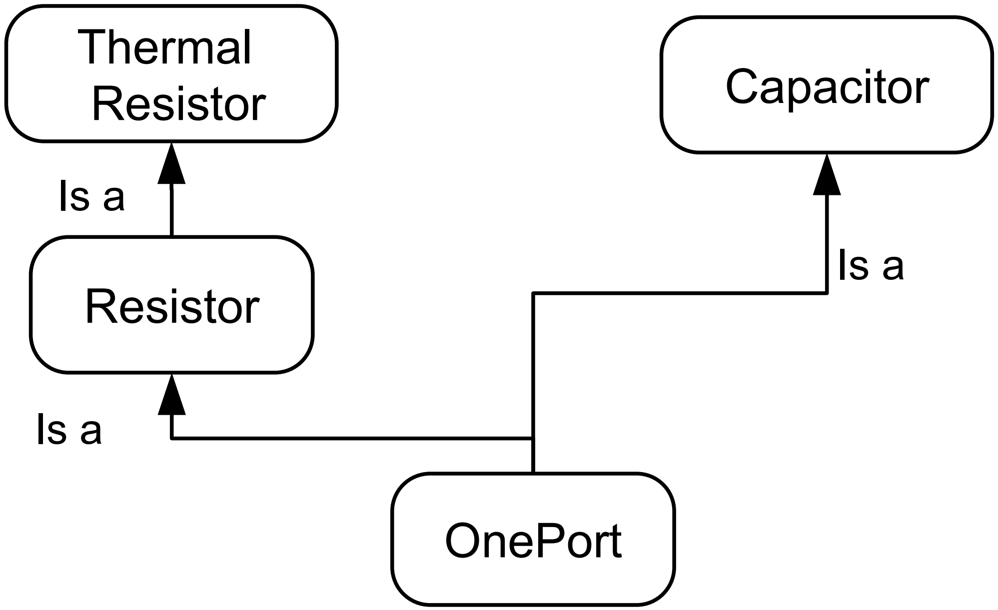
can be established among models

A child model inherits the parent features

(variables, parameters, equations, sub-models)

and adds its specific ones

Principle #3: Inheritance



Equation-Based Object-Oriented Modelling

Description of basic components by differential-algebraic equations, discrete-event equations and a-causal ports

Equation-Based Object-Oriented Modelling

Description of basic components by differential-algebraic equations, discrete-event equations and a-causal ports

+

Modular and hierarchical composition (object connection diagrams)

Equation-Based Object-Oriented Modelling

Description of basic components by differential-algebraic equations, discrete-event equations and a-causal ports

+

Modular and hierarchical composition
(object connection diagrams)

+

Object-Oriented features
(inheritance, encapsulation)

Equation-Based Object-Oriented Modelling

Description of basic components by differential-algebraic equations, discrete-event equations and a-causal ports

+

Modular and hierarchical composition
(object connection diagrams)

+

Object-Oriented features
(inheritance, encapsulation)

+

Automatic generation of simulation code

The Modelica Language

Modelica Fact Sheet

- Language for equation-based, object-oriented dynamic system modelling
- Version 1.0 introduced in 1998
- Current version: 3.4 released in 2018
- Developed and maintained by non-profit Modelica Association

Modelica Fact Sheet

- Language for equation-based, object-oriented dynamic system modelling
- Version 1.0 introduced in 1998
- Current version: 3.4 released in 2018
- Developed and maintained by non-profit Modelica Association
- Companion Modelica Standard Library of basic models
- Tool-Independent language definition

Modelica Fact Sheet

- Language for equation-based, object-oriented dynamic system modelling
- Version 1.0 introduced in 1998
- Current version: 3.4 released in 2018
- Developed and maintained by non-profit Modelica Association
- Companion Modelica Standard Library of basic models
- Tool-Independent language definition
- Supported by 9 commercial and 2 open-source simulation tools
- Extensive Open-Source & Commercial Model Libraries
- Development of new models eased by EOO approach

Example Models

```
type Voltage = Real(unit="V", nominal = 1e4);
type Current = Real(unit="A", nominal = 1e4);
type Power = Real (unit="W", nominal = 1e8);
type Resistance = Real (unit="V/A");
```

Example Models

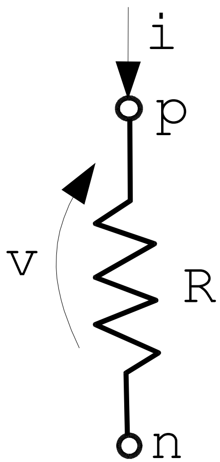
```
type Voltage = Real(unit="V", nominal = 1e4);
type Current = Real(unit="A", nominal = 1e4);
type Power = Real (unit="W", nominal = 1e8);
type Resistance = Real (unit="V/A");
```

```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```

Example Models

```
type Voltage = Real(unit="V", nominal = 1e4);
type Current = Real(unit="A", nominal = 1e4);
type Power = Real (unit="W", nominal = 1e8);
type Resistance = Real (unit="V/A");
```

```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```

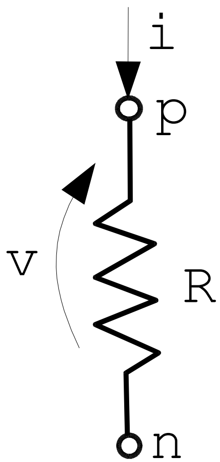



```
model Resistor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Resistance R;
equation
  v = p.v - n.v;
  i = p.i;
  0 = p.i + n.i;
  v = R*i;
end Resistor;
```

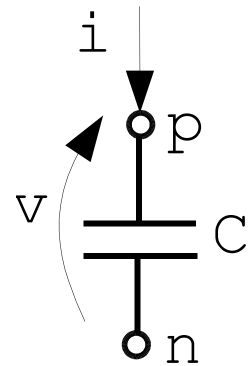
Example Models

```
type Voltage = Real(unit="V", nominal = 1e4);
type Current = Real(unit="A", nominal = 1e4);
type Power = Real (unit="W", nominal = 1e8);
type Resistance = Real (unit="V/A");
```

```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```




```
model Resistor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Resistance R;
equation
  v = p.v - n.v;
  i = p.i;
  0 = p.i + n.i;
  v = R*i;
end Resistor;
```

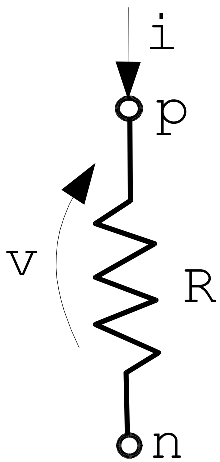



```
model Capacitor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Capacitance C;
equation
  v = p.v - n.v;
  i = p.i;
  0 = p.i + n.i;
  i = C*der(v);
end Capacitor;
```

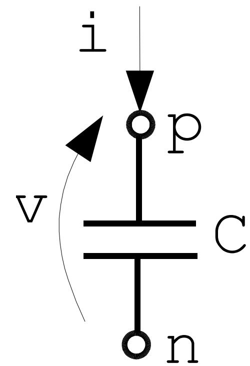
Example Models

```
type Voltage = Real(unit=”V”, nominal = 1e4);
type Current = Real(unit=”A”, nominal = 1e4);
type Power = Real (unit=”W”, nominal = 1e8);
type Resistance = Real (unit=”V/A”);
```

```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```



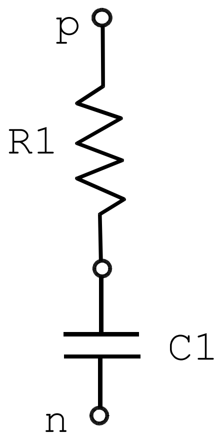

```
model Resistor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Resistance R;
equation
  v = p.v-n.v;
  i = p.i;
  0 = p.i + n.i;
  v = R*i;
end Resistor;
```




```
model Capacitor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Capacitance C;
equation
  v = p.v-n.v;
  i = p.i;
  0 = p.i + n.i;
  i = C*der(v);
end Capacitor;
```

Models in DECLARATIVE form!

