Object-Oriented Modelling and Simulation: State of the Art and Future Perspectives

Francesco Casella

francesco.casella@polimi.it

Dipartimento di Elettronica, Informazione e Bioingegneria

Politecnico di Milano - Italy

POLITECNICO

MILANO 1863

Outline

- Principles of Equation-Based Object-Oriented Modelling (EOOM)
- Introduction to the Modelica OO Modelling Language
  - A bit of history
  - The language
  - The computational model
- Paradigmatic use cases
- The future (according to myself):

  OO modelling of very large distributed cyber-physical systems

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

(Yaman Barlas, Simultech 2016 keynote)

The best formalization of a simulation model

is more easily understood by a human

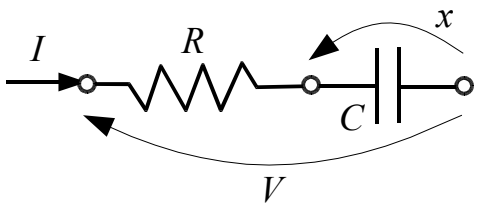
not by a computer

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*

Example: RC component



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

(DAE – declarative model)

Principle #1: Declarative Modelling

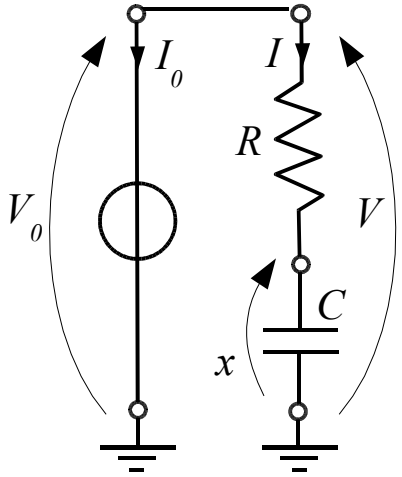
The solution work-flow is only determined at the *overall system level*

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

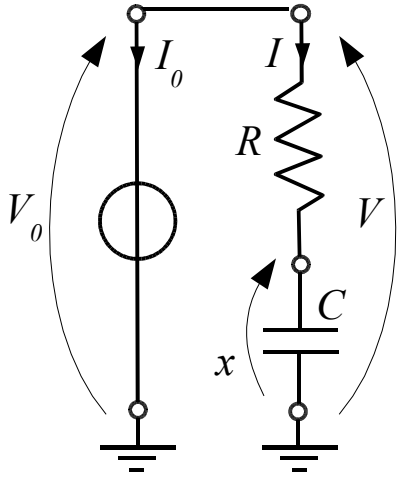
The solution work-flow is only determined at the *overall system level*

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

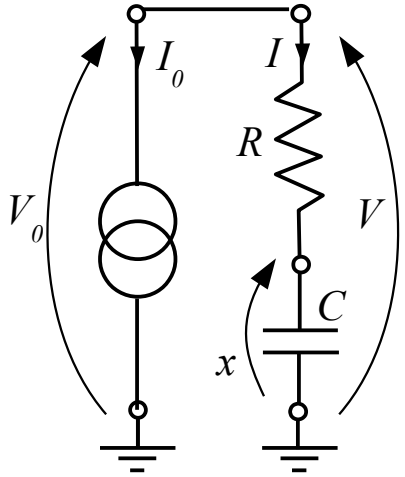
The same component can be reused in different contexts

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

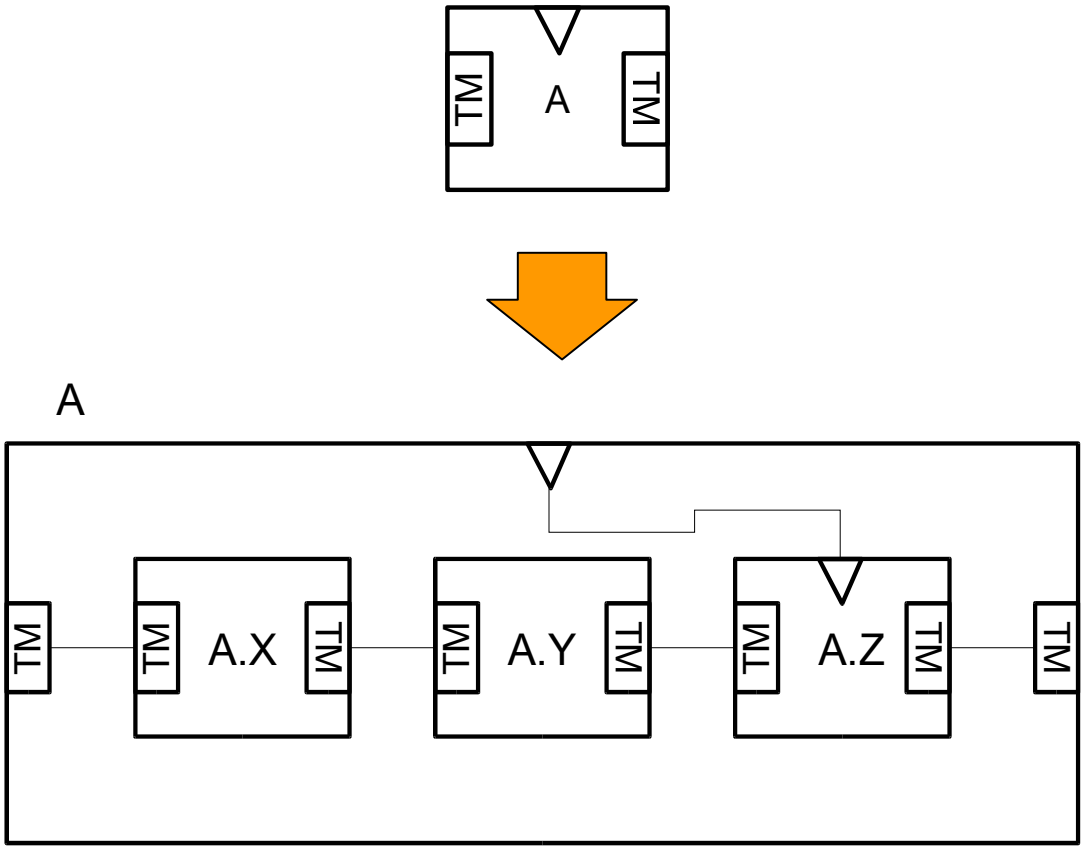
- Physical ports: coupled effort and flow variables
  - Electrical systems: Voltage and Current
  - 1D Mechanical systems (Tr): Displacement and Force
  - 1D Mechanical systems (Rot): Angle and Torque
  - Hydraulic systems: Pressure and Flow
  - Thermal Systems: Temperature and Thermal Power Flow
  -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

- 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

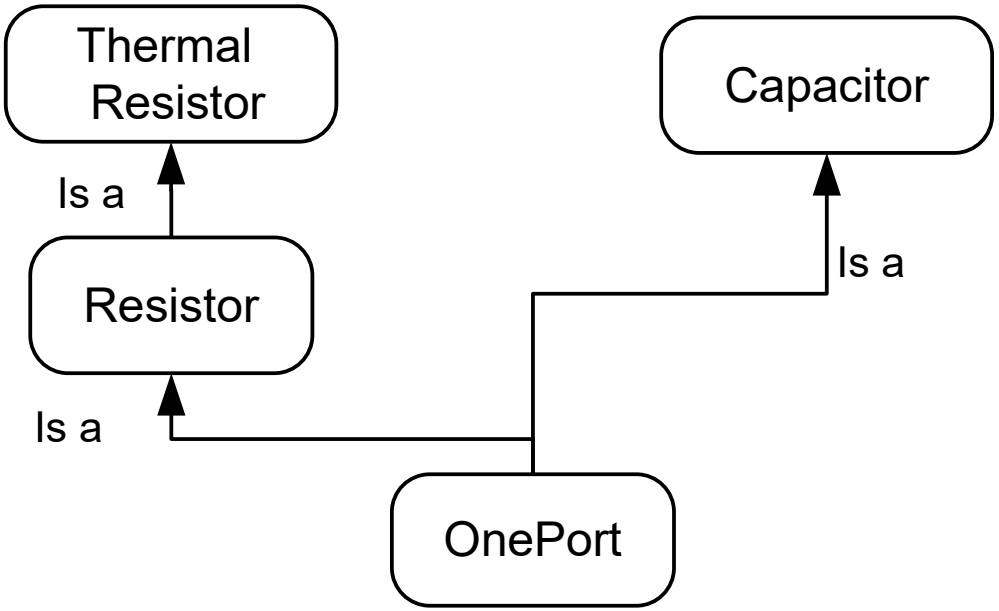
can be established among models

A child model inherits the parent features

(variables, parameters, equations, sub-models)

and adds its specific ones

Principle #3: Inheritance



EOOLTs

Several EOO modelling languages and tools follow this paradigm

- gPROMS
- Modelica
- EcoSimPro
- SimScape

In this talk I will mainly focus on Modelica, which I know best

The Modelica Language

Facts & Figures - I

- Equation-Based, Object-Oriented Modelling Language
- Tool-independent, defined by non-profit Modelica Association
- Version 1.0 rolled out in 1997, heir of earlier OOM languages
  Dymola, Omola, Ascend, NMF, IDA
- Current version 3.3 rev1, mostly backwards-compatible additions
- Companion Modelica Standard Library
  - Basic Component Models in different domains

Facts & Figures - II

Tools supporting Modelica

Tool name	***Vendor***	***License***
Dymola	Dassault Systèmes	Commercial
SimulationX	ITI (ESI Group)	Commercial
MapleSim	MapleSoft	Commercial
Wolfram System Modeller	Wolfram Research	Commercial
Amesim	Siemens PLM Sw.	Commercial
OpenModelica	OSMC	Open Source
JModelica	Modelon	Open Source

Facts & Figures - III

Modelica-related EU ITEA2 Projects 2006-2016

Combined funding

75 Million €

Brief Introduction to the language

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

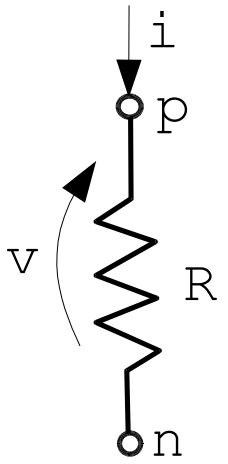
```
type Voltage = Real(unit="V", nominal = 1e4);
type Current = Real(unit="A", nominal = 1e4);
type Power = Real (unit="W", nominal = 1e8);
type Resistance = Real (unit="V/A");
```

```
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;
```

