Using OpenModelica for Bleeding-Edge System Simulations of Large Energy System *Models*

Francesco Casella

(francesco.casella@polimi.it)

OpenModelica

Outline

- Motivation
- The Delta Robot and Stewart Platform
- An Advanced Fuell Cell Model
- The SOS-CO2 Oxy-Combustion Power Plant Model
- Generation and Transmission System Modelling with PowerGrids 2.0.0
- Conclusions & Outlook

Motivation

Why Using OpenModelica for Serious Modelling Work?

Why Using OpenModelica for Serious Modelling Work?

- *You don't pay a license fee*

Why Using OpenModelica for Serious Modelling Work?

- *You don't pay a license fee*
- *You can release your work fully open-source*
  - *Scientific publications*
  - *Increased credibility*
  - *Collaborative development with OS model*

Why Using OpenModelica for Serious Modelling Work?

- *You don't pay a license fee*
- *You can release your work fully open-source*
  - *Scientific publications*
  - *Increased credibility*
  - *Collaborative development with OS model*
- *You can look under the hood, and change stuff*

Why Using OpenModelica for Serious Modelling Work?

- *You don't pay a license fee*
- *You can release your work fully open-source*
  - *Scientific publications*
  - *Increased credibility*
  - *Collaborative development with OS model*
- *You can look under the hood, and change stuff*
- *You can try innovative approaches*

The Even Bigger Question

DOES IT WORK?

Case Study #1:

The Delta Robot & The Stewart Platform

Credits

Object-Oriented Models of Parallel Manipulators

Paolo Campanini[1] Gianni Ferretti[2]

[1]MUSP Lab, Strada Torre della Razza, 29122 Piacenza, Italy, paolo.campanini@musp.it
[2]Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Piazza Leonardo da Vinci 32, 20133 Milano, Italy, gianni.ferretti@polimi.it

Abstract

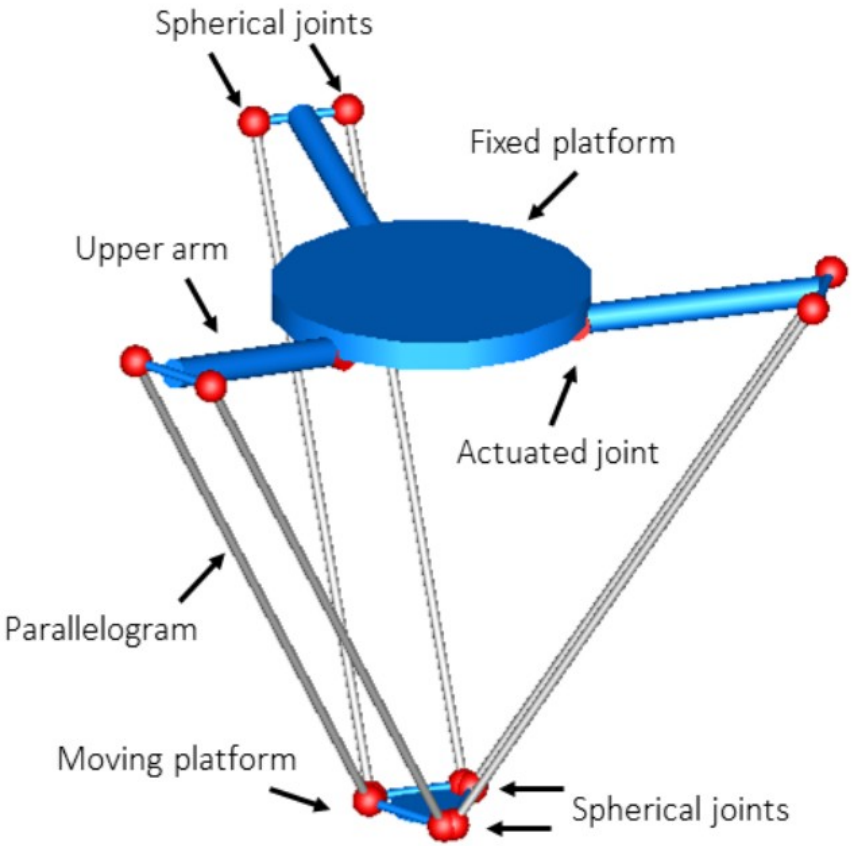
In this paper, the development of models of parallel manipulator is described, based on components of the Modelica standard library only. At first, the dynamic model of a Delta robot is illustrated and validated with respect to experimental data. Then, the model of a Stewart platform is discussed. Thanks to the symbolic manipulation capabilities of the OpenModelica compiler, the model is then used to automatically generate the inverse dynamics, which is in general is a quite difficult task.

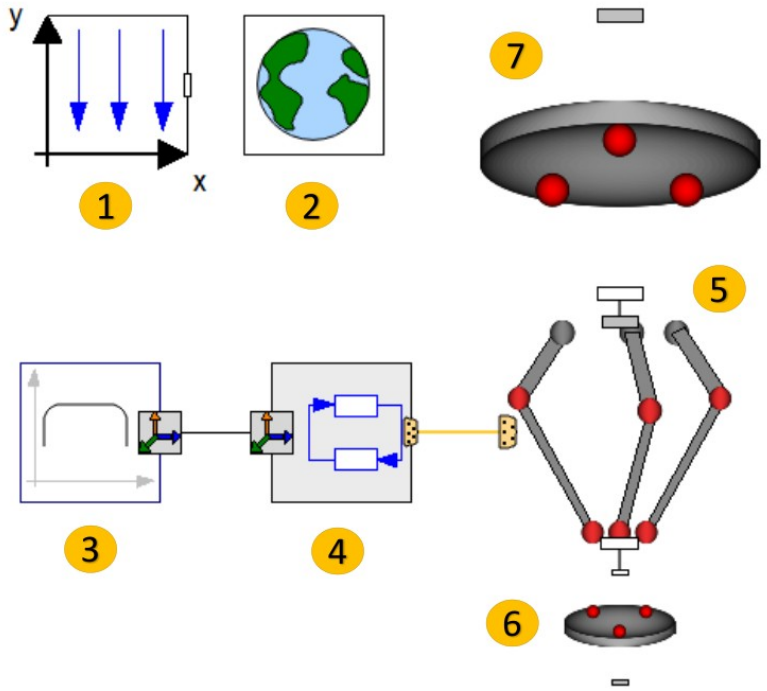
Keywords: Object-oriented modelling; simulation; parallel manipulators; Modelica; DAE systems; closed chains

DOI 10.3384/ecp21181241 Proceedings of the 14th International Modelica Conference September 20-24, 2021, Linköping, Sweden