Modular & Hierarchical Composition

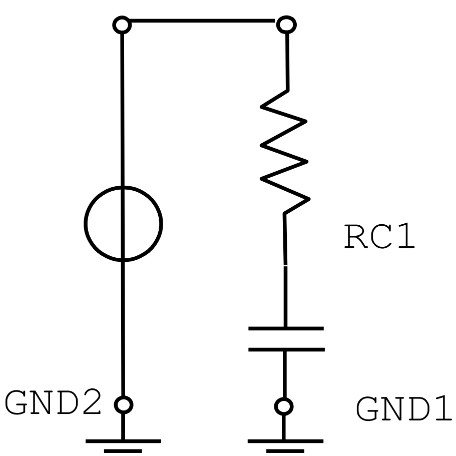



```
model RCNet
  parameter Resistance Rnet;
  parameter Capacitance Cnet;
  Resistor R1(R=Rnet);
  Capacitor C1(C=Cnet);
  Pin p,n;
equation
  connect(R1.n, C1.p);
  connect(R1.p, p);
  connect(C1.n, n);
end RCNet;
```

Modifiers
(parameter propagation)

Equivalent to:

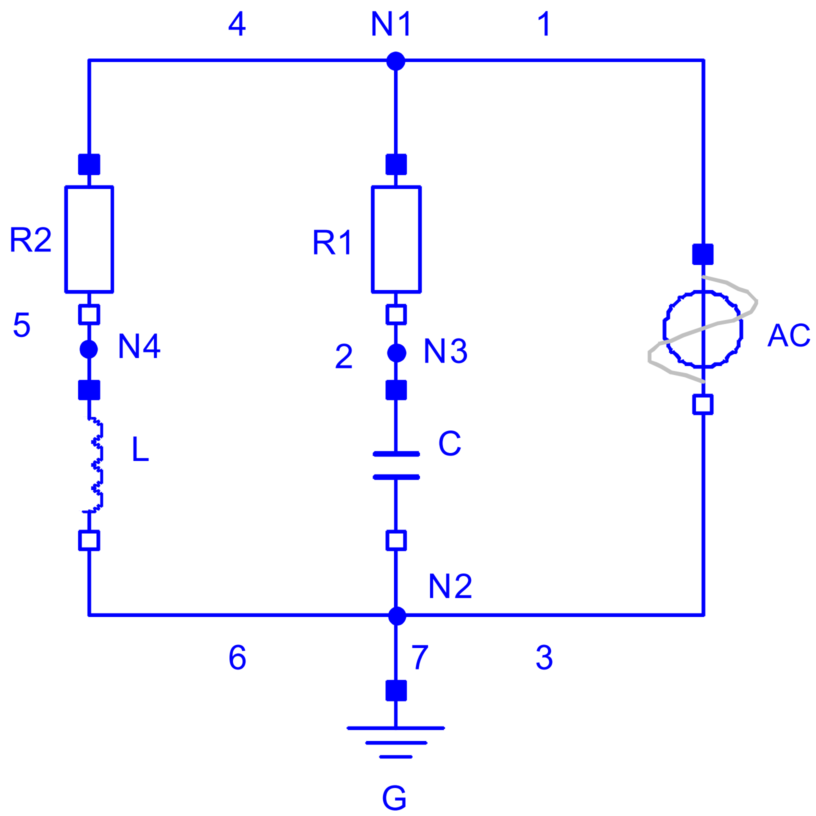
```
R1.n.v = C1.p.v;
R1.n.i + C1.p.i = 0;
```




```
model SimpleCircuit
  RCnet RC1(Rnet=100, Cnet=1e-6);
  Vsource V0;
  Ground GND1, GND2;
equation
  connect(RC1.n, GND1.p);
  connect(RC1.p, V0.p);
  connect(V0.n, GND2.p);
end SimpleCircuit;
```

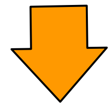
Graphical Annotations and Object Diagrams

- Graphical annotations allow to build and visualize composite models graphically
- The underlying model description is textual



Inheritance: Factoring Out Common Features

Resistor and Capacitor have common features



Factor them out in a base class OnePort

```
partial model OnePort
  Pin p,n;
  Voltage v;
  Current i;
equation
    v = p.v - n.v;
    i = p.i;
    0 = p.i + -n.i;
end OnePort;
```

Inheritance: Factoring Out Common Features

Resistor and Capacitor have common features

Factor them out in a base class OnePort

```
partial model OnePort
  Pin p,n;
  Voltage v;
  Current i;
equation
    v = p.v - n.v;
    i = p.i;
    0 = p.i + -n.i;
end OnePort;
```

```
model Resistor
  extends OnePort;
  parameter Resistance R;
equation
    v = R*i;
end Resistor;
```

```
model Capacitor
  extends OnePort;
  parameter Capacitance C;
equation
    C*der(v) = i;
end Capacitor;
```

Computational Model for Continuous-Time Systems

Model (DAEs) $F(x, \dot{x}, v, p, u, t) = 0$

Causalization

(solving for $\dot{x}, v$)

State-Space representation. (ODEs)

$$\dot{x} = f(x, p, u, t)$$
$$v = g(x, p, u, t)$$

Computational Model for Continuous-Time Systems

Model (DAEs) $F(x, \dot{x}, v, p, u, t) = 0$

Causalization

(solving for $\dot{x}, v$)

State-Space representation. (ODEs)

$$\dot{x} = f(x, p, u, t)$$
$$v = g(x, p, u, t)$$

ODE Time integration

Computational Model for Continuous-Time Systems

Model (DAEs) $F(x, \dot{x}, v, p, u, t) = 0$

Causalization

(solving for $\dot{x}, v$)

State-Space representation.
(ODEs)

$$\dot{x} = f(x, p, u, t)$$
$$v = g(x, p, u, t)$$

ODE Time integration

Export as state-space block
for other simulation environmens
or co-simulation

Related Standards: FMI

Functional Mock-Up Interface

- Open Standard for *causal* dynamic model exchange
- Internal representation of an FMU: state-space system with inputs and outputs
  - DLL or C-code for computation
  - XML description of variables and parameters
- Automatically generated from OO models with only input and output connectors at the top level
  - Actuator inputs
  - Sensor outputs