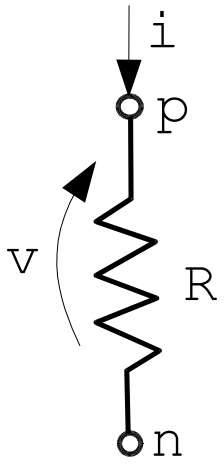



```
model Resistor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Resistance R;
equation
  v = p.v - n.v;
  i = p.i;
  0 = p.i + n.i;
  v = R*i;
end Resistor;
```

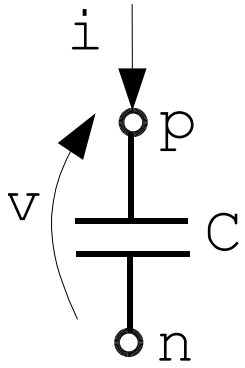
Example Models

```
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;
```

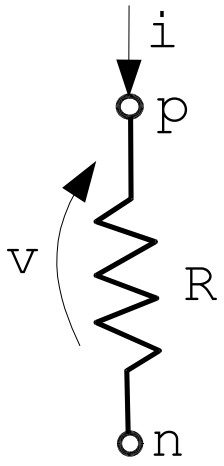



```
model Capacitor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Capacitance C;
equation
  v = p.v - n.v;
  i = p.i;
  0 = p.i + n.i;
  i = C*der(v);
end Capacitor;
```

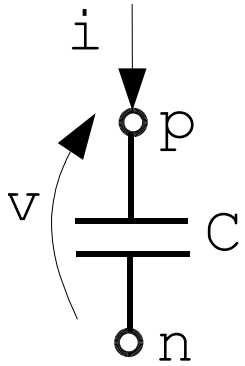
Example Models

```
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;
```



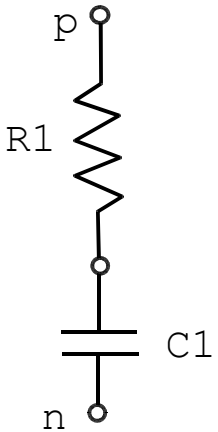

```
model Resistor
  Pin p,n;
  Voltage v;
  Current i;
  parameter Resistance R;
equation
  v = p.v-n.v;
  i = p.i;
  0 = p.i + n.i;
  v = R*i;
end Resistor;
```




```
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

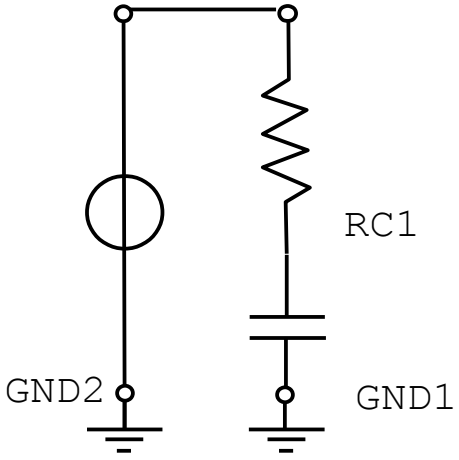



```
model RCNet
  parameter Resistance Rnet;
  parameter Capacitance Cnet;
  Resistor R1(R=Rnet);
  Capacitor C1(C=Cnet);
  Pin p,n;
equation
  connect(R1.n, C1.p);
  connect(R1.p, p);
  connect(C1.n, n);
end RCNet;
```

Modifiers
(parameter propagation)

Equivalent to:

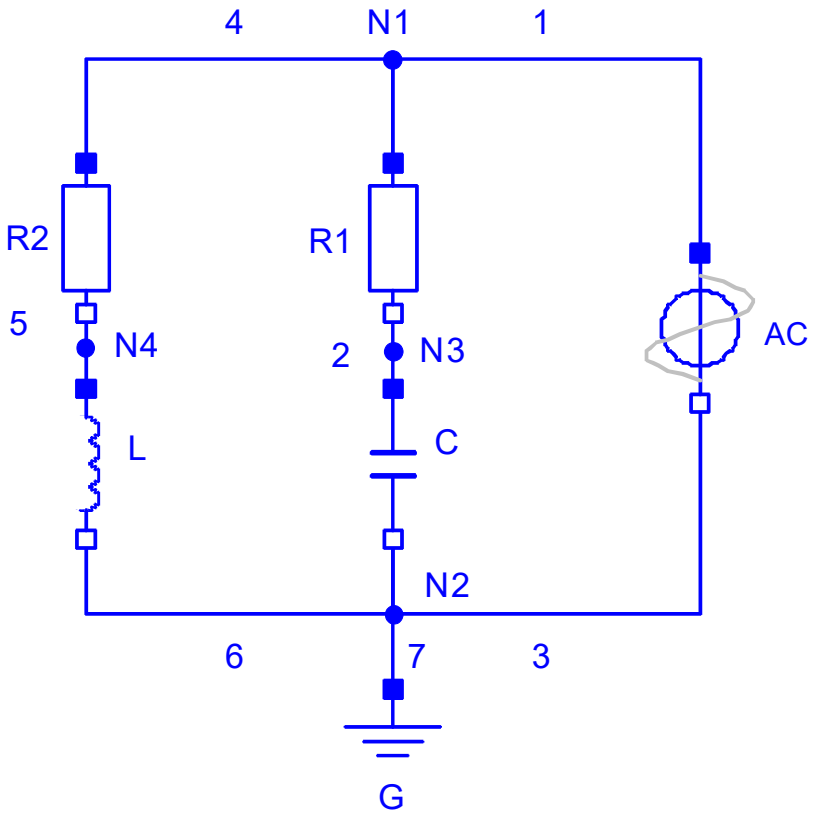
```
R1.n.v = C1.p.v;
R1.n.i + C1.p.i = 0;
```




```
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;
```

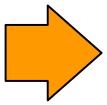
Inheritance: Factoring Out Common Features

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;
```

Hybrid models

- Discrete variables: only change at discrete events, otherwise constant
- Equations for discrete variables inside *when*-clauses, only active at event instants.

```
model OnOff
  parameter Real Threshold;
  RealInput Cmd;
  output Boolean y;
equation
  when (Cmd > Threshold) then
    y = not(pre(y));
  end when;
end OnOff;
```

```
model Stepper
  parameter Real Threshold;
  parameter Real Increment;
  RealInput Cmd;
  discrete RealOutput y;
equation
   when (Cmd > Threshold) then
      y = pre(y) + Increment;
  end when;
end Stepper;
```

- Discrete equations can be freely combined with continuous equations

Computational model: Continuous-time systems

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

Causalization

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

State-Space representation. (ODEs)

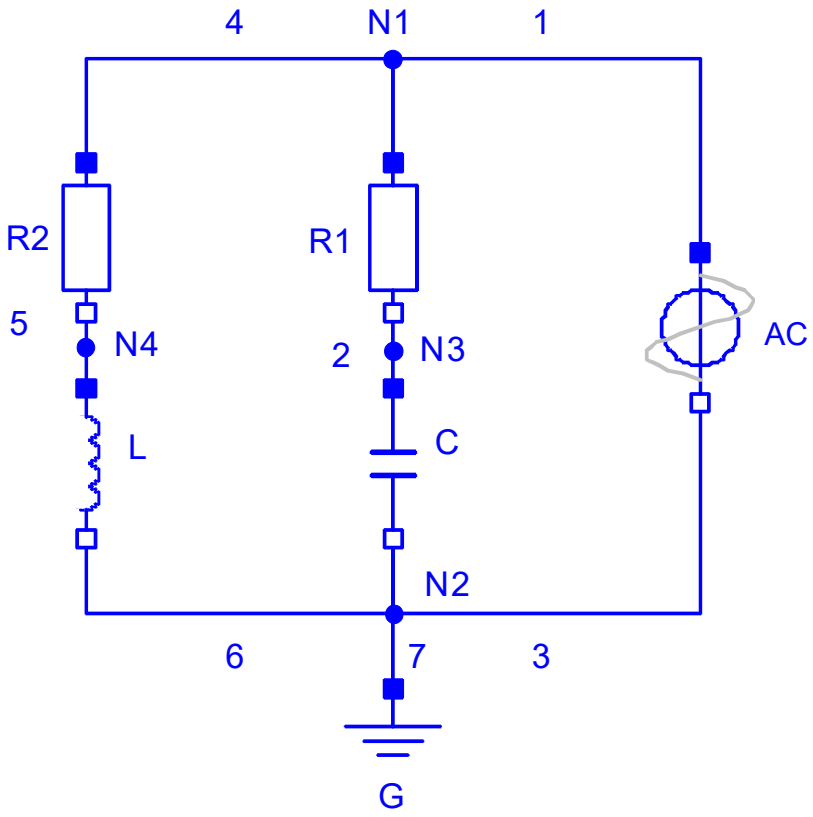
$$\dot{x}=f\left(x, p, t\right)$$
$$v=g\left(x, p, t\right)$$

ODE Time integration

Simulation results

$$x=x\left(t\right)$$
$$v=v\left(t\right)$$

Causalization: Example



Causalization: Example

AC	`0 = AC.p.i+AC.n.i`
	`AC.v = AC.p.v-AC.n.v`
	`AC.i = AC.p.i`
	`AC.v = AC.VA*`
	`sin(2*AC.PI*`
	`AC.f*time)`
		L	`0 = L.p.i+L.n.i`
			`L.v = L.p.v-L.n.v`
			`L.i = L.p.i`
			`L.v = L.L*der(L.i)`

R1	`0 = R1.p.i+R1.n.i`
	`R1.v = R1.p.v-R1.n.v`
	`R1.i = R1.p.i`
	`R1.v = R1.R*R1.i`
		G	`G.p.v = 0`