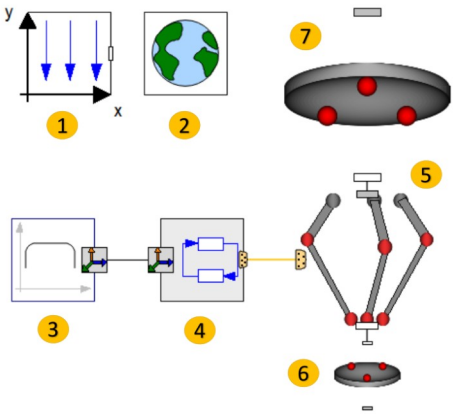
The Delta Robot

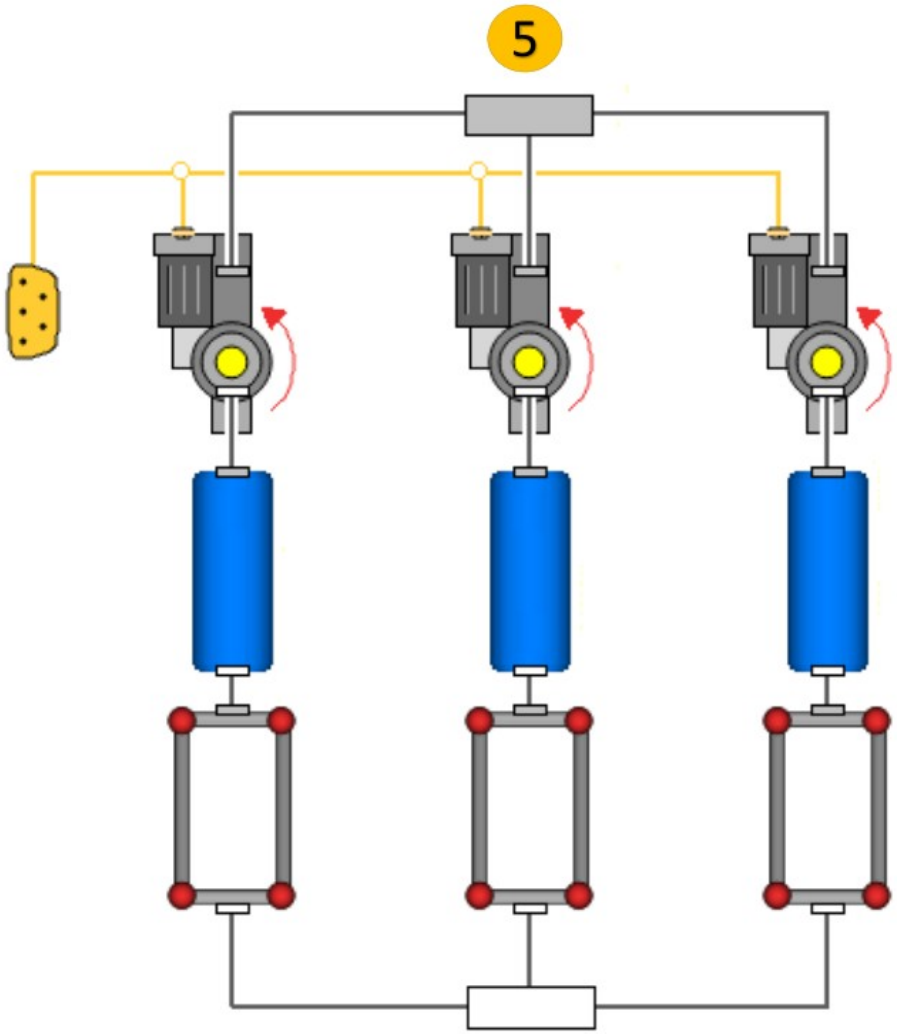
The Model - I


Spherical joints
Fixed platform
Upper arm
Actuated joint
Parallelogram
Moving platform
Spherical joints