Related Standards: FMI

Functional Mock-Up Interface

- Open Standard for *causal* dynamic model exchange
- Internal representation of an FMU: state-space system with inputs and outputs
  - DLL or C-code for computation
  - XML description of variables and parameters
- Automatically generated from OO models with only input and output connectors at the top level
  - Actuator inputs
  - Sensor outputs
- Supported by all Modelica tools and by over 130 simulation tools *https://fmi-standard.org/tools/*
- FMI-ME (model exchange): $$\dot{x} = f(x, p, u, t)$$ $$v = g(x, p, u, t)$$
- FMI-CS (co-simulation) $$x_{k+1} = f_h(x_k, p, u_k, t_k)$$ $$v_k = g(x_k, p, u_k, t_k)$$

Related Standards: SSP & DCP

- Open Standard for *parametrization* and *connection* of FMUs to form complete system models
- Allows to describe system models built by assembling FMUs
- System integration level

Related Standards: SSP & DCP

- Open Standard for *parametrization* and *connection* of FMUs to form complete system models

ssp System Structure & Parameterization

- Allows to describe system models built by assembling FMUs
- System integration level

dcp Distributed Co-Simulation Protocol

- Open Standard Protocol for co-simulation of FMI-CS blocks
- Supports multiple distributed simulation architectures
  - Over TCP/IP
  - Over Bluetooth
  - Over USB
  - Over CAN Bus

Case Study

Satellite Attitude Modelling and Control

Joint work with
prof. Marco Lovera

Dept. Aerospace Engineering
Politecnico di Milano

Requirements

- Support the design of satellite attitude control systems
  - Actuator sizing
  - Feasibility study with idealized actuators, sensors, and control laws
  - Detailed engineering design of equipment and control laws

Requirements

- Support the design of satellite attitude control systems
  - Actuator sizing
  - Feasibility study with idealized actuators, sensors, and control laws
  - Detailed engineering design of equipment and control laws
- Unified modelling framework,
  reuse models as much as possible

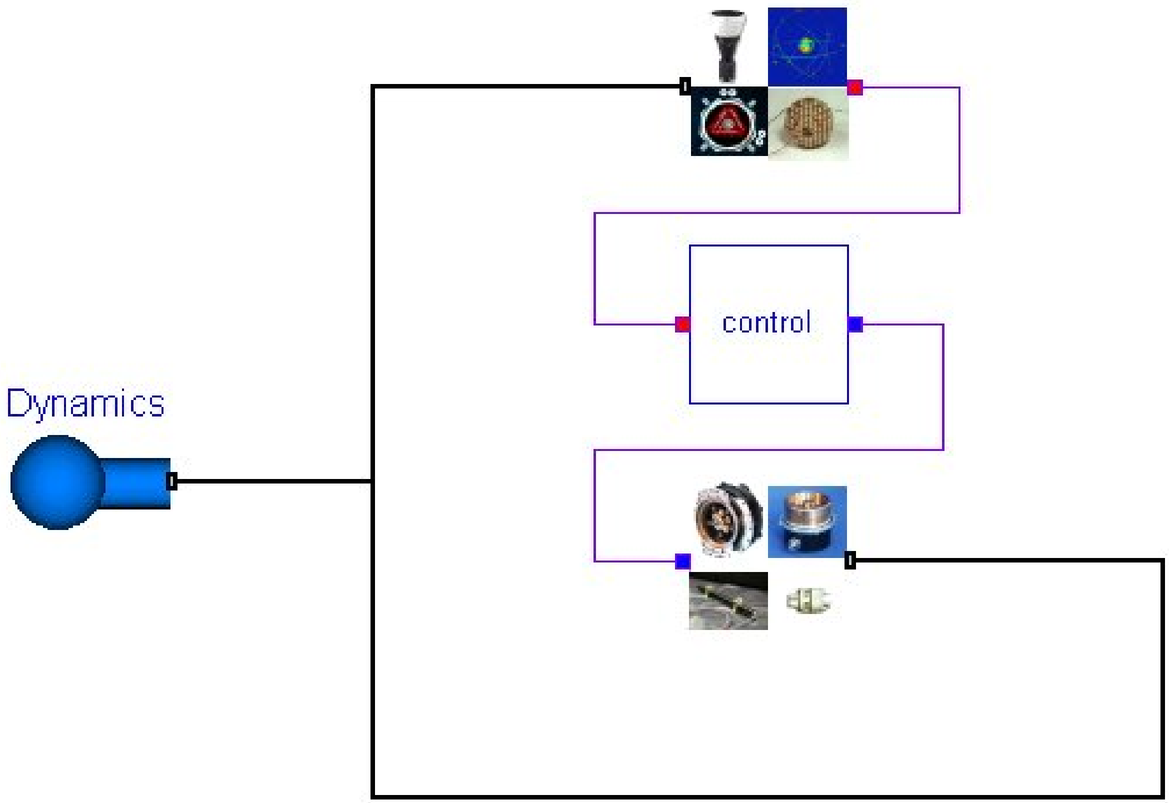
Requirements

- Support the design of satellite attitude control systems
  - Actuator sizing
  - Feasibility study with idealized actuators, sensors, and control laws
  - Detailed engineering design of equipment and control laws
- Unified modelling framework,
  reuse models as much as possible
- Environment model
  - Accurate gravity field model based on harmonic expansions
  - Accurate magnetic field model based on harmonic expansions
  - Atmospheric model