R2	`0 = R2.p.i+R2.n.i`
	`R2.v = R2.p.v-R2.n.v`
	`R2.i = R2.p.i`
	`R2.v = R2.R*R2.i`
		connections (effort)	`R1.p.v. = AC.p.v // 1`
			`C.p.v = R1.v.v // 2`
			`AC.n.v = C.n.v // 3`
			`R2.p.v = R1.p.v // 4`
			`L.p.v = R2.n.v // 5`
			`L.n.v = C.n.v // 6`
			`G.p.v = AC.n.v // 7`

C	`0 = C.p.i+C.n.i`
	`C.v = C.p.v-C.n.v`
	`C.i = C.p.i`
	`C.i = C.C*der(C.v)`
		connections (flow)	`0 = AC.p.i+R1.p.i+R2.p.i // N1`
			`0 = C.n.i+G.p.i+AC.n.i+L.n.i // N2`
			`0 = R1.n.i+C.p.i // N3`
			`0 = R2.n.i+L.p.i // N4`

Causalization: Example

After removing trivial equations (a = b, a + b = 0)

```
1) C.i = R1.v/R1.R                              // f(R1.v)
2) R1.v = R1.p.v - C.v                          // f(R1.v,R1.p.v)-C.v
3) der(L.i) = L.v/L.L                           // f(L.v,der(L.i))
4) R1.p.v = AC.VA*sin(2*AC.f*AC.PI*time)        // f(R1.p.v)
5) L.v = R1.p.v - R2.v                          // f(L.v,R1.p.v,R2.v)
6) der(C.v) = C.i/C.C                           // f(der(C.v),C.i)
7) R2.v = R2.R*L.i                              // f(R2.v) - L.i
```

	R2.v	R1.p.v	L.v	R1.v	C.i	der(L.i)	der(C.v)
1)	0	0	0	1	1	0	0
2)	0	1	0	1	0	0	0
3)	0	0	1	0	0	1	0
4)	0	1	0	0	0	0	0
5)	1	1	1	0	0	0	0
6)	0	0	0	0	1	0	1
7)	1	0	0	0	0	0	0

Causalization: Example

After applying the Block Lower Triangular Transformation

	R2.v	R1.p.v	L.v	R1.v	C.i	der(L.i)	der(C.v)
7)	1	0	0	0	0	0	0
4)	0	1	0	0	0	0	0
5)	1	1	1	0	0	0	0
2)	0	1	0	1	0	0	0
1)	0	0	0	1	1	0	0
3)	0	0	1	0	0	1	0
6)	0	0	0	0	1	0	1

```
7) R2.v := R2.R*L.i
4) R1.p.v := AC.VA*sin(2*AC.f*AC.PI*time)
5) L.v := R1.p.v - R2.v
2) R1.v := R1.p.v - C.v
1) C.i := R1.v/R1.R
3) der(L.i) := L.v/L.L
6) der(C.v) := C.i/C.C
```

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

Causalization: General Case

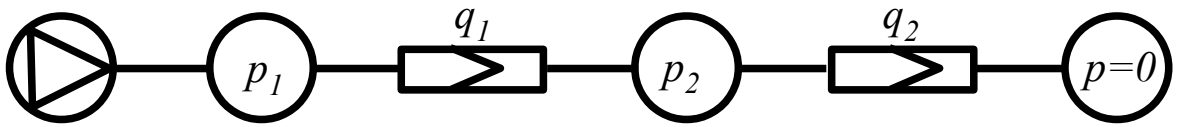
NxN blocks can show up on the diagonal, $N > 1$ (*algebraic loops*)

Systems of implicit equations, solved numerically and/or symbolically

$$
\begin{array}{c} \\ f_2 \\ f_4 \\ f_3 \\ f_5 \\ f_1 \end{array}
\begin{array}{c}
\begin{array}{ccccc} z_2 & z_1 & z_3 & z_5 & z_4 \end{array} \\
\begin{pmatrix}
\mathbf{1} & 0 & 0 & 0 & 0 \\
1 & \mathbf{1} & 0 & 0 & 0 \\
1 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & 1 & \mathbf{1} & \mathbf{1} & 0 \\
0 & 0 & 1 & 0 & \mathbf{1}
\end{pmatrix}
\end{array}
$$

Symbolic Index Reduction

- O-O model building can lead to DAEs with constraints among states
- The DAEs cannot be solved for all the derivatives



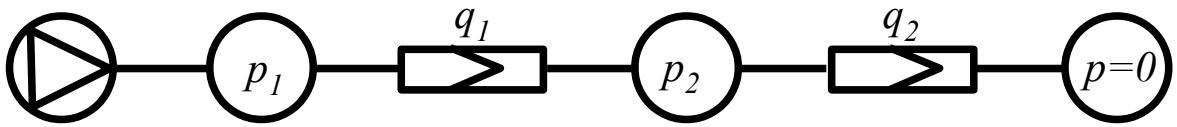
$$C_1\dot{p}_1-(q_0-q_1)=0$$
$$C_2\dot{p}_2-(q_1-q_2)=0$$
$$q_0-f(t)=0$$
$$p_1-p_2=0$$
$$p_2-p_3-R_2q_2=0$$

$$\left[\frac{\partial F}{\partial z}\right]=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&R_2\end{bmatrix}$$

Structural Singularity!
(high-index DAEs)

Symbolic Index Reduction

- O-O model building can lead to DAEs with constraints among states
- The DAEs cannot be solved for all the derivatives



$$C_1\dot{p}_1-(q_0-q_1)=0$$
$$C_2\dot{p}_2-(q_1-q_2)=0$$
$$q_0-f(t)=0$$
$$p_1-p_2=0$$
$$p_2-p_3-R_2q_2=0$$

$$\left[\frac{\partial F}{\partial z}\right]=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&R_2\end{bmatrix}$$

Structural Singularity! (high-index DAEs)

- Pantelides' Algorithm and Dummy-Derivatives Algorithm

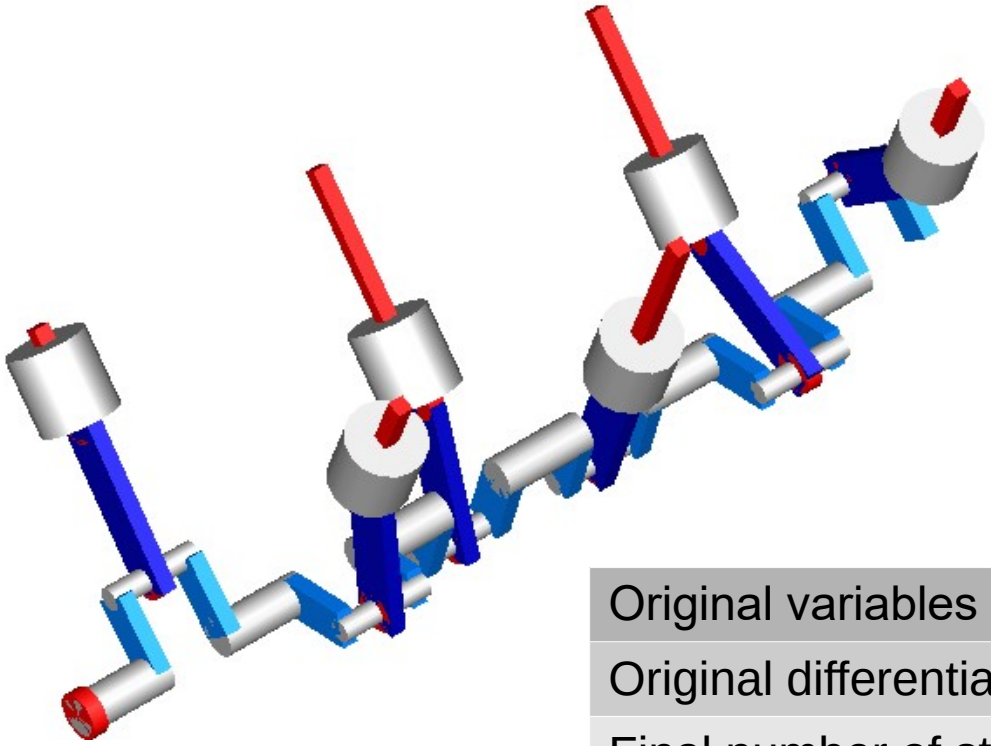
$$C_1\dot{p}_1=(q_0-q_1)$$
$$C_2\dot{p}_2=(q_1-q_2)$$
$$q_0-f(t)=0$$
$$p_1-p_2=0$$
$$p_2-R_2q_2=0$$
$$\dot{p}_1-\dot{p}_2=0$$

$$C_1\dot{p}_1=(q_0-q_1)$$
$$C_2\dot{p}_{2\text{der}}=(q_1-q_2)$$
$$q_0-f(t)=0$$
$$p_1-p_2=0$$
$$p_2-R_2q_2=0$$
$$\dot{p}_1-p_{2\text{der}}=0$$

$$\dot{p}_1=\frac{f(t)-p_2/R_2}{C_1+C_2}$$
$$p_2=p_1$$
$$p_{2\,der}=\frac{f(t)-p_2/R_2}{C_1+C_2}$$
$$q_0=f(t)$$
$$q_1=\frac{C_2q_0+C_1q_2}{C_1+C_2}$$
$$q_2=\frac{p_2}{R_2}$$

A Successful Use Case

- Multibody model of a V6 Engine from the Modelica Standard Library



$$\left.\begin{aligned} \dot{x} &= f(x, p, t) \\ v &= g(x, p, t) \end{aligned}\right\}$$

Original variables	12491
Original differentiated variables	577
Final number of states	2 (!)
Linear systems	12x2 + 1x36
Nonlinear systems	6x2
Assignments	1334

Computational model: Hybrid systems

DAEs $$F\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)=0$$

Conditions $$c=G\left(relation\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)\right)$$

Discrete eqs $$m=H\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)$$

Computational model: Hybrid systems

Continuous-time Integration

DAEs $$F\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)=0$$

Conditions $$c=G\left(relation\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)\right)$$

Discrete eqs $$m=H\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)$$

- DAEs are solved with constant m
- Conditions are monitored for changes

Computational model: Hybrid systems

Event detection

DAEs $$F\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)=0$$

Conditions $$c=G\left(relation\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)\right)$$