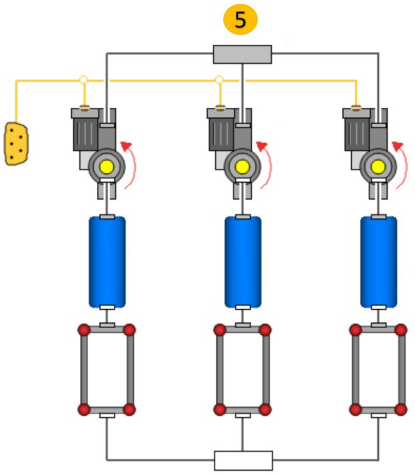

y
x
1
2
7
5
3
4
6

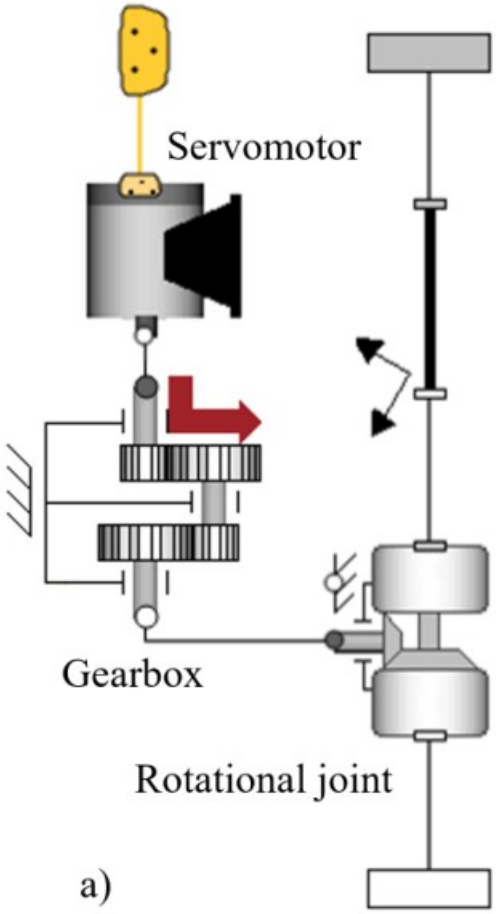
The Model - II

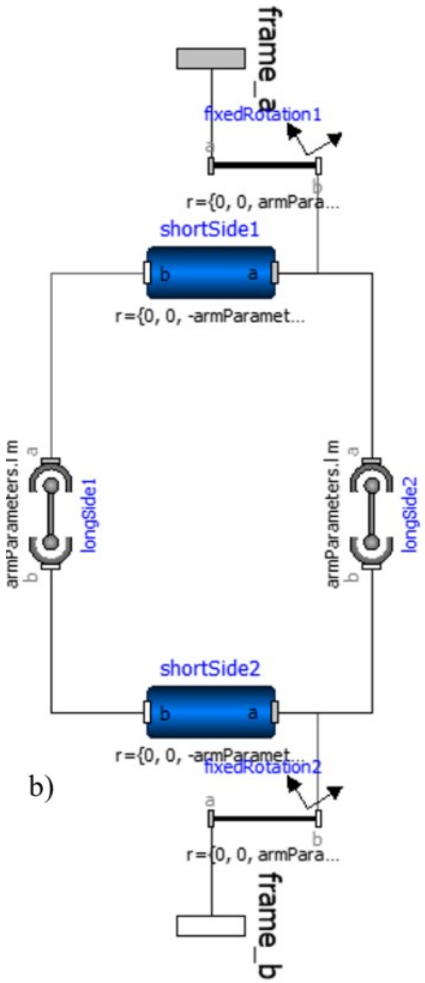




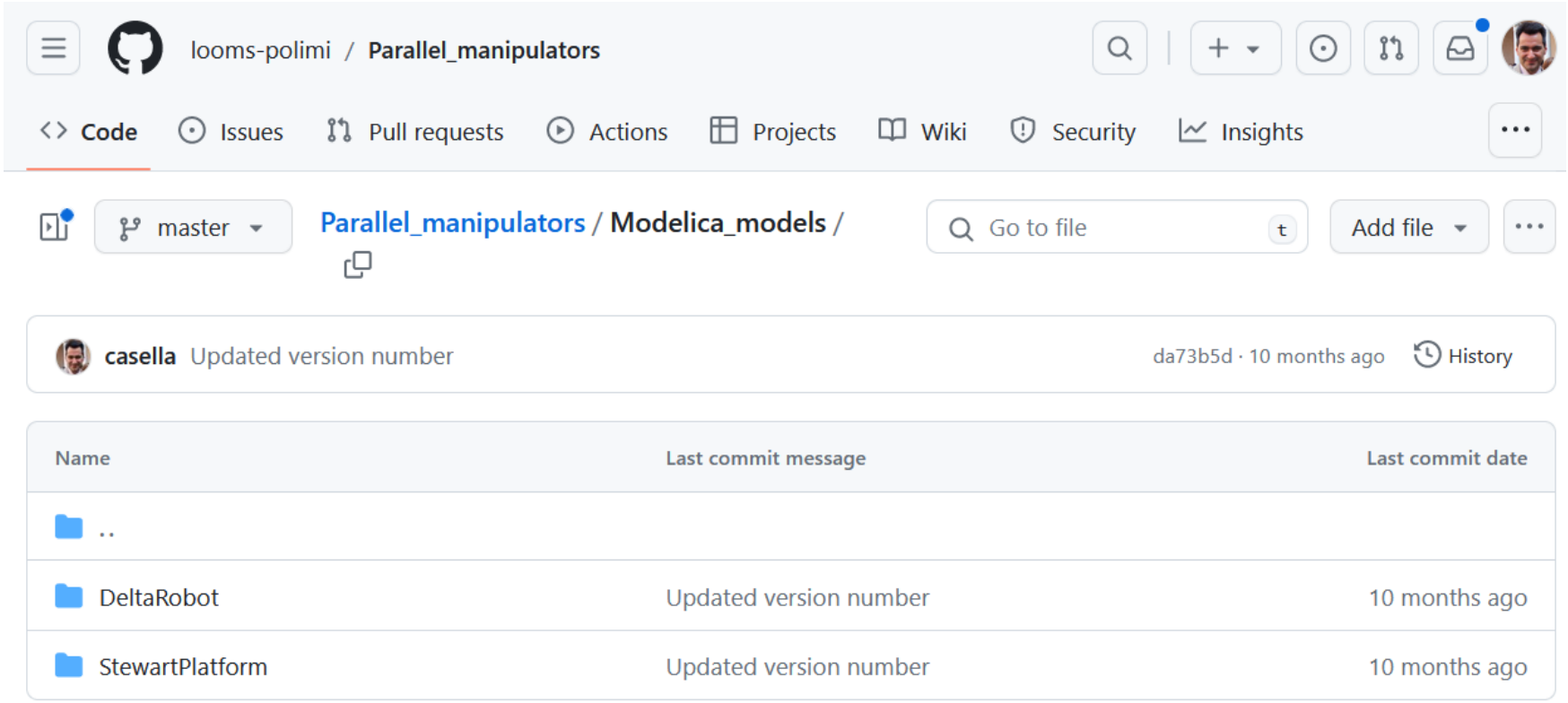
The Model - III







Open Source Code on GitHub



Highlights

- Mostly based on open-source Modelica MultiBody library
- Closed kinematic chains galore (→ not trivial)

Highlights

- Mostly based on open-source Modelica MultiBody library
- Closed kinematic chains galore (→ not trivial)
- `StateSelect.always` on position and velocity of linear actuators
- 5346 equations, 12 state variables

Highlights

- Mostly based on open-source Modelica MultiBody library
- Closed kinematic chains galore (→ not trivial)
- `StateSelect.always` on position and velocity of linear actuators
- 5346 equations, 12 state variables
- Worked out of the box in OMC/OMEdit

Highlights

- Mostly based on open-source Modelica MultiBody library
- Closed kinematic chains galore (→ not trivial)
- `StateSelect.always` on position and velocity of linear actuators
- 5346 equations, 12 state variables
- Worked out of the box in OMC/OMEdit
- Fully open source code, described in the Modelica Conference Paper

Highlights

- Mostly based on open-source Modelica MultiBody library
- Closed kinematic chains galore (→ not trivial)
- `StateSelect.always` on position and velocity of linear actuators
- 5346 equations, 12 state variables
- Worked out of the box in OMC/OMEdit
- Fully open source code, described in the Modelica Conference Paper
- Tested daily in the OSMC CI