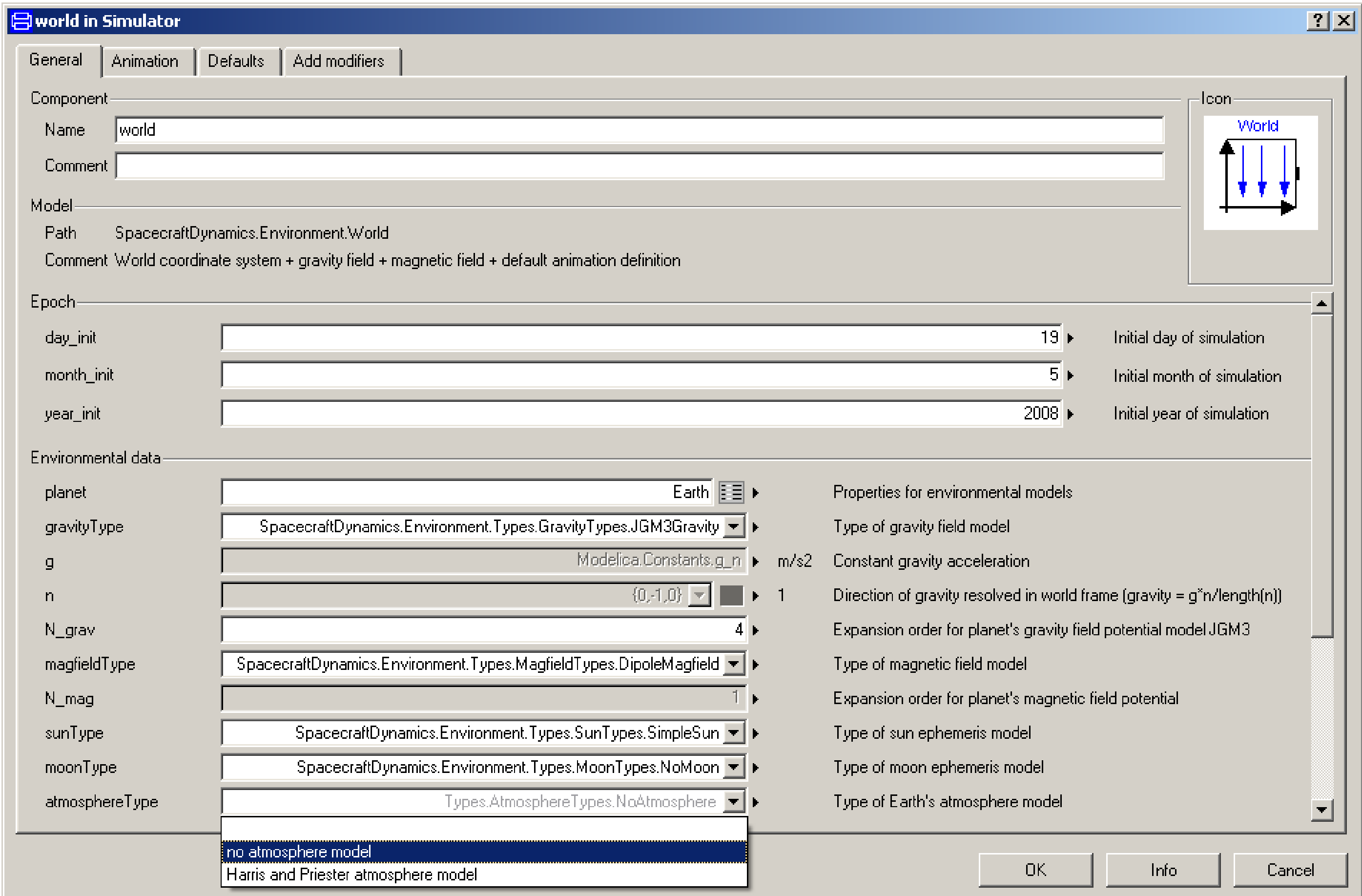
Requirements

- Support the design of satellite attitude control systems
  - Actuator sizing
  - Feasibility study with idealized actuators, sensors, and control laws
  - Detailed engineering design of equipment and control laws
- Unified modelling framework,
  reuse models as much as possible
- Environment model
  - Accurate gravity field model based on harmonic expansions
  - Accurate magnetic field model based on harmonic expansions
  - Atmospheric mode
- Systematic use of inheritance and replaceable classes to achieve maximum flexibility of detail

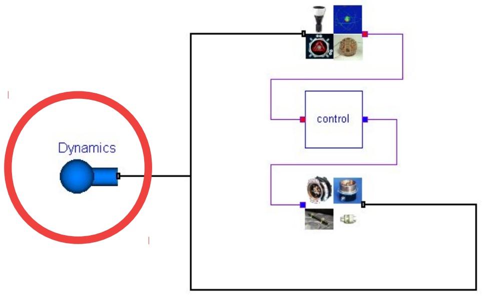
Spacecraft Model Architecture (Top View)



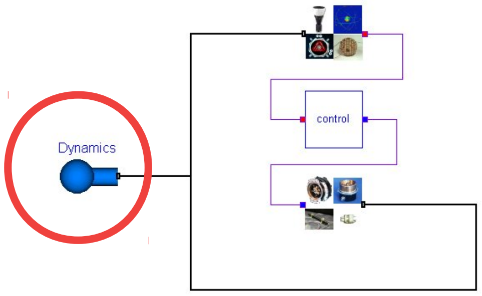
Choice of Environmental Model Parameters