Discrete eqs $$m=H\left(x, \dot{x}, v, m, pre\left(m\right), p, c, t\right)$$

- When conditions changes are detected, the exact event time is numerically computed
- E.g.:
  - Condition $v_1 > v_2$
  - Zero Crossing function $f_{zc}(t) = v_1(t) - v_2(t)$
  - Find t such that $f_{zc}(t) = 0$

Computational model: Hybrid systems

Event handling

DAEs $F(x, \dot{x}, v, m, pre(m), p, c, t) = 0$

Conditions $c = G(relation(x, \dot{x}, v, m, pre(m), p, c, t))$

Discrete eqs $m = H(x, \dot{x}, v, m, pre(m), p, c, t)$

1) Solve DAEs and *active* discrete equations simultaneously for $\dot{x}, v, m$
2) Let $m = pre(m)$
3) Solve DAEs and *active* discrete equations simultaneously for $\dot{x}, v, m$
4) If $m = pre(m)$ then resume continuous time integration with constant m
   else goto 2

Clocked Variables

- Introduced in Modelica 3.3, based on synchronous language concepts (Lustre, Lucid Synchrone)

- Clock inference based on structural dependency analysis.
  sample() and hold() break the dependency

Clocked Variables

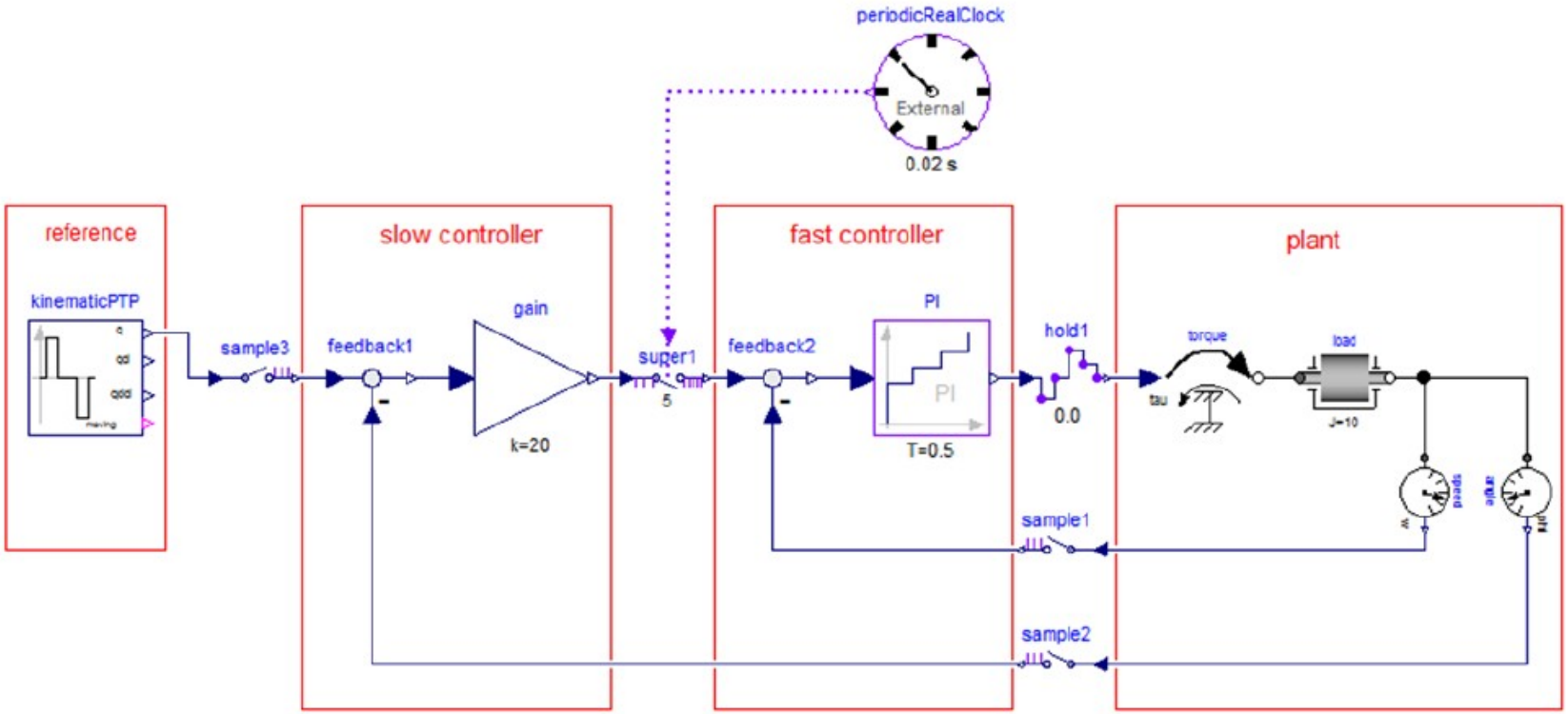
- Introduced in Modelica 3.3, based on synchronous language concepts (Lustre, Lucid Synchrone)

- *Clock inference* based on structural dependency analysis. sample() and hold() break the dependency graphs

- Automatic clock partitioning
  - {v1, v2, v3, v4} continuous time
  - {vc1, vc2} clocked Tc
  - {vc3, vc4} clocked Tc*4

```
  ...
  Real v1, v2, v3, v4;
  Real vc1, vc2, vc4, vc5;
  parameter Real Tc = 0.1;
equation
  ...
  vc1 = sample(v1, Clock(Tc));
  when Clock() then
    vc2 = previous(vc2) + vc1;
  end when;
  vc3 = subSample(vc2, 4);
  vc4 = 3*vc3;
  v2 = hold(vc2);
  v4 = hold(vc4);
  v5 + v4 = 0;
  ...
```

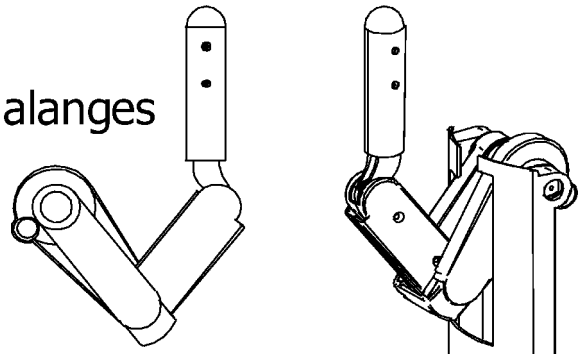
Clocked Variables: Use in Modular Models

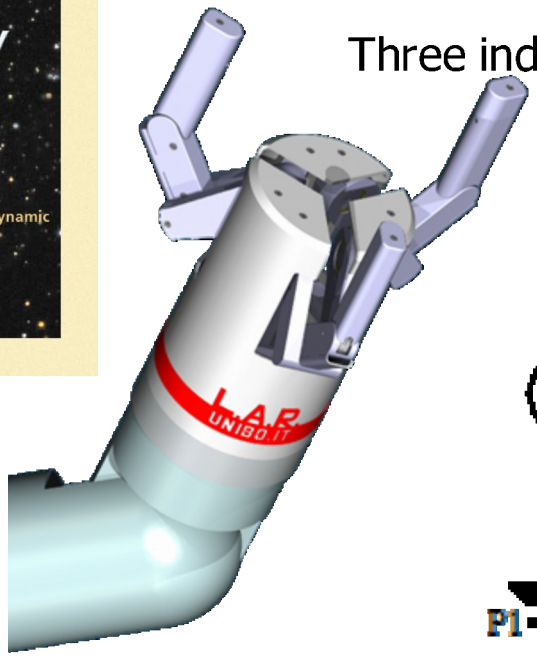


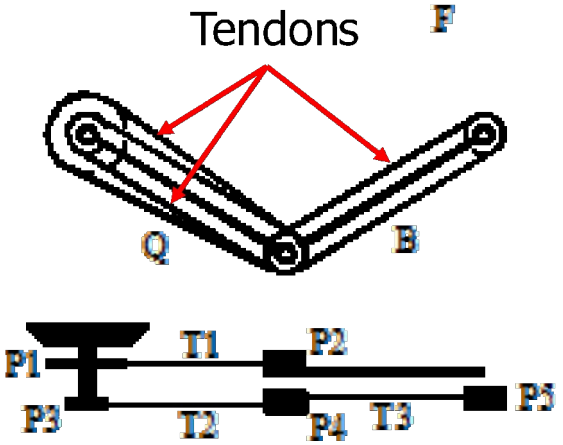
M. Otter, B. Thiele, H. Elmqvist: A library for Syncronous Control Systems in Modelica
Proceedings 9th Modelica Conference, Munich 2012