Performance

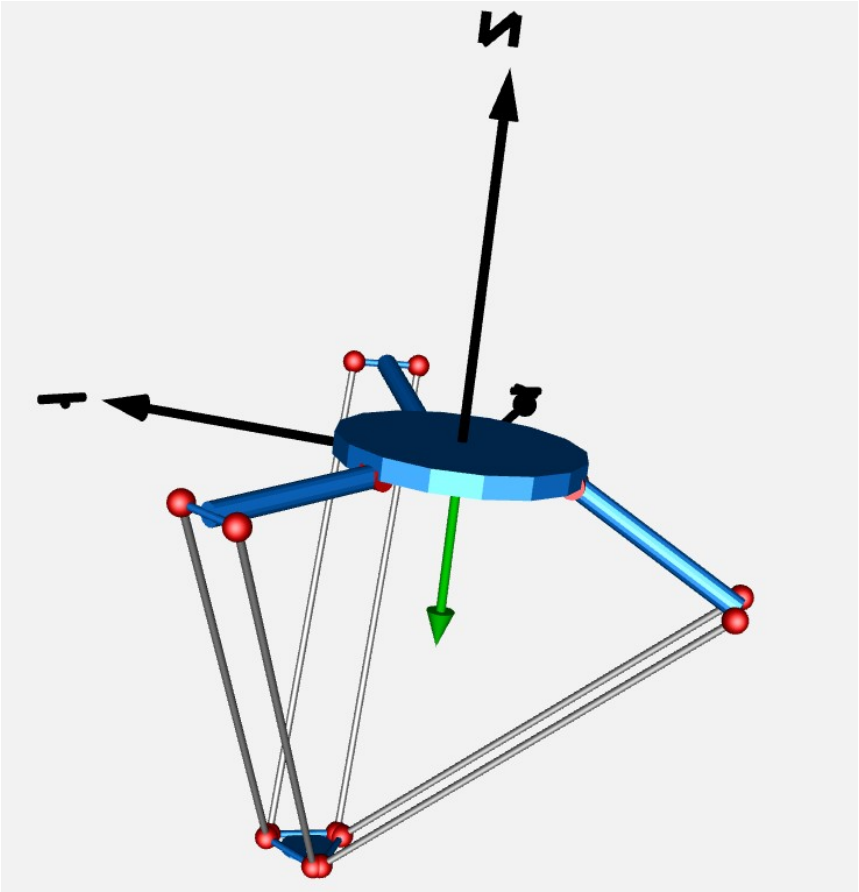
-

DeltaRobot test using OpenModelica

Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
9	9	9	9	9	9	9	0

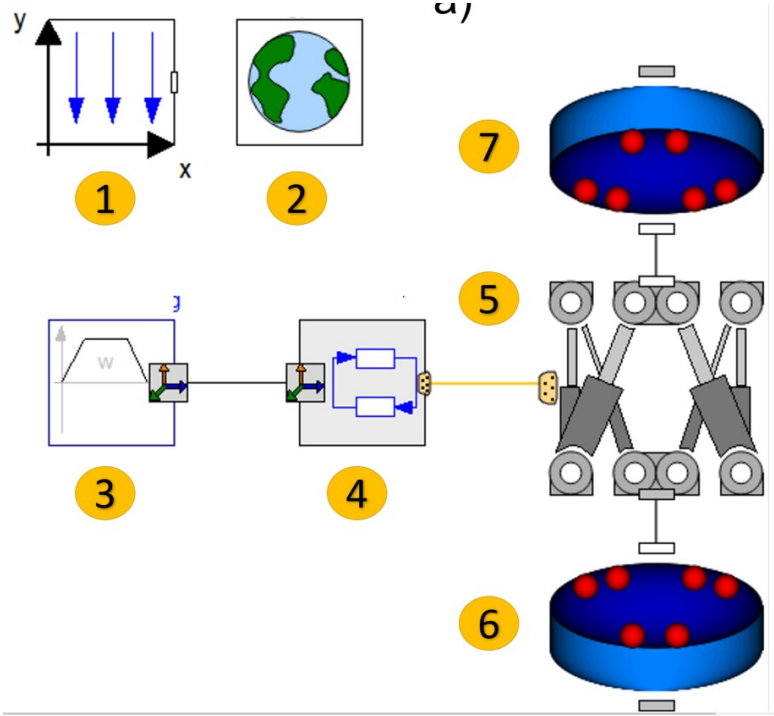
Model	Verified	Simulate	Total build	Model Parsing	Frontend	Backend	SimCode	Templates	Compile
DeltaRobot.Scenarios.DeltaRobot_IdealActPos_PositionValidation (sim)		1.65	59.11	1.80	1.22	31.16	2.22	3.36	21.14
DeltaRobot.Scenarios.DeltaRobot_IdealActPos_Scenario1 (sim)		0.16	40.02	1.79	1.22	22.80	1.44	3.47	11.08
DeltaRobot.Scenarios.DeltaRobot_IdealActVel_Scenario1 (sim)		0.20	38.99	1.76	1.18	23.42	0.96	3.15	10.29
DeltaRobot.Scenarios.DeltaRobot_IdealActVel_VelocityValidation (sim)		1.97	34.26	1.83	1.14	19.78	1.30	2.25	9.79
DeltaRobot.Tests.IdealActuatorPos_test (sim)		0.03	3.17	1.86	0.33	0.17	0.04	0.25	2.39
DeltaRobot.Tests.IdealActuatorVel_test (sim)		0.03	2.88	1.79	0.33	0.15	0.02	0.05	2.33
DeltaRobot.Tests.ServoMotor_test (sim)		0.01	2.01	1.74	0.11	0.01	0.00	0.18	1.71
DeltaRobot.Tests.Switch2_test (sim)		0.02	2.46	1.74	0.11	0.22	0.01	0.04	2.08
DeltaRobot.Tests.Trajectory_test (sim)		0.02	3.52	1.72	0.31	0.30	0.07	0.05	2.80

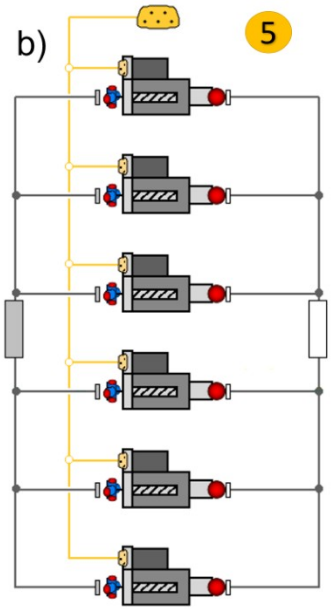
Demo



The Stewart Platform

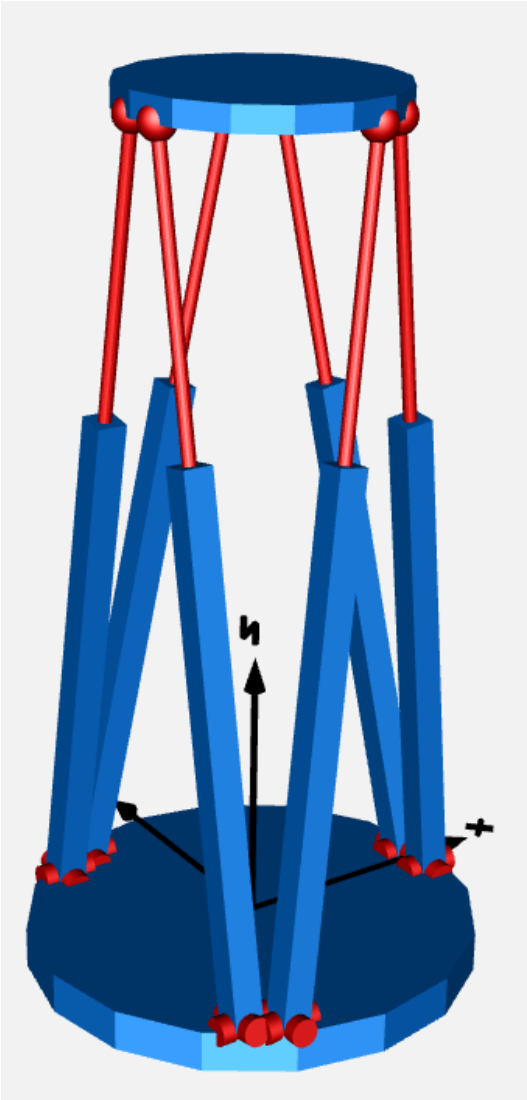
The Stewart Platform





9389 equations
12 state variables

Demo



Outlook

- Open-Source OS (if you use Linux)
- Open-Source Modelica Tool: OMC/OMEdit
- Open-Source Base Library: MSL
- Open-Source Model Framework

Outlook

- Open-Source OS (if you use Linux)
- Open-Source Modelica Tool: OMC/OMEdit
- Open-Source Base Library: MSL
- Open-Source Model Framework

Imagine the potential...