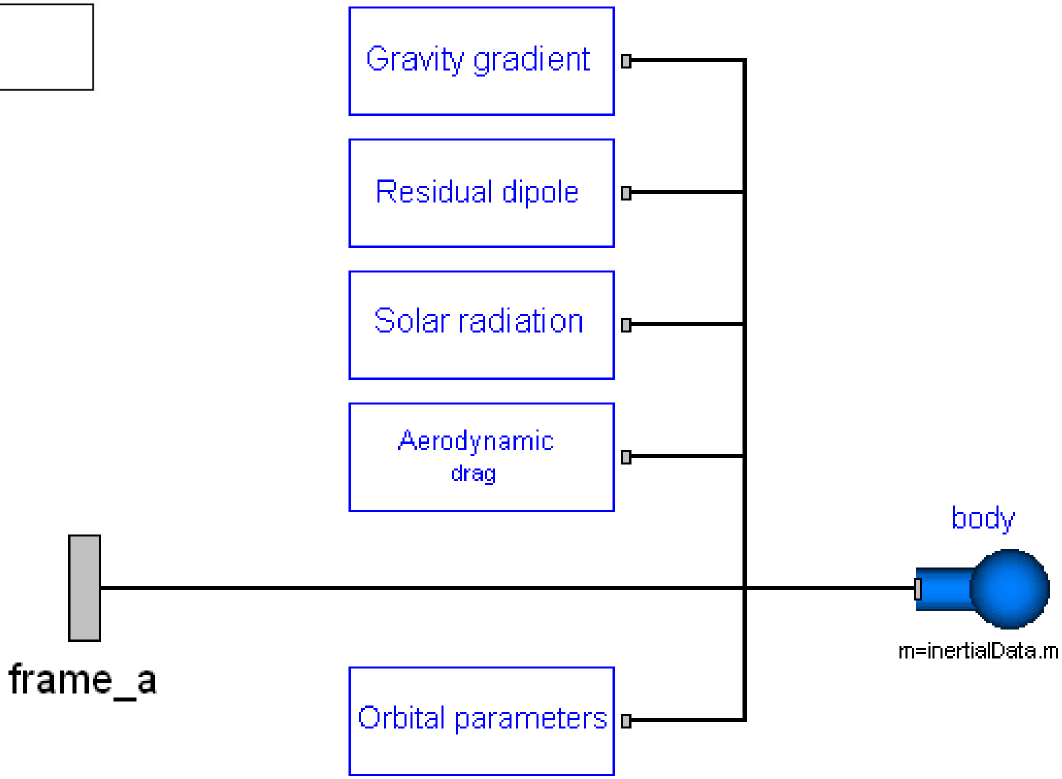


Spacecraft Dynamics

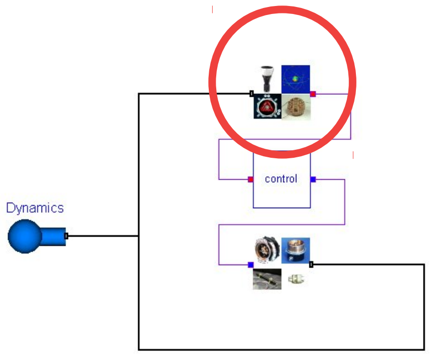


Spacecraft Dynamics

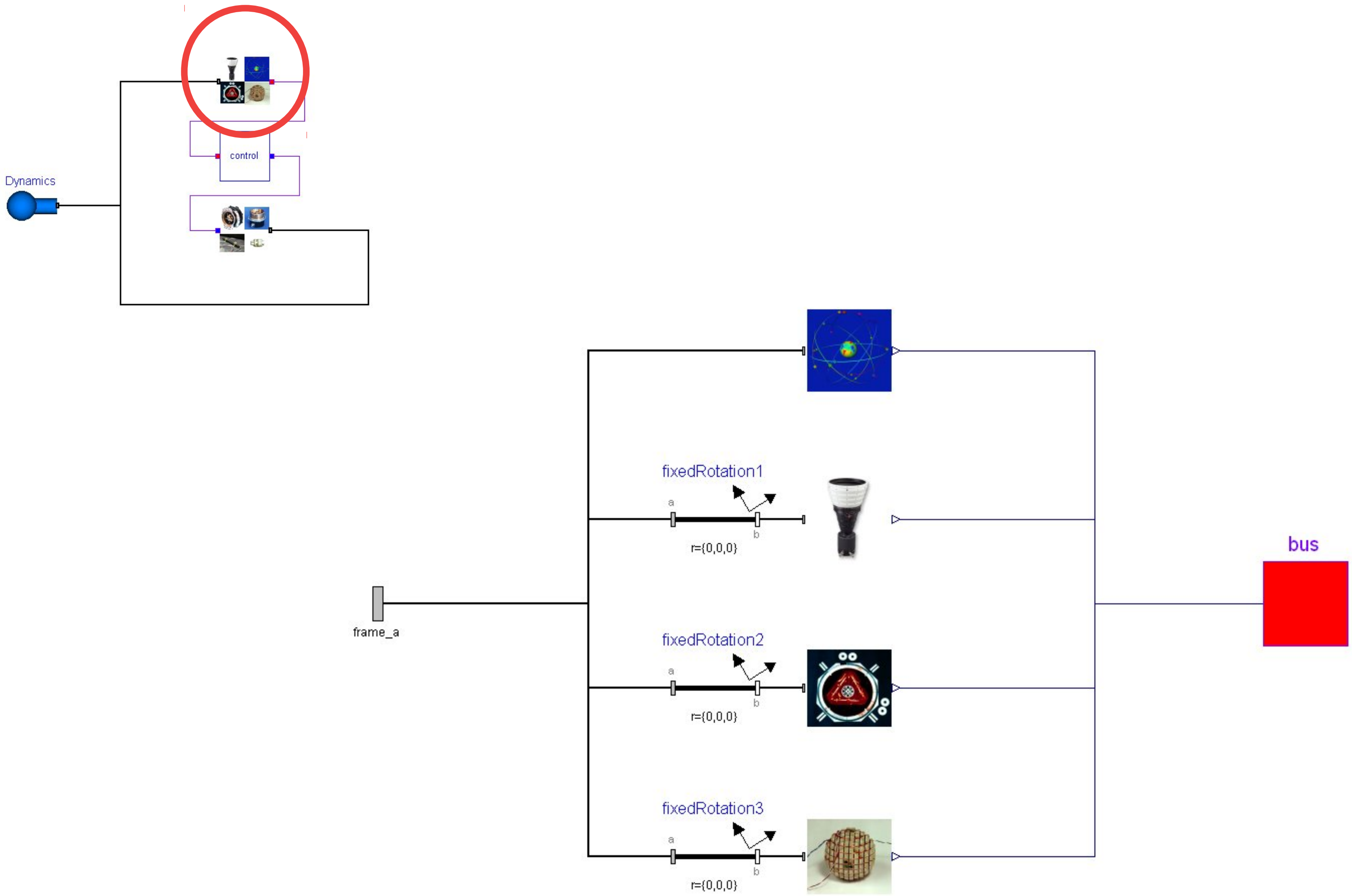




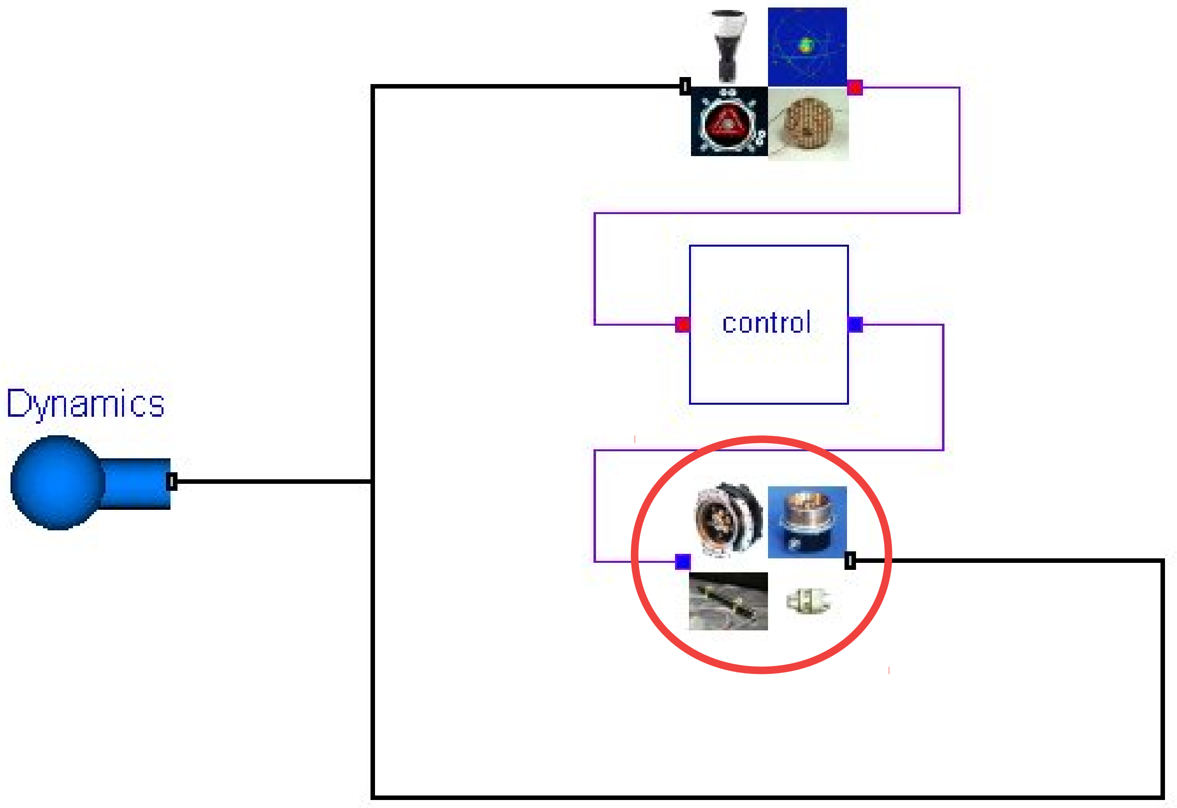
Sensor Block



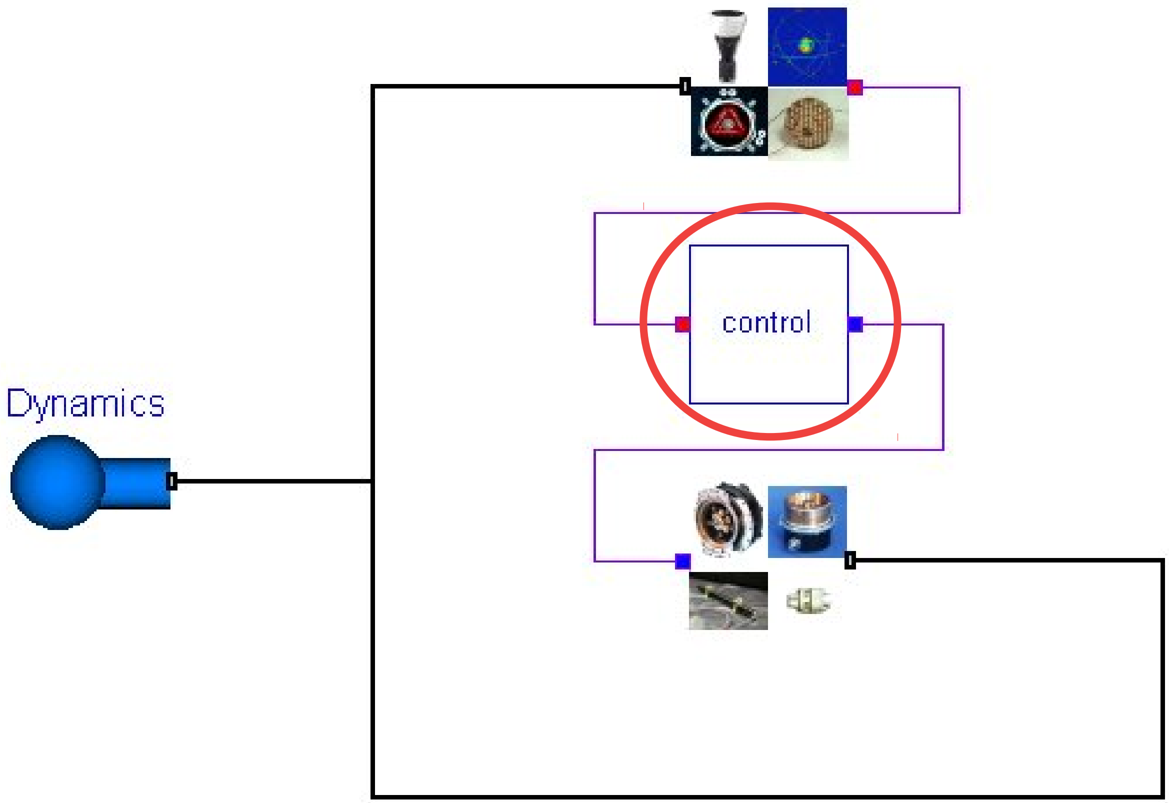
Sensor Block



Actuator and Control Blocks



Actuator and Control Blocks



Use of Replaceable Objects

- All the components of the SensorBlock, ActuatorBlock, and ControlBlock models are *replaceable*
  - The base model defines empty interfaces with connectors
  - When instantiating a specific spacecraft model, the empty interfaces can be replaced with any of their child models
    - Idealized models (e.g. ideal torque or force generation)
    - More realistic models (with some non-ideal effects)
    - Actual engineering models of commercial units
  - The replaceable structure is recursive down to the level of the invdividual sub-assembly model
  - All the concrete replacements are stored in a reusable library

Use of Replaceable Objects

- All the components of the SensorBlock, ActuatorBlock, and ControlBlock models are *replaceable*
  - The base model defines empty interfaces with connectors
  - When instantiating a specific spacecraft model, the empty interfaces can be replaced with any of their child models
    - Idealized models (e.g. ideal torque or force generation)
    - More realistic models (with some non-ideal effects)
    - Actual engineering models of commercial units
  - The replaceable structure is recursive down to the level of the invdividual sub-assembly model
  - All the concrete replacements are stored in a reusable library
- Á la carte customization of the level of detail of each block
- Extensive families of models can be developed
  - Throughout the project lifetime
  - With guaranteed consistency of data and models