Paradigmatic Use Cases

A Gripper for Space Robotics





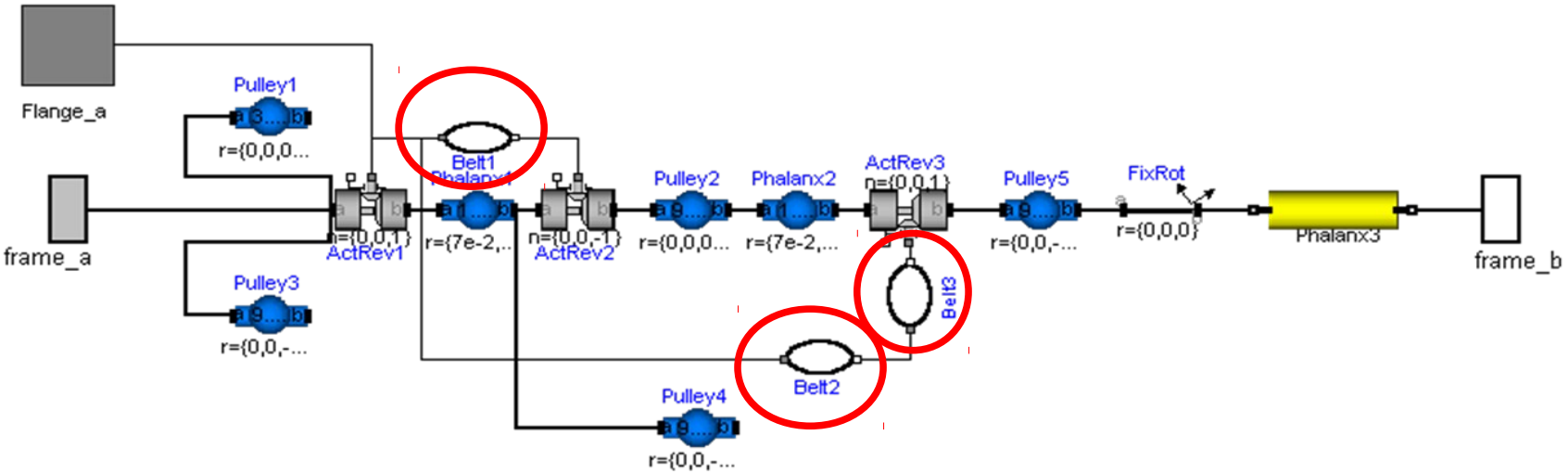


Courtesy: prof. Gianni Ferretti, Politecnico di Milano

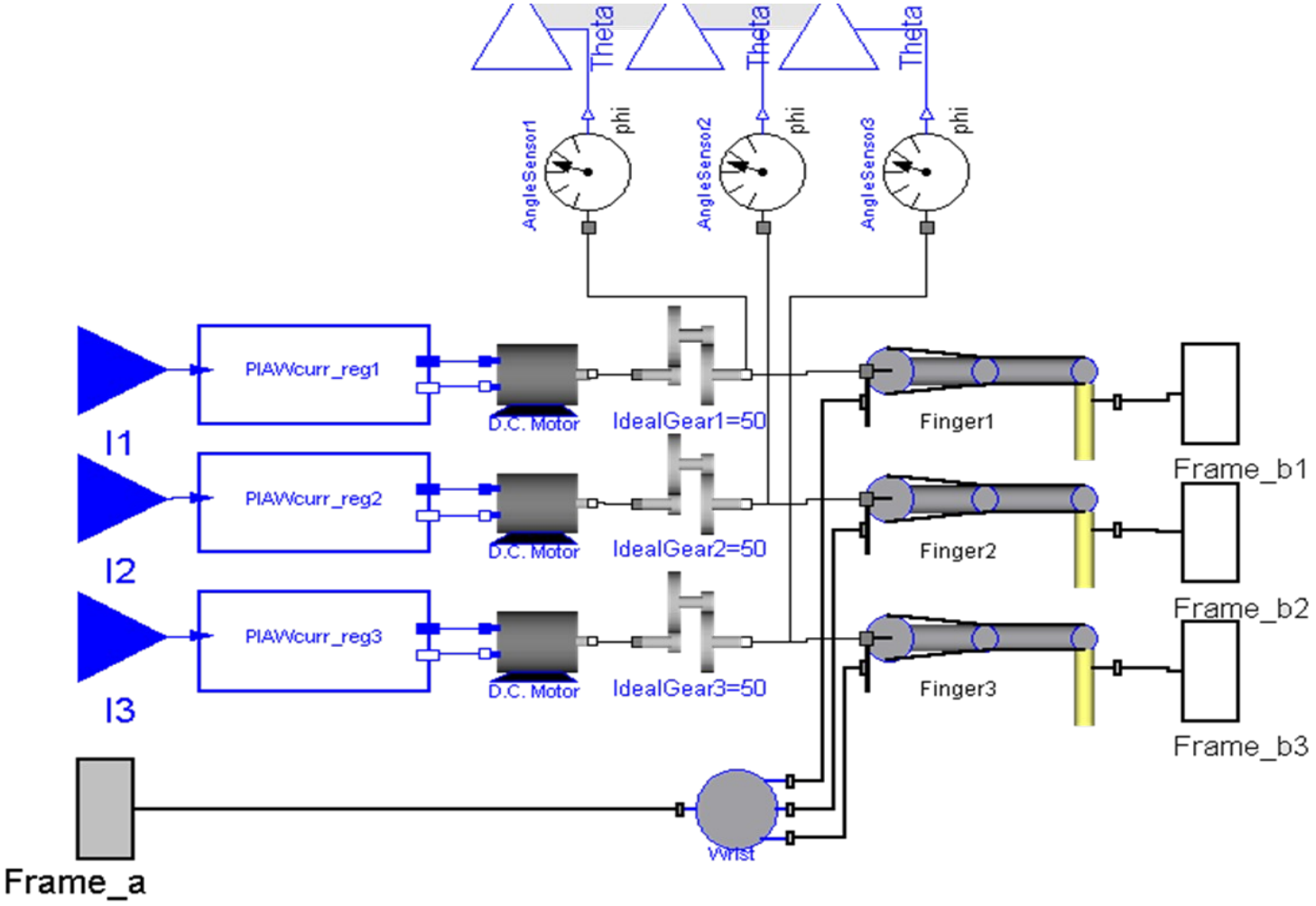
2

The Finger Model

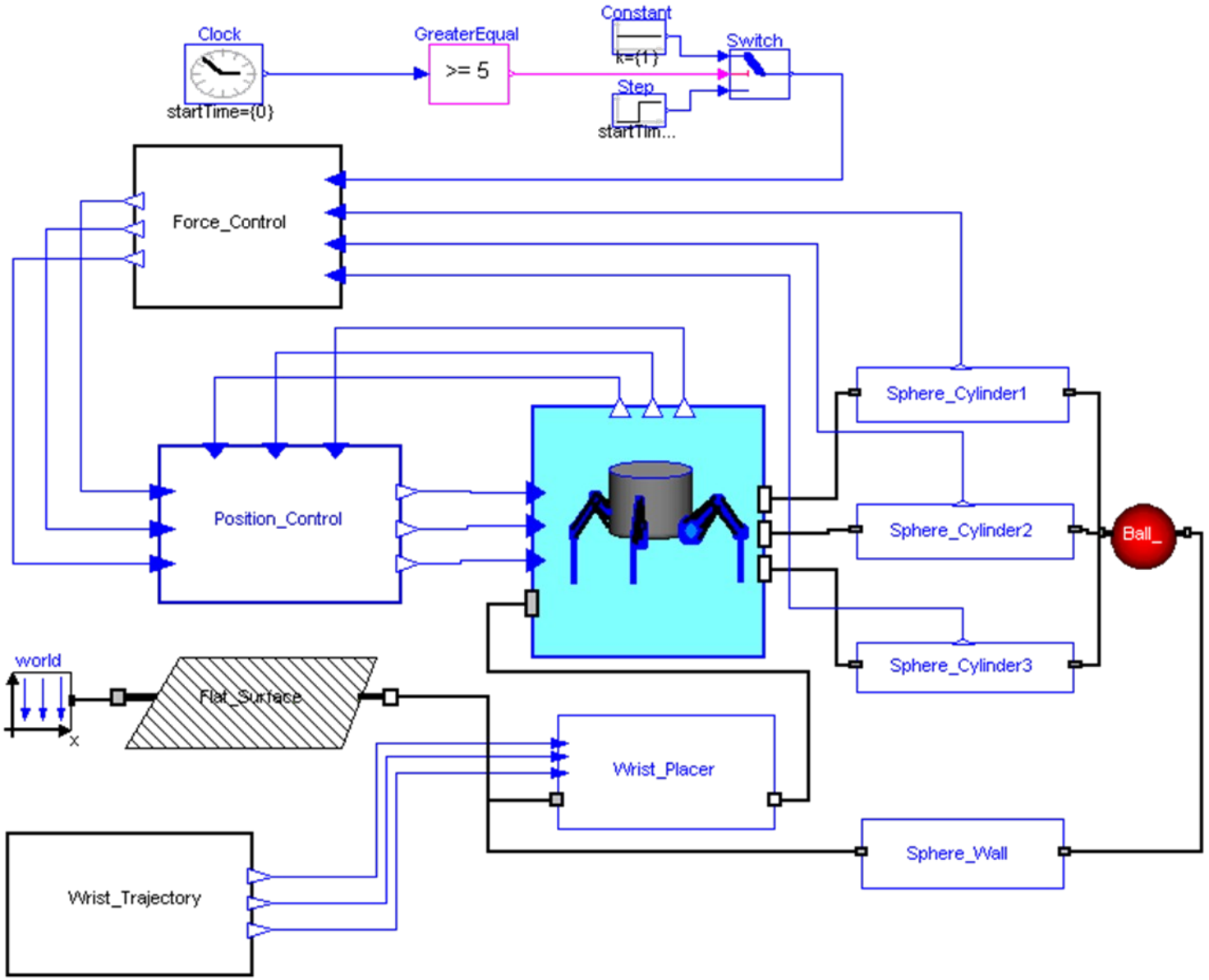
- Use of the Modelica.Mechanics.MultiBody library
- New components developed for tendon-pulley interactions described by equations