Case Study #2:

A Fuel-Cell Model for Oxy-Combustion Cycles

Credits

2023 IEEE Conference on Control Technology and Applications (CCTA)
August 16-18, 2023. Bridgetown, Barbados

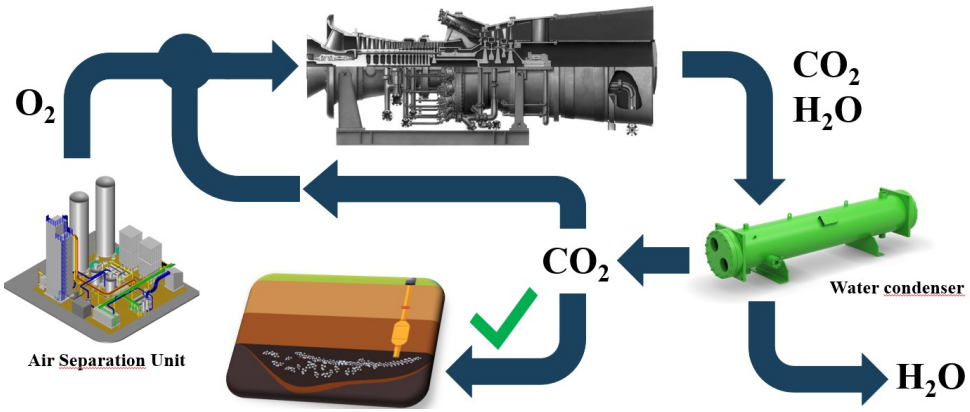
A Control-Oriented Modelica 1-D Model of a Planar Solid-Oxide Fuel Cell for Oxy-Combustion Cycles

Matteo Luigi De Pascali[1], Alessandro Donazzi[2], Emanuele Martelli[2] and Francesco Casella[1]

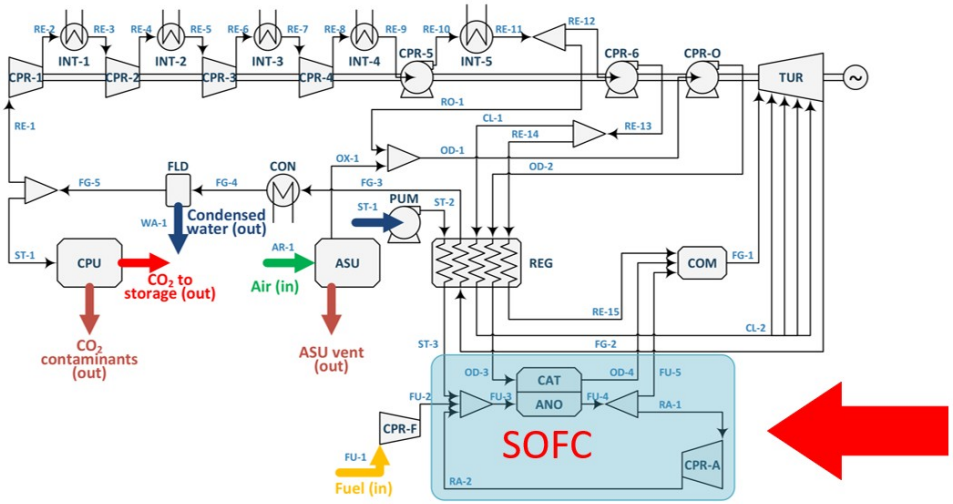
***Abstract*— The authors propose an object-oriented Modelica 1-D SOFC model following the indications of a previous work. The model is intended to be used in the study of the SOS-CO_2 cycle for control studies purposes. The SOFC working conditions are challenging and require adapting the model to handle properly reformate mixtures at the anode and carbon dioxide-rich mixtures at the cathode while keeping its formulation as simple as possible. This paper reviews the previous contribution, adapts the model for the new working conditions, provides insights related to the Modelica implementation, reproduces the tests presented in literature and extends the validation through a comparison with experimental data. Finally the model is tested under SOS-CO_2 cycle operative conditions.**