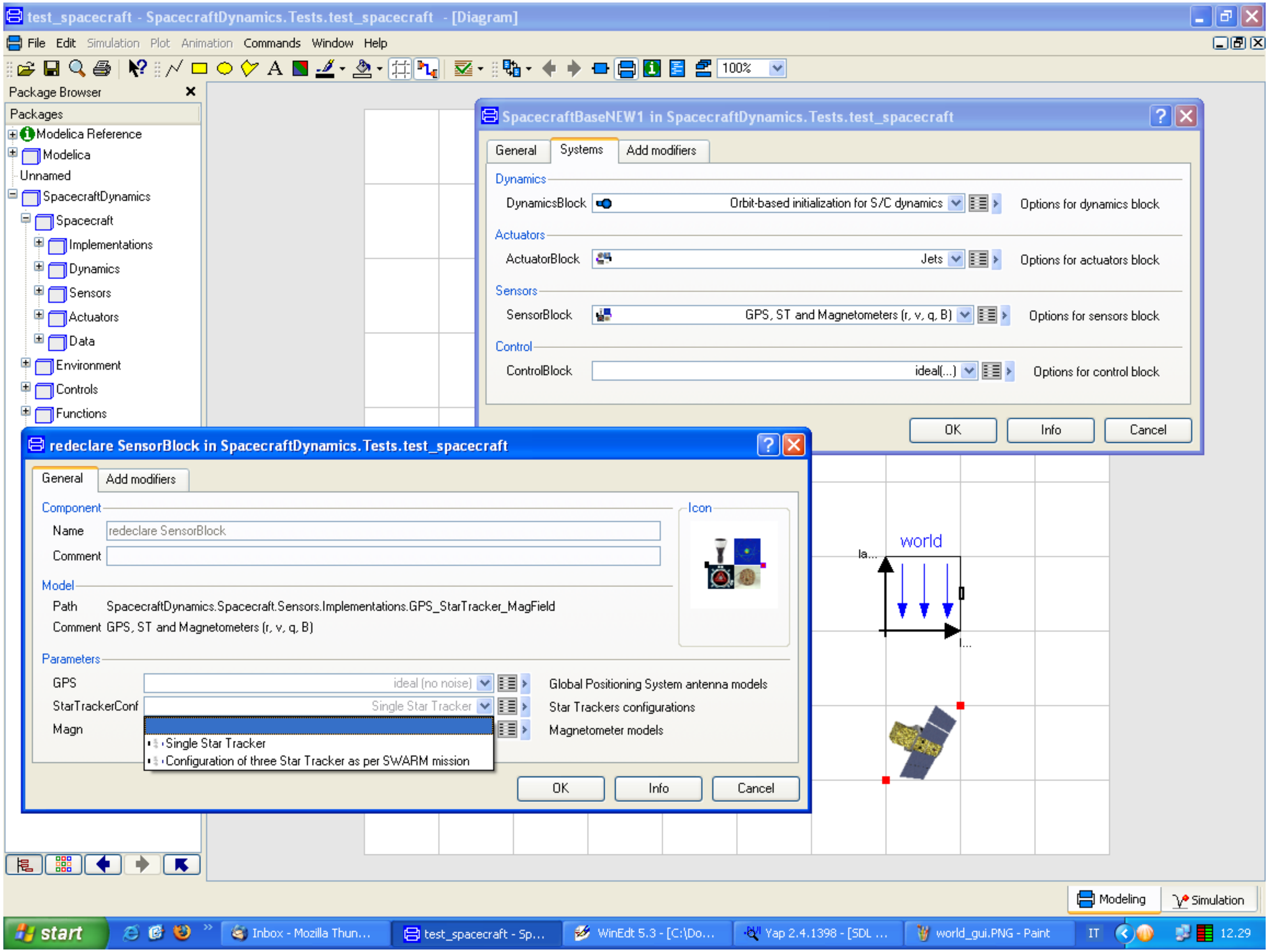
Use of Replaceable Objects

- All the components of the SensorBlock, ActuatorBlock, and ControlBlock models are *replaceable*
  - The base model defines empty interfaces with connectors
  - When instantiating a specific spacecraft model, the empty interfaces can be replaced with any of their child models
    - Idealized models (e.g. ideal torque or force generation)
    - More realistic models (with some non-ideal effects)
    - Actual engineering models of commercial units
  - The replaceable structure is recursive down to the level of the invdividual sub-assembly model
  - All the concrete replacements are stored in a reusable library
- Á la carte customization of the level of detail of each block
- Extensive families of models can be developed
  - Throughout the project lifetime
  - With guaranteed consistency of data and models
- Compare with classical copy-paste-modify approach...

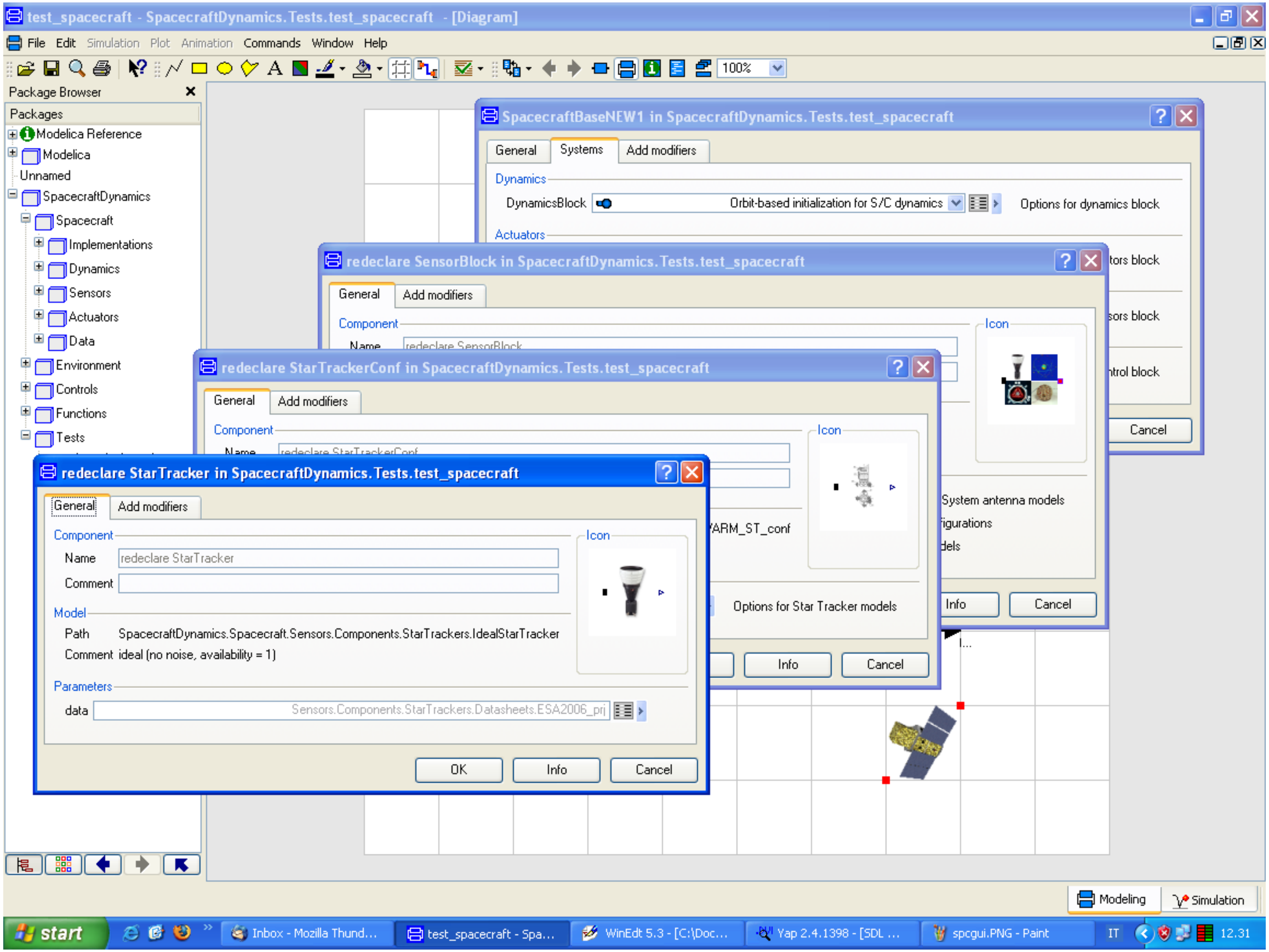
Example: Textual Code View

```
model Example
  import SpacecraftDynamics.Spacecraft.* ;
  inner Environment.World world;
  Implementations.SpacecraftBase spacecraft(
    redeclare model SensorBlock =
      Sensors.Implementations.GPS_StarTracker_MagField(
        redeclare model StarTrackerConf =
          Sensors.Components.StarTrackers.Assemblies.SingleST_conf(
            redeclare model StarTracker =
              Sensors.Components.StarTrackers.StarTrackerBase(
                data=Sensors.Components.StarTrackers.Datasheets.ESA2006_prj)))
    redeclare model ActuatorBlock =
    ...
    redeclare model ControlBlock =
    ...
  );
end Example;
```

Example: Setup via GUI



Example: Setup via GUI



Use for Actuator Sizing

- Ideal SensorBlock
- ActuatorBlock with unknown input torques and forces
- Equations in ControlBlock
  prescribe the exact pointing of the target
- Exactly the same physical model of spacecraft and environment as for the forward closed-loop simulation

Use for Actuator Sizing

- Ideal SensorBlock
- ActuatorBlock with unknown input torques and forces
- Equations in ControlBlock
  prescribe the exact pointing of the target
- Exactly the same physical model of spacecraft and environment as for the forward closed-loop simulation

⇒ System equations automatically solved backwards to compute the required forces and torques

Use for Actuator Sizing

- Ideal SensorBlock
- ActuatorBlock with unknown input torques and forces
- Equations in ControlBlock
  prescribe the exact pointing of the target
- Exactly the same physical model of spacecraft and environment as for the forward closed-loop simulation

⇨ System equations automatically solved backwards to compute the required forces and torques

⇨ Much better than rule-of-thumb sizing!

Use for Actuator Sizing

- Ideal SensorBlock
- ActuatorBlock with unknown input torques and forces
- Equations in ControlBlock
  prescribe the exact pointing of the target
- Exactly the same physical model of spacecraft and environment as for the forward closed-loop simulation

➡ System equations automatically solved backwards to compute the required forces and torques

➡ Much better than rule-of-thumb sizing!