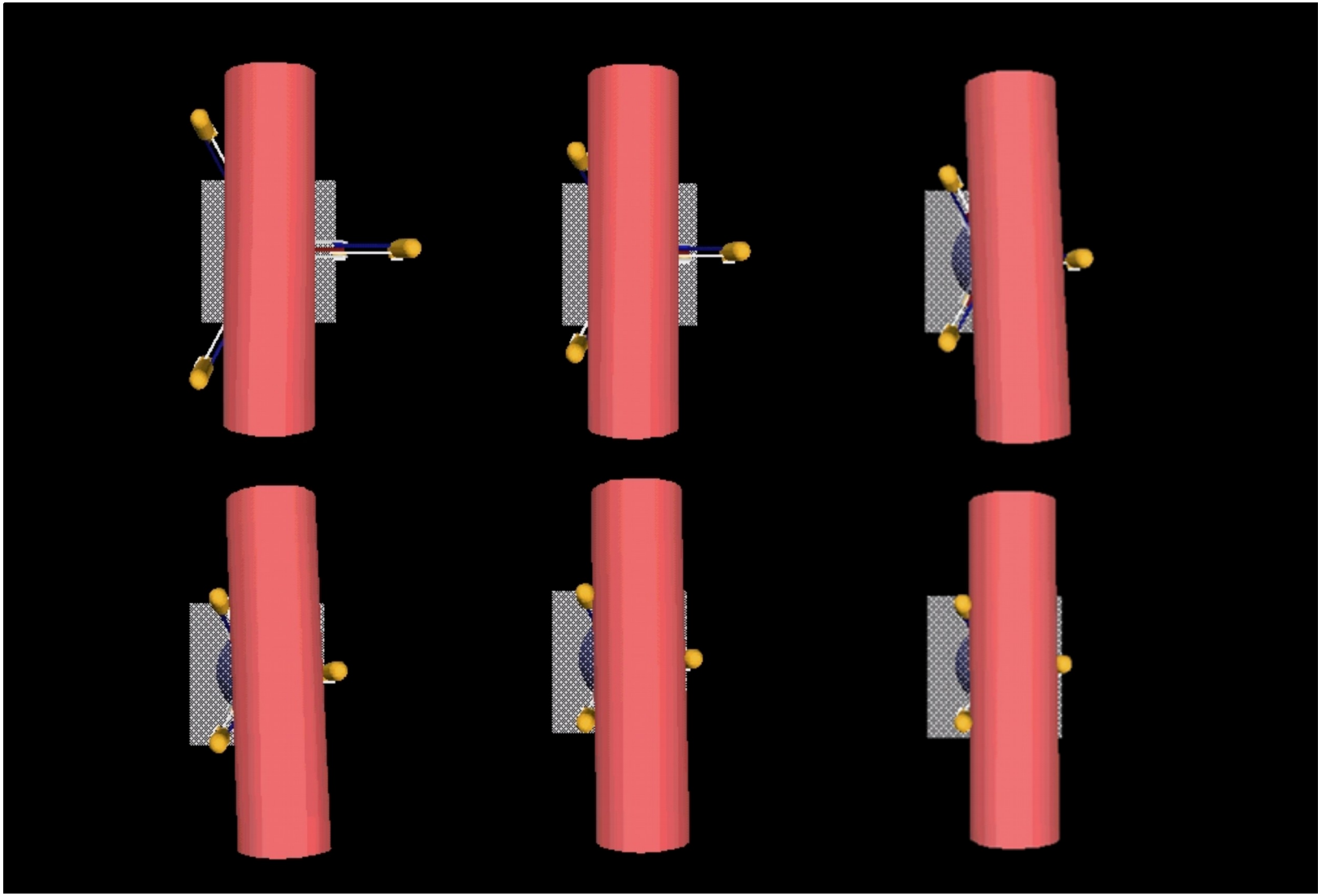
The Actuation Chain



The Overall System Model

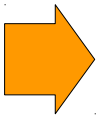


Grasping a Cylinder in Space

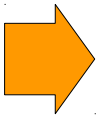


OOM and Reusability: Do Not Reinvent the Wheel!

- Development effort focused on innovative components
  - Tendon–Pulley Interaction
  - Finger–Sphere interaction
  - Control System strategy and design
- Re-use of standard Modelica.Mechanics.MultiBody library framework and components



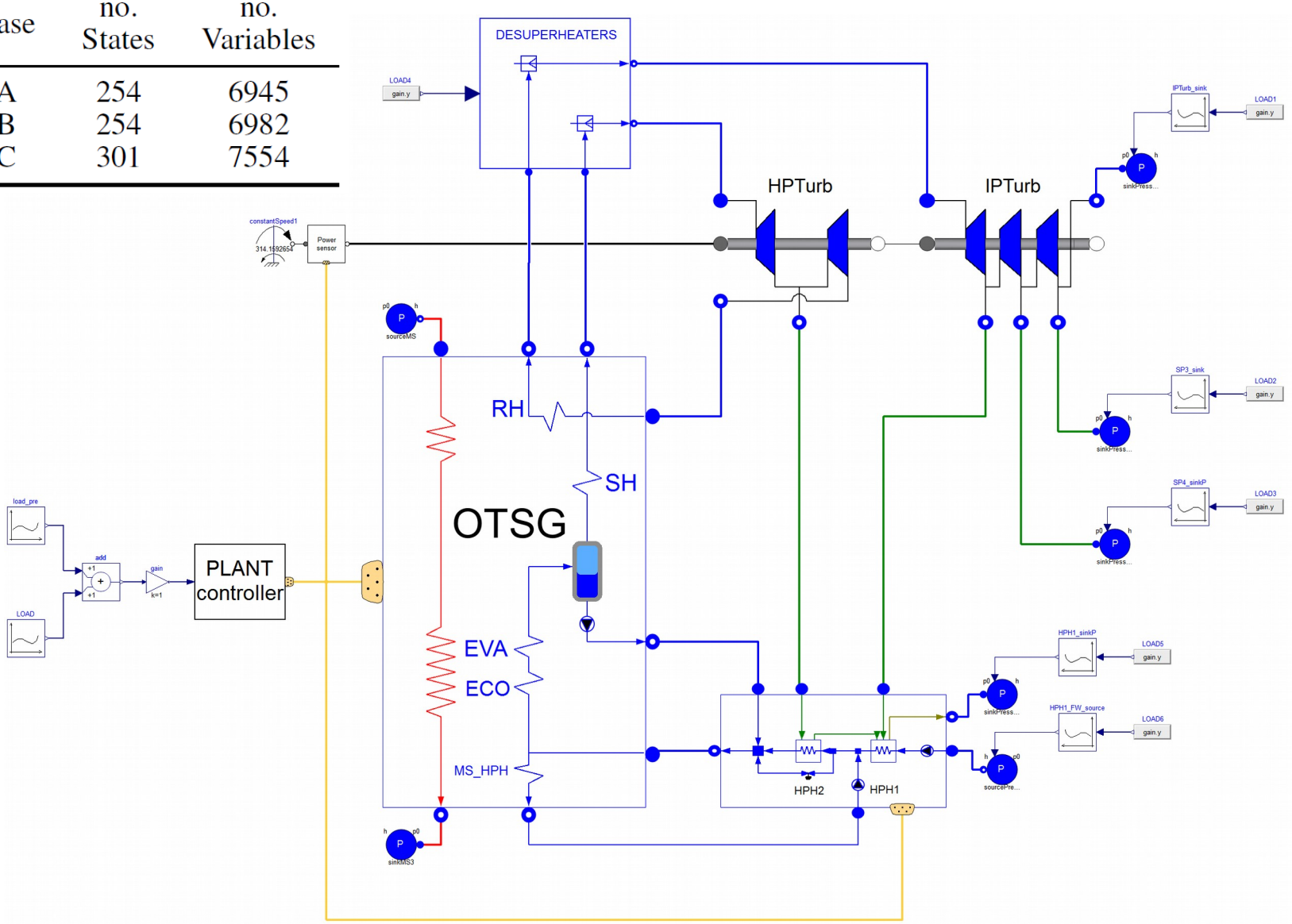
20% New models



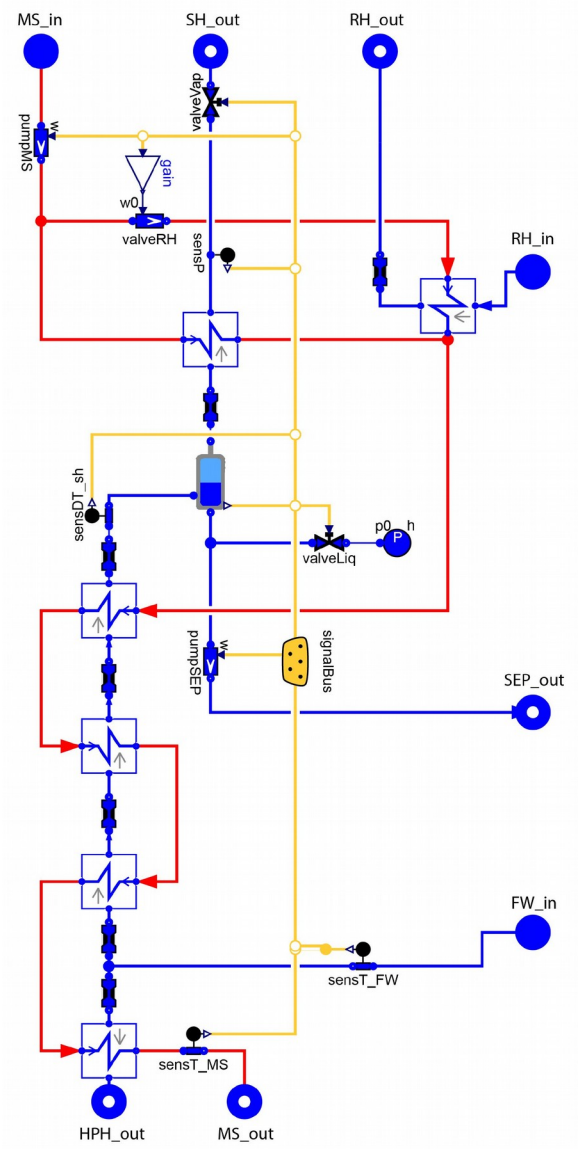
80% Re-used models

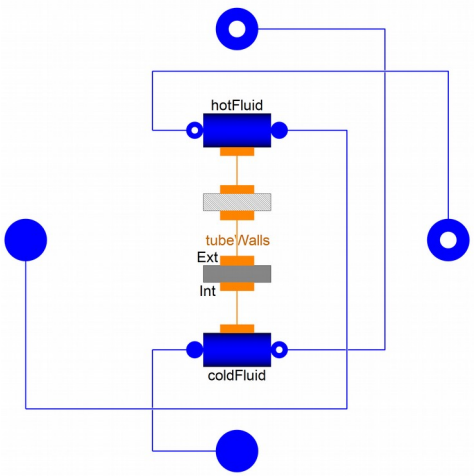
A Once-Through Molten-Salt Power Generation System

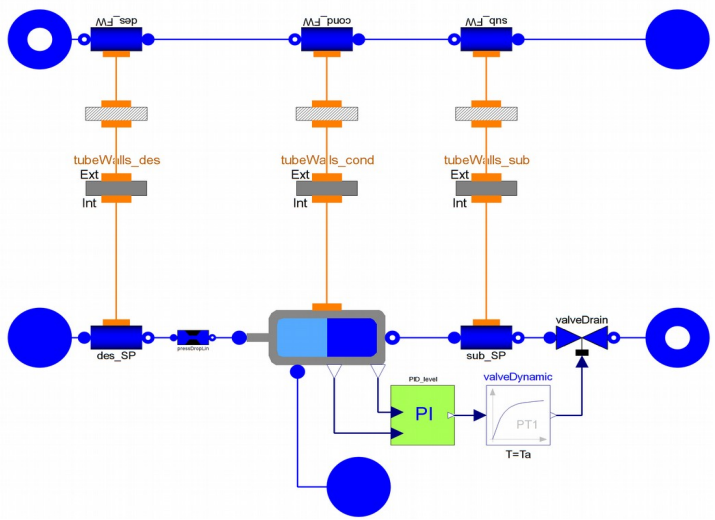
Case	no. States	no. Variables
A	254	6945
B	254	6982
C	301	7554