Context

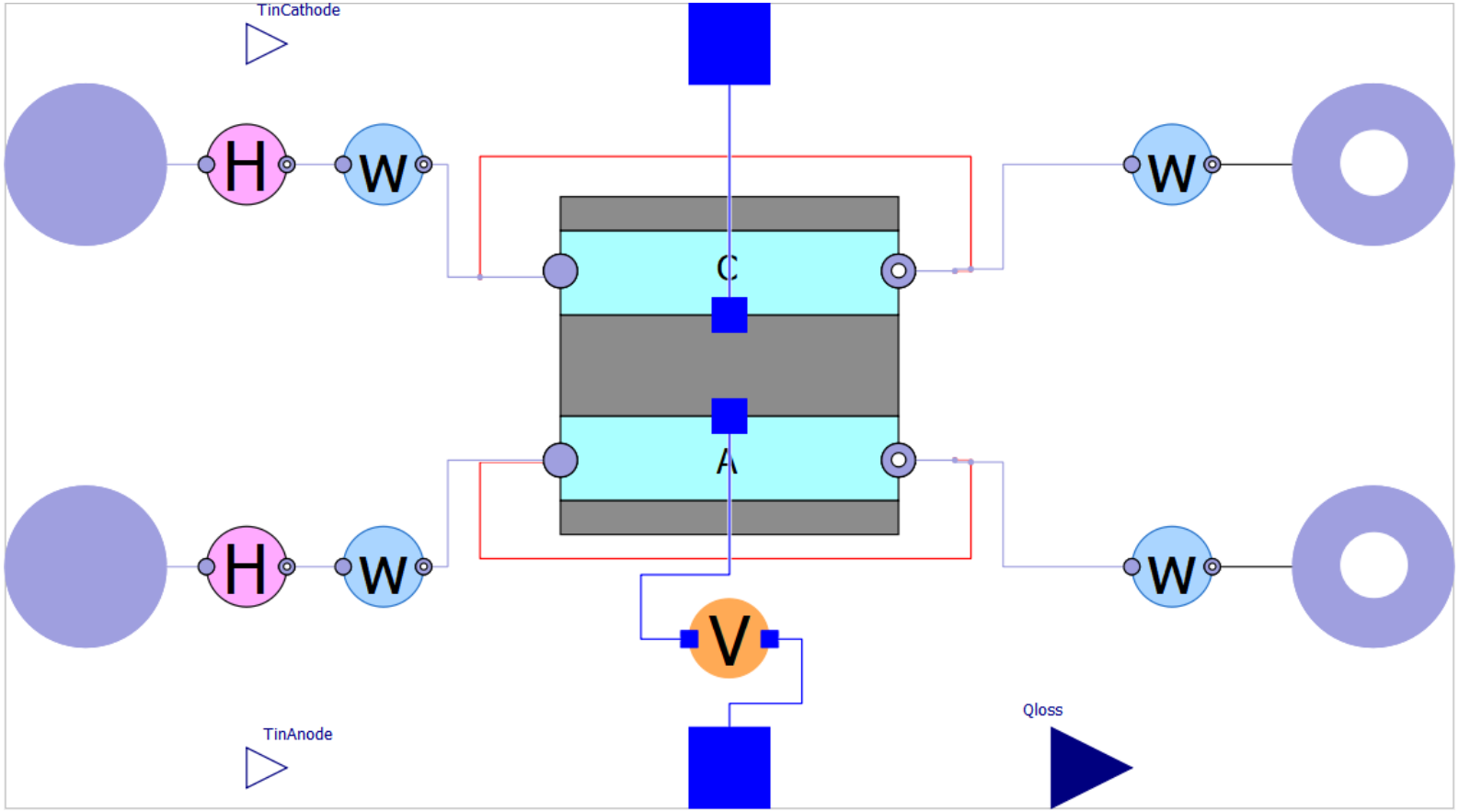
Oxy-Combusion Thermal Power Plant Concept