➡ Unified & Consistent modelling framework

Use for Idealized Control Law Validation

- Ideal SensorBlock and ActuatorBlock
- Equations in ControlBlock implement textbook-version of control law

Use for Idealized Control Law Validation

- Ideal SensorBlock and ActuatorBlock
- Equations in ControlBlock implement textbook-version of control law

➡ First assessment of control performance

➡ Very fast simulation (low detail)

Use for Accurate Control Performance Validation

- Detailed SensorBlock and ActuatorBlock
- Equations in ControlBlock implement detailed version of control law, possibly in discrete time

Use for Accurate Control Performance Validation

- Detailed SensorBlock and ActuatorBlock
- Equations in ControlBlock implement detailed version of control law, possibly in discrete time

⇨ Assessment of control performance in realistic conditions

Use for SW/HW-in-the-Loop Controller Validation

- Empty ControlBlock with top level inputs/outputs for sensor/actuator signals
- ControlBlock inputs/outputs propagated to top model inputs/outputs
- Overal model exported as FMU

Use for SW/HW-in-the-Loop Controller Validation

- Empty ControlBlock with top level inputs/outputs for sensor/actuator signals
- ControlBlock inputs/outputs propagated to top model inputs/outputs
- Overal model exported as FMU

➡ Closed-loop simulation on actual control sw

➡ Closed-loop simulation on real-time control hw

The Communities

The Modelica Association

- Non-profit association developing
  Open Standards for Dynamic System Simulation
- Established in 1997
- Brings together
  - Modelica/FMI tool developers
  - Scientists and researchers from academia
  - R&D engineers from companies
  - Multidisciplinary interest in Tools, Methods, and Applications

The Modelica Association

- Non-profit association developing
  Open Standards for Dynamic System Simulation
- Established in 1997
- Brings together
  - Modelica/FMI tool developers
  - Scientists and researchers from academia
  - R&D engineers from companies
  - Multidisciplinary interest in Tools, Methods, and Applications
- Organizes the International and Regional (American & Asian) Modelica Conferences
- 100% Open access products
  - Open access standards
  - Free model libraries
  - Open access Modelica Conference proceedings

The Modelica Association

- Non-profit association developing Open Standards for Dynamic System Simulation
- Established in 1997
- Brings together
  - Modelica/FMI tool developers
  - Scientists and researchers from academia
  - R&D engineers from companies
  - Multidisciplinary interest in Tools, Methods, and Applications
- Organizes the International and Regional (American & Asian) Modelica Conferences
- 100% Open access products
  - Open access standards
  - Free model libraries
  - Open access Modelica Conference proceedings
- Activity organized in coordinated Modelica Association Projects (MAPs)

The Modelica Association Projects

- MAP-LANG:
Develops the Modelica Language Specification
- MAP-LIB:
Develops the Modelica Standard Library
- MAP-FMI:
Develops the FMI standard
- MAP-SSP
Develops the SSP standard
- MAP-DCP:
Develops the DCP standard

- Other projects can be added if they are consistent with the overall goals of the Modelica Association
(Open Standards for System Simulation)

The Open Source Modelica Consortium (OSMC)

- Non-profit organization developing
  the Open Modelica Compiler (OMC) suite of tools
- Established in 2007
- 22 Companies and Institutes
  - ABB
  - Bosch-Rexroth
  - Siemens Turbo Machinery
  - Saab
  - EDF
  - RTE
  - ...
- 28 Universities

The OpenModelica Toolkit

- OpenModelica Compiler (OMC)
  - Full-fledged Modelica 3.4 compiler
  - Dual licensing: GPL v3 and OSMC-PL
  - Current release: 1.14.0
  - Coverage: 95% MSL, 80% known open-source Modelica Libraries
  - Target for 2.0.0 release in 2020: 100% MSL, >95% other libraries
  - Python, Matlab, Julia interactive interfaces with API

The OpenModelica Toolkit

- OpenModelica Compiler (OMC)
  - Full-fledged Modelica 3.4 compiler
  - Dual licensing: GPL v3 and OSMC-PL
  - Current release: 1.14.0
  - Coverage: 95% MSL, 80% known open-source Modelica Libraries
  - Target for 2.0.0 release in 2020: 100% MSL, >95% other libraries
  - Python, Matlab, Julia interactive interfaces with API
- OMEdit
  - IDE with GUI, similar to Dymola
  - Debugging of equation-based models
  - Uses OMC as Modelica engine

The OpenModelica Toolkit

- OpenModelica Compiler (OMC)
  - Full-fledged Modelica 3.4 compiler
  - Dual licensing: GPL v3 and OSMC-PL
  - Current release: 1.14.0
  - Coverage: 95% MSL, 80% known open-source Modelica Libraries
  - Target for 2.0.0 release in 2020: 100% MSL, >95% other libraries
  - Python, Matlab, Julia interactive interfaces with API
- OMEdit
  - IDE with GUI, similar to Dymola
  - Debugging of equation-based models
  - Uses OMC as Modelica engine
- OMSimulator
  - Simulation environment for FMI/SSP system models
  - Integrated with OMEdit
- And many others

Development of the OpenModelica tools

- Core functionality
  - OSMC developers paid by the Consortium
- New features (also experimental):
  - Supported by research funding bodies
  - ITEA2 and ITEA3 projects
  - National research projects (mostly Sweden and Germany)
- Independent contributors

Development of the OpenModelica tools

- Core functionality
  - OSMC developers paid by the Consortium
- New features (also experimental):
  - Supported by research funding bodies
  - ITEA2 and ITEA3 projects
  - National research projects (mostly Sweden and Germany)
- Independent contributors

- About 60 contributors so far
- Development hosted on GitHub https://github.com/OpenModelica
- Continuous Integration testing on each commit, online coverage reports available
- Nightly builds available
- Bug tracking infrastructure, about 250 tickets fixed per year

OpenModelica & Open Source Model Development

- OpenModelica enables building 100% open-source modelling&simulation toolchains
- Example: RTE's Dynaωo tool for power system simulation

OpenModelica & Open Source Model Development

- OpenModelica enables building 100% open-source modelling&simulation toolchains
- Example: RTE's Dynaωo tool for power system simulation
- Joining the consortium enables to
  - Support the long-term development of the OMC tool suite
  - Have a say on the strategic development decisions
  - Get priority in the fixing of bugs and implementation of new features
  - Contribute features directly (e.g. Bosch-Rexroth's C++ runtime)

OpenModelica & Open Source Model Development

- OpenModelica enables building 100% open-source modelling&simulation toolchains
- Example: RTE’s Dynaωo tool for power system simulation
- Joining the consortium enables to
    - Support the long-term development of the OMC tool suite
    - Have a say on the strategic development decisions
    - Get priority in the fixing of bugs and implementation of new features
    - Contribute features directly (e.g. Bosch-Rexroth’s C++ runtime)
- The investment in Modelica model development is guaranteed by the whole tool-vendor ecosystem, including at least another open-source tool (JModelica)

⇨ Low-risk + no vendor lock-in

Thank you
for you kind attention!

Any questions?