Modular Decompostion

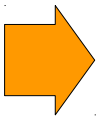




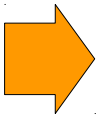


OOM and Reusability

- Effort concentrated in the innovative components
  - Molten Salt fluid model
  - Preheaters
  - Phase Separator
- Re-use of ThermoPower library framework and components



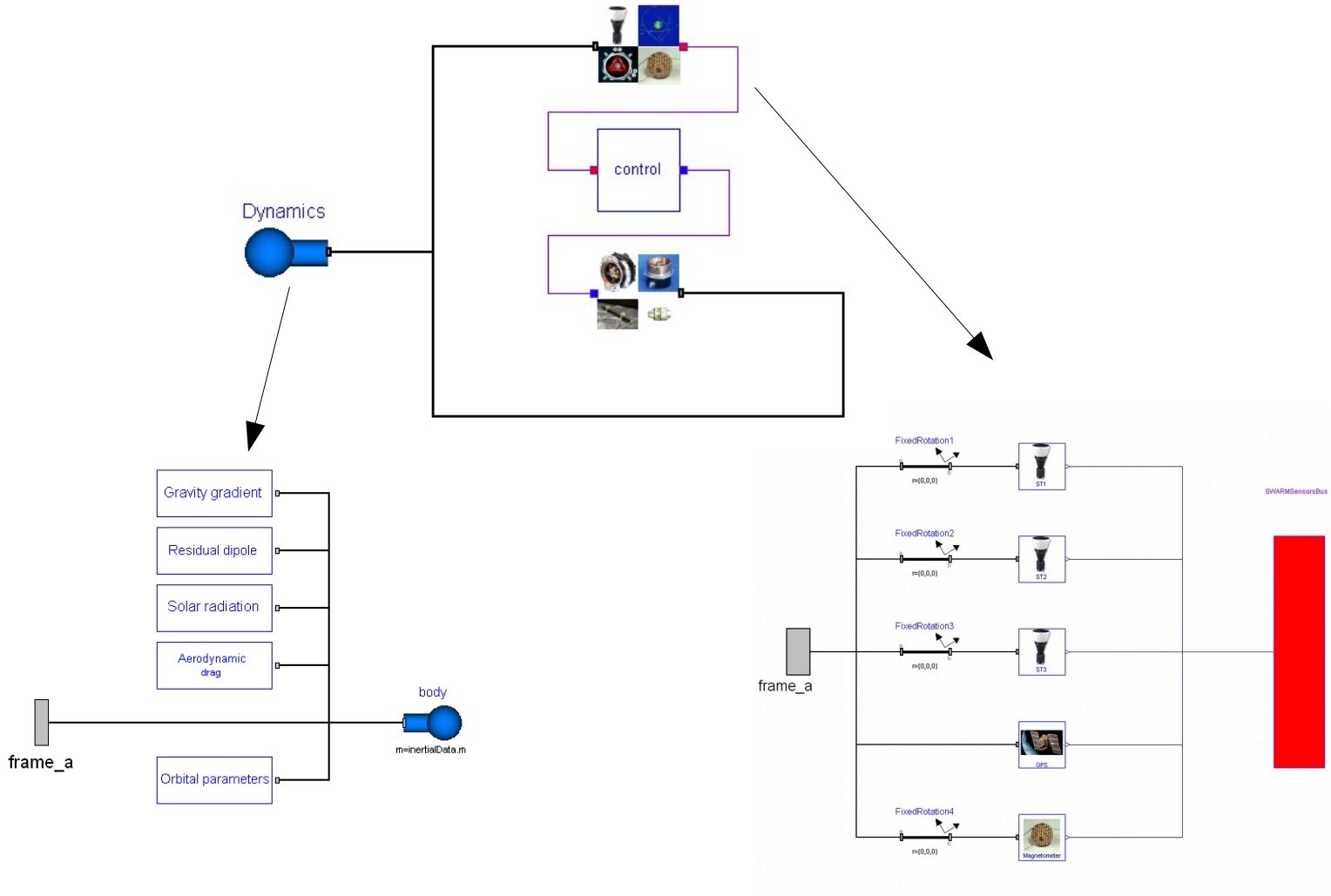
20% New models



80% Re-used models

Replaceable Models: a Space Application

Modelling of Satellite Attitude Control Systems
with reconfigurable sensor and actuator configurations



Replaceable Models: a Space Application

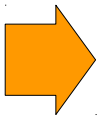
- Standard interfaces defined for replaceable components
- Very concise representation of a specific configuration
  - alternative design configurations
  - different levels of detail @ different design stages
- All redeclared elements taken from a library of components

```
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))));
end Example;
```

Replaceable Models: a Space Application

- Standard interfaces defined for replaceable components
- Very concise representation of a specific configuration
  - alternative design configurations
  - different levels of detail @ different design stages
- All redeclared elements taken from a library of components

```
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))));
end Example;
```

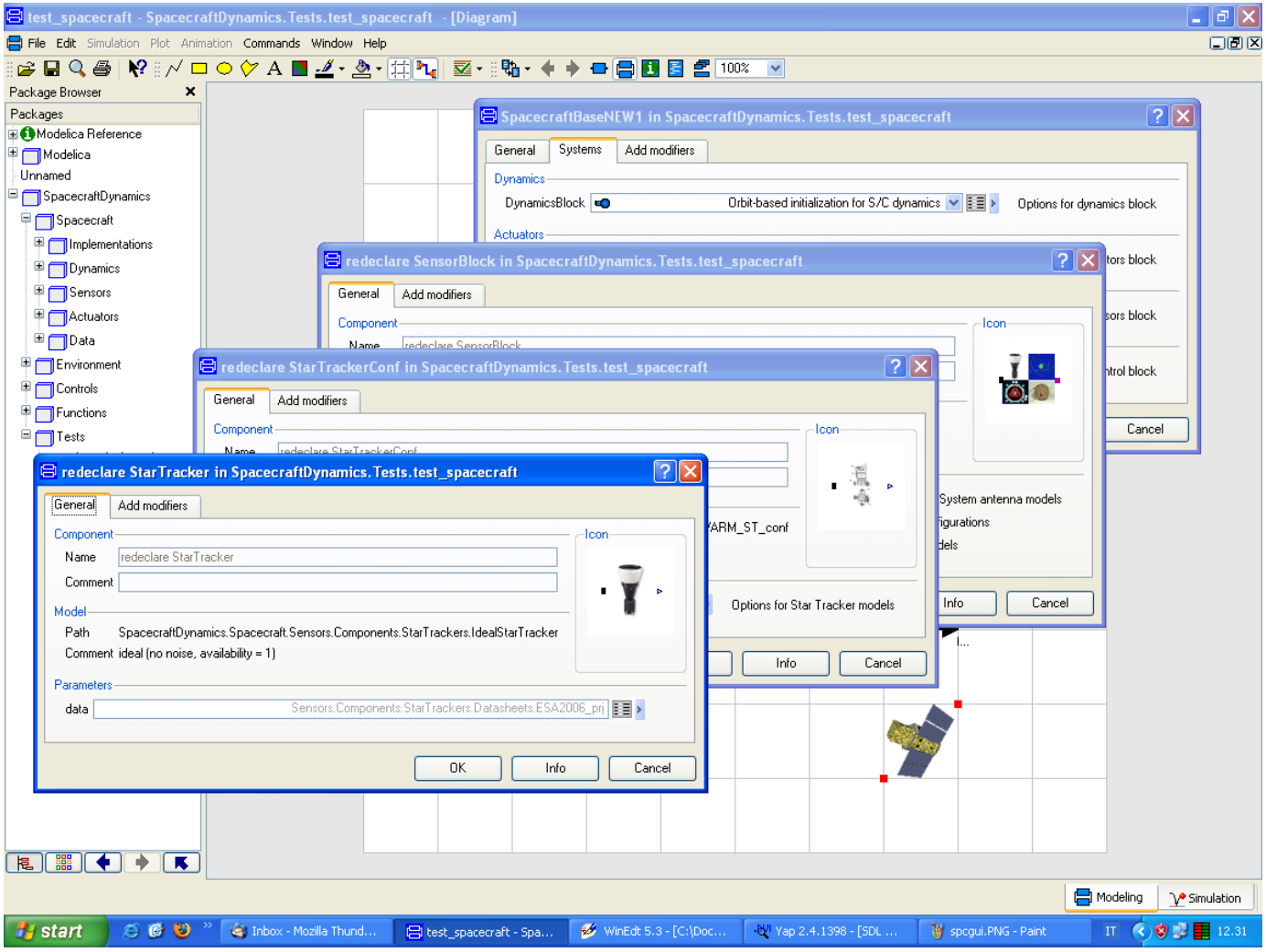


All model variants are consistent and up-to-date

Avoid copy-paste-modify-use-throw away cycle

Replaceable Models: a Space Application

GUI support for model configuration (no manual writing of code)



The Future

Emerging Large-Scale Engineering Systems

- Electrical Smart Grids, including thermal users and storage (heat pumps, solar thermal systems)
- Self-driving Cars
- Cyber-Physical Systems
- Internet of Things

Emerging Large-Scale Engineering Systems

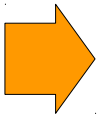
- Electrical Smart Grids, including thermal users and storage (heat pumps, solar thermal systems)
- Self-driving Cars
- Cyber-Physical Systems
- Internet of Things

- System (and control!) design based on abstractions
- Correct behaviour depends on etherogeneous, multi-domain physical behaviour
- Things can go wrong and break design assumptions

Emerging Large-Scale Engineering Systems

- Electrical Smart Grids, including thermal users and storage (heat pumps, solar thermal systems)
- Self-driving Cars
- Cyber-Physical Systems
- Internet of Things