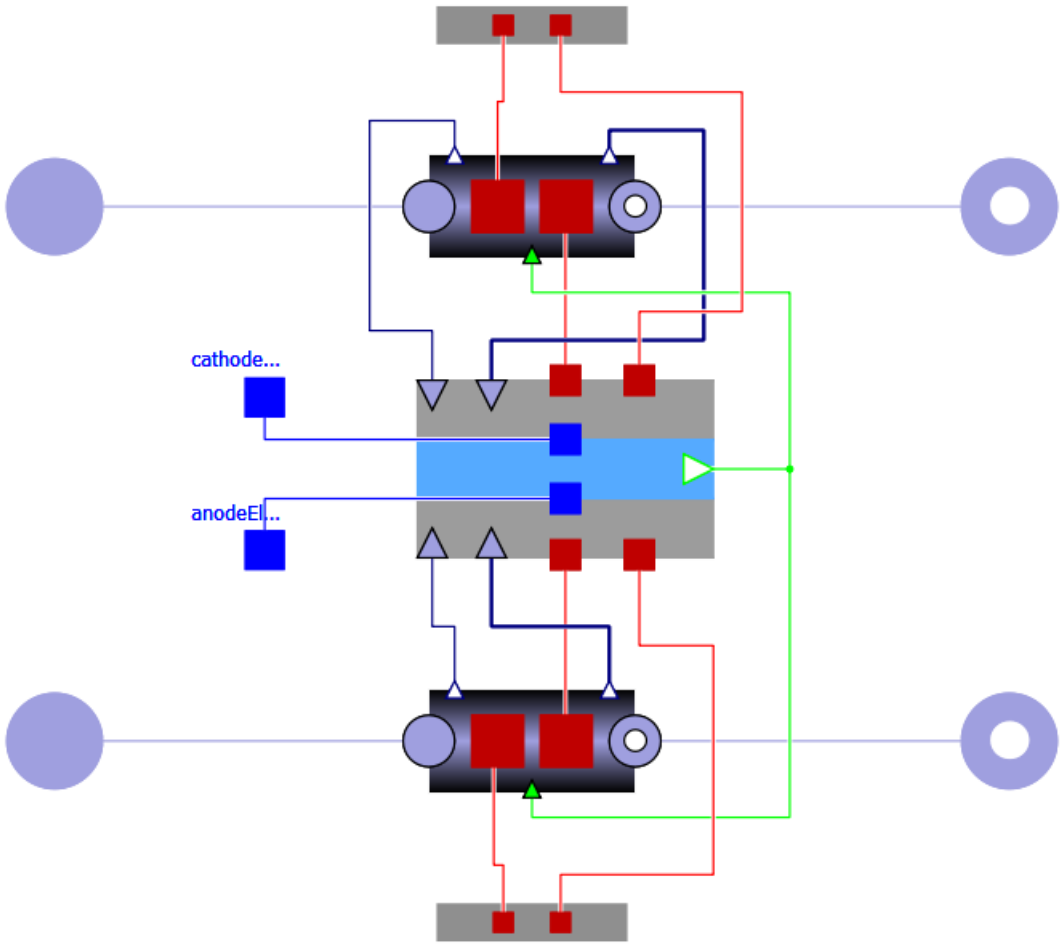
The SOS-CO_2 power plant



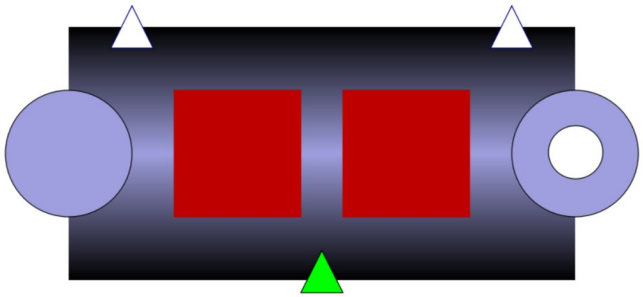
The Stack Model

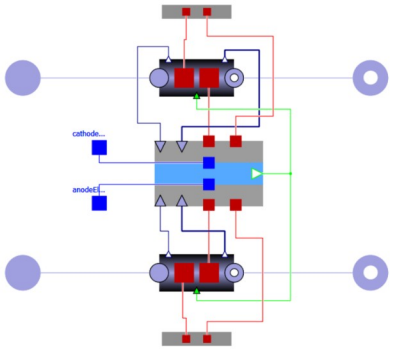


The Single Module Model



The 1D Channel Model



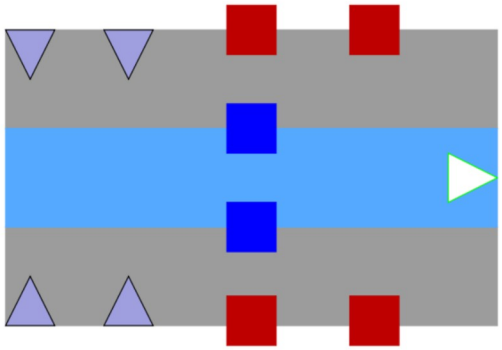


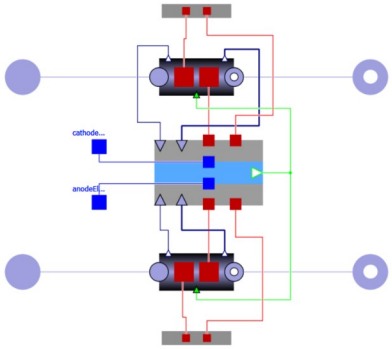
$$\frac{\mathrm{d}M}{\mathrm{d}t} = -V\rho^2 \left(\frac{\partial v_o}{\partial T_o} \frac{\mathrm{d}T_o}{\mathrm{d}t} + \frac{\partial v_o}{\partial p_o} \frac{\mathrm{d}p_o}{\mathrm{d}t} + \frac{\partial v_o}{\partial \bar{X}_o} \frac{\mathrm{d}\bar{X}_o}{\mathrm{d}t} \right)$$

$$M \frac{\mathrm{d}X_{j,o}}{\mathrm{d}t} = w_i \left(X_{j,i} - X_{j,o} \right) + S \cdot \mathrm{MM}_j \sum_{r=1,3} a_{j,r} r_r - X_{j,o} w_{O_2} + w_{j,O2}$$

$$M \left(\frac{\partial u_o}{\partial T_o} \frac{\mathrm{d}T_o}{\mathrm{d}t} + \frac{\partial u_o}{\partial p_o} \frac{\mathrm{d}p_o}{\mathrm{d}t} + \frac{\partial u_o}{\partial \bar{X}_o} \frac{\mathrm{d}\bar{X}_o}{\mathrm{d}t} \right) = w_i \cdot h_i - w_o \cdot h_o + Q_{PEN} + Q_{Plate}$$

The Proton Exchange Membrane Model





$$V_{OCP} = -\frac{\Delta g_{0,hor}}{nF} - \frac{RT}{nF} \cdot \ln\left(\frac{p_{H_2O}}{p_{H2} \cdot \sqrt{\frac{p_{O2}}{p_{ref}}}}\right)$$

$$v_{act}^{el} = \frac{RT_{PEN}}{\beta nF} \ln\left(\frac{j}{2j_0^{el}} + \sqrt{\left(\frac{j}{2j_0^{el}}\right)^2 + 1}\right)$$

$$j_0^{el} = \frac{RT_{PEN}}{nF} k_{el} \cdot \exp\left(-\frac{E_{el}}{RT_{PEN}}\right)$$

$$r_{sr} = k_{0,sr} \cdot \exp\left(-\frac{E_{a,sr}}{RT}\right) p_{CH_4}\, p_{H_2O} \left(1 - \frac{k_{P,sr}}{k_{Eq,sr}}\right)$$

Highlights

- About 10,000 equations
- Strongly nonlinear equations:
  - Chemical equilibria ($CH_4 + 2H_20 \rightarrow CO_2 + 4H_2$),
  - Arrhenius-type exponential laws

Highlights

- About 10,000 equations
- Strongly nonlinear equations:
  - Chemical equilibria ($CH_4 + 2H_20 \rightarrow CO_2 + 4H_2$),
  - Arrhenius-type exponential laws
- Steady state initializazion is extremely difficult: no homotopy, no party

Highlights

- About 10,000 equations
- Strongly nonlinear equations:
  - Chemical equilibria ($CH_4 + 2H_20 \rightarrow CO_2 + 4H_2$),
  - Arrhenius-type exponential laws
- Steady state initializazion is extremely difficult: no homotopy, no party
- Homotopy-based strategy leads to sequential computations at $\lambda = 0$
  - Prescribed current
  - Simplified linear polarization curve
  - Fixed temperature in Arrhenius terms

Highlights

- About 10,000 equations
- Strongly nonlinear equations:
  - Chemical equilibria ($CH_4 + 2H_20 \rightarrow CO_2 + 4H_2$),
  - Arrhenius-type exponential laws
- Steady state initializazion is extremely difficult: no homotopy, no party
- Homotopy-based strategy leads to sequential computations at $\lambda = 0$
  - Prescribed current
  - Simplified linear polarization curve
  - Fixed temperature in Arrhenius terms
- Model initially developed with Dymola
- We wanted to make it fully open-source for publication → run in OMC

Adapting the Model to OMC

- Steady-state initialization succeeded in Dymola but not in OMC
- Two issues:
  - Insufficient symbolic manipulation for the required $\lambda = 0$ simplifications $\rightarrow$ sequential computations
  - Bad tearing

Insufficient Symbolic Manipulation

```
dM_dt = wIn - wOut + wO2 "Overall mass balance";
```

```
dM_dt = -V*rhoOut^2*(fluidOut.dv_dT*der(Tout) +
        fluidOut.dv_dp*der(pOut) +
        fluidOut.dv_dX*der(Xout));
```

```
initial equation
  if initType == InitType.steadyState then
    if initialEquation then
      der(pOut) = 0;
    end if;
    der(Tout) = 0;
    der(Xout) = zeros(nX);
```

Insufficient Symbolic Manipulation

```
dM_dt = wIn - wOut + wO2 "Overall mass balance";


if initial() and initType == InitType.steadyState then
  dM_dt = 0;
else
  dM_dt = -V*rhoOut^2*(fluidOut.dv_dT*der(Tout) +
          fluidOut.dv_dp*der(pOut) +
          fluidOut.dv_dX*der(Xout));
end if;


initial equation
  if initType == InitType.steadyState then
    if initialEquation then
      der(pOut) = 0;
    end if;
    der(Tout) = 0;
    der(Xout) = zeros(nX);
```

Insufficient Symbolic Manipulation

```
dM_dt = wIn - wOut + wO2 "Overall mass balance";


if isOMC then
  if initial() and initType == InitType.steadyState then
    dM_dt = 0;
  else
    dM_dt = -V*rhoOut^2*(fluidOut.dv_dT*der(Tout) +
            fluidOut.dv_dp*der(pOut) +
            fluidOut.dv_dX*der(Xout));
  end if;
else
  dM_dt = -V*rhoOut^2*(fluidOut.dv_dT*der(Tout) +
          fluidOut.dv_dp*der(pOut) +
          fluidOut.dv_dX*der(Xout));
end if;


initial equation
  if initType == InitType.steadyState then
    if initialEquation then
      der(pOut) = 0;
    end if;
    der(Tout) = 0;
    der(Xout) = zeros(nX);
```