- System (and control!) design based on abstractions
- Correct behaviour depends on etherogeneous, multi-domain physical behaviour
- Things can go wrong and break design assumptions

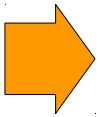


Simulation-based verification of system performance with adequate physical detail

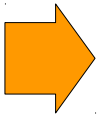
OOM and Modelica are ideally suited (current tools still aren't)

The Challenge

- Focus of currently available Modelica tools: individual systems
  - One/two robots
  - One power plant
  - One car
- Typical complexity today: 500 to 20000 equations



Handle 10,000,000 equations or more

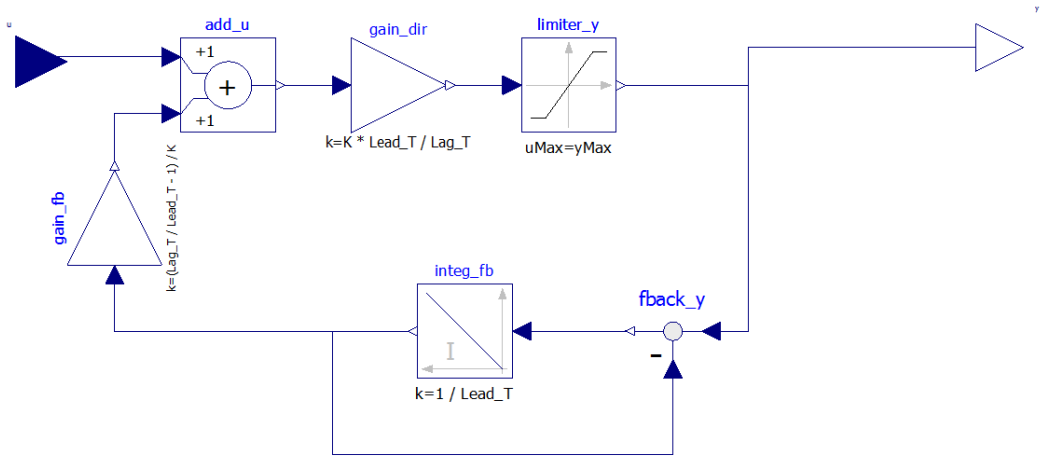


Develop appropriate numerical integration algorithms for these systems

Example: Models of Continental PG&T Grids

```
equation
// mechanical equations
  Snom_GC_mod*Ta*der(omega)/omega = Pm_req - Pe;
  der(delta) = omega - omega_ref;
// lead-lag vd
  Tqo*der(ed) + ed = (Xq - X)*iq;
  ed = vd - X * iq;
//lead-lag  + lag vq
  Tdo*der(eq) + eq = vf - (Xd - X)*id;
  eq = vq + X * id;
//normalization
  Vd = vd*V_ll_nom_mod;
  Vq = vq*V_ll_nom_mod;
  Id = id*Snom_GC_mod/V_ll_nom_mod;
  Iq = iq*Snom_GC_mod/V_ll_nom_mod;
// conversion from Park ref. to pin ref.
  Vpr_ll =  Vd*sin(delta) + Vq*cos(delta);
  Vpi_ll = -Vd*cos(delta) + Vq*sin(delta);
  Ipr =  Id*sin(delta) + Iq*cos(delta);
  Ipi = -Id*cos(delta) + Iq*sin(delta);
// power calculation
  Pe = 3 * (Vpr*Ipr + Vpi*Ipi);
  Qe = 3 * (Ipr*Vpi - Vpr*Ipi);
```

```
algorithm
// Detection of high current - side a
  when I_a_mod > Ilmax_mod then
    TimerOn_a := true;
    TimerStartValue_a := time;
  end when;
  when I_a_mod < Ilmax_mod and pre(TimerOn_a) then
    TimerOn_a := false;
  end when;
algorithm
// Handles the actual status of the breaker - side a
  when pre(TimerOn_a) and
       time > pre(TimerStartValue_a)+Ilmax_delay then
    BreakerStatus_a := 0;
  end when;
equation
  Yl_act = Yl * Complex(BreakerStatus_a * BreakerStatus_b);
  Ysa_act = Ys * Complex(BreakerStatus_a);
  Ysb_act = Ys * Complex(BreakerStatus_b);
  Ia = Il + Isa;
  Il + Ib = Isb;
  Isa = Ysa_act * Va;
  Isb = Ysb_act * Vb;
  Il = Yl_act * Vl;
  Va = Vl + Vb;
```



Courtesy: Politecnico di Milano, Dynamica, CESI, TERNA

Example: Models of Continental PG&T Grids

- Improvements of the OpenModelica compiler already achieved the 1,000,000 equations goal
- Realistic use cases:

Network	Generators	Lines	Transformers	Equations
RETE_C	74	369	583	56386
RETE_E	267	1458	1202	157022
RETE_D	2317	1946	2489	579470
RETE_G	407	6833	2824	593886

PERFORMANCE RESULTS, TIMES GIVEN IN S

Network	Flatt.	C-code Gen.	Exec. Gen.	Sim.
RETE_C	24	24	13	12
RETE_E	73	67	35	44
RETE_D	334	315	123	111
RETE_G	318	303	144	186

- One order of magnitude improvement expected by end 2017 in model building time (12 min → 1-2 min)

Multi-Rate Integration Algorithms

- Features of typical large-scale distributed etherogenous systems
  - Local connectivity
  - Dynamic decoupling between distant units
  - Localized activity
  - Co-existing different time scales
- Single-rate integration algorithms are inefficient:
  entire system state vector computed at the pace of the fastest component

Multi-Rate Integration Algorithms

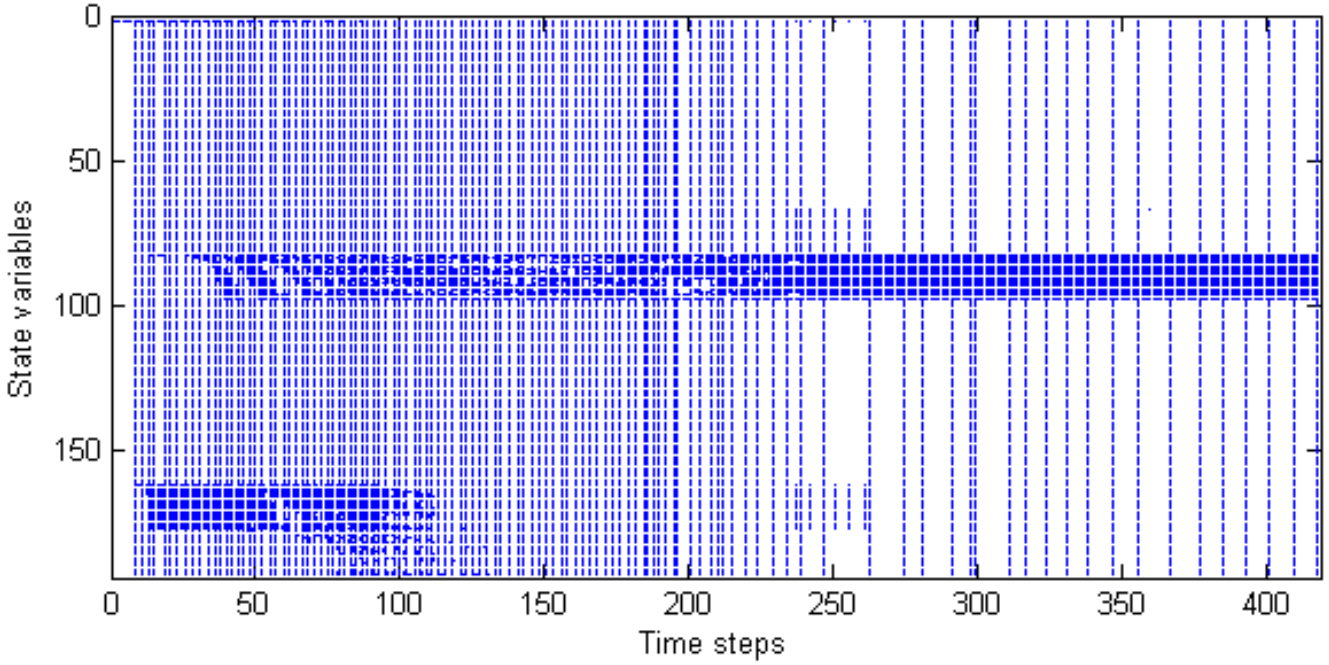
- Features of typical large-scale distributed etherogenous systems
  - Local connectivity
  - Dynamic decoupling between distant units
  - Localized activity
  - Co-existing different time scales
- Single-rate integration algorithms are inefficient:
  entire system state vector computed at the pace of the fastest component

- Multi-rate algorithms:
  - Global time steps
  - *Automatic* partitioning in active and latent states based on error estimation
  - Iterative refinements of active states using interpolated latent state values

Example: Transient of a Power System

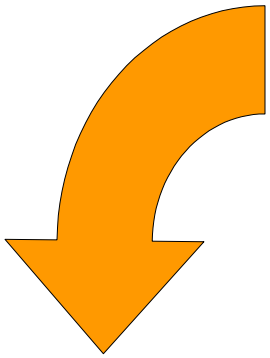
- Many geneators and loads
- Very fast electrical phenomena (swing equation)
- Fast turbine dynamics
- Slow thermo-hydraulic phenomena (boiler dynamics)
- Control loops with widely different bandwidths

- Activity diagram of a load shedding transient

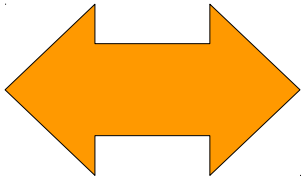


The future of EOOM as I see it

Modelica models of
increasingly large and complex
cyber-physical systems



EOOM Tools with
innovative methods
handling 10,000,000 or
100,000,000 equations



New multi-rate algorithms
exploiting dynamic
decoupling, local activity
and different time scales

Acknowledgments

Prof. Claudio Maffezzoni

Colleagues and PhD Students @ Politecnico:

Alberto Leva, Gianni Ferretti, Marco Lovera
Tiziano Pulecchi, Stefano Trabucchi, Akshay Ranade

Prof. Peter Fritson of Linköping University
and the whole OpenModelica Development Team

All the colleagues of the Modelica Association

Thank you
for you kind attention!