Bad Tearing

- Dymola's heuristics automatically selects the best tearing variables for the core nonlinear implicit initial equations: p, T, X, i

Bad Tearing

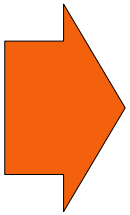
- Dymola's heuristics automatically selects the best tearing variables for the core nonlinear implicit initial equations: p, T, X, i
- Unfortunately, OMC's heuristics select other variables that lead to convergence failure

Bad Tearing

- Dymola's heuristics automatically selects the best tearing variables for the core nonlinear implicit initial equations: p, T, X, i
- Unfortunately, OMC's heuristics select other variables that lead to convergence failure
- Adding `__OpenModelica_tearingSelect=always` to p, T, X, i leads to successful convergence also in OMC

Bad Tearing

- Dymola's heuristics automatically selects the best tearing variables for the core nonlinear implicit initial equations: p, T, X, i
- Unfortunately, OMC's heuristics select other variables that lead to convergence failure
- Adding `__OpenModelica_tearingSelect=always` to p, T, X, i leads to successful convergence also in OMC



The Tearing Dilemma:

Rely on fantastic tearing algorithms

vs.

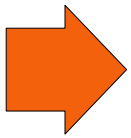
Rely on the modeller's experience?

Outlook

- After the fixes, the model runs successfully
  - Build time: about 1 min
  - Simulation time: about 10 s

Outlook

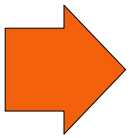
- After the fixes, the model runs successfully
  - Build time: about 1 min
  - Simulation time: about 10 s



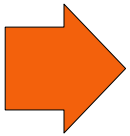
Adequate performance

Outlook

- After the fixes, the model runs successfully
  - Build time: about 1 min
  - Simulation time: about 10 s



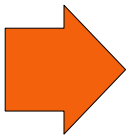
Adequate performance



Full publication possible (incl. source code)

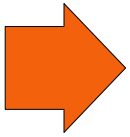
Outlook

- After the fixes, the model runs successfully
  - Build time: about 1 min
  - Simulation time: about 10 s



Adequate performance

Full publication possible (incl. source code)

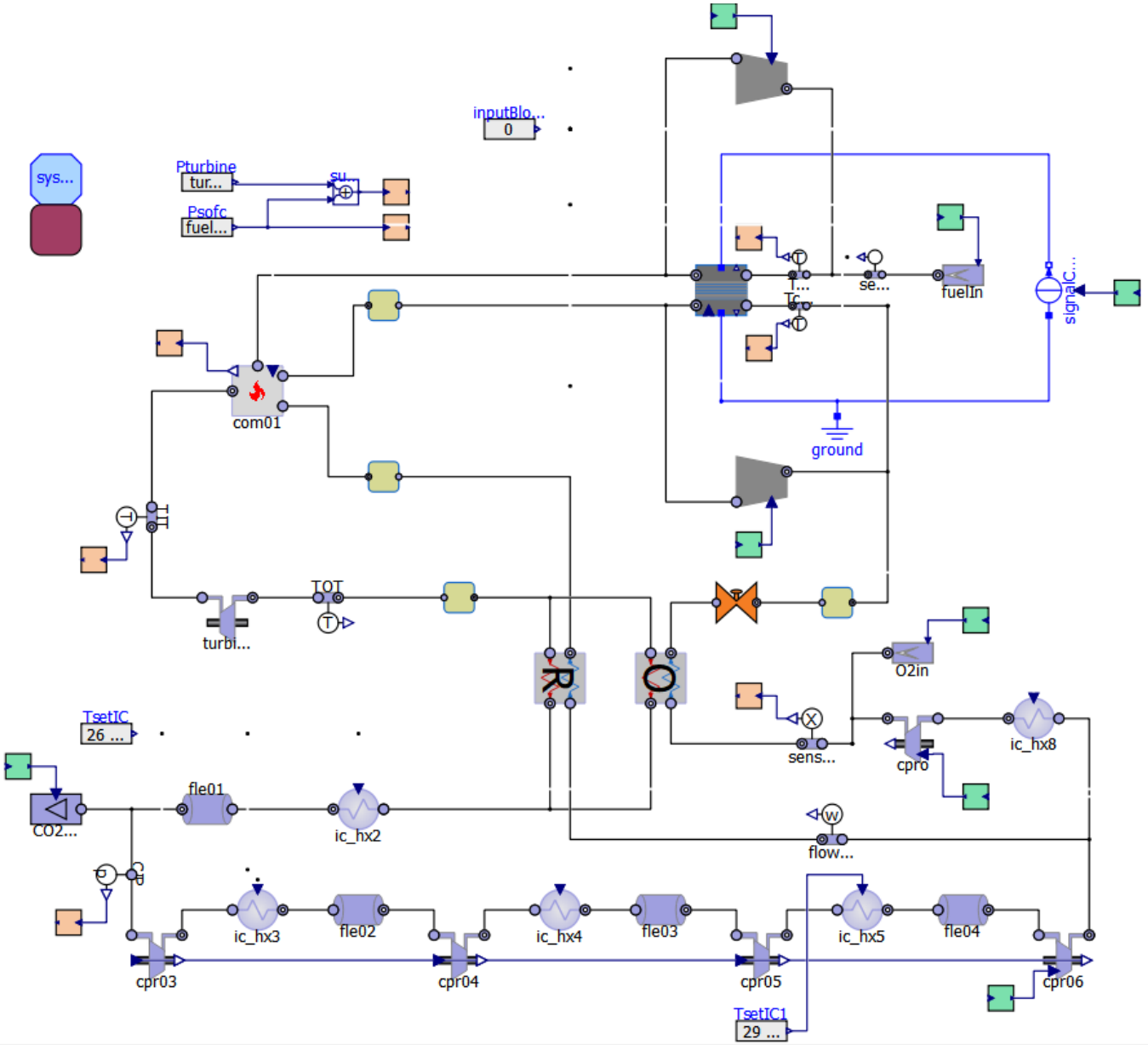


Fully open-source (incl. Modelica tool)

Case Study #3:

The SOS-CO2 Power Plant

The SOS-CO2 Plant Model



Building the Model

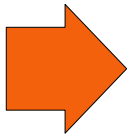
- Dymola requires 2-3 minutes to build the model on a fast i9 laptop

Building the Model

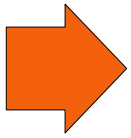
- Dymola requires 2-3 minutes to build the model on a fast i9 laptop
- OMC currently takes about 30 minutes
  - 10 min compiling C code
  - 8 min tearing
  - 2 min matching & sorting
  - 2 min removing simple equations

Building the Model

- Dymola requires 2-3 minutes to build the model on a fast i9 laptop
- OMC currently takes about 30 minutes
  - 10 min compiling C code
  - 8 min tearing
  - 2 min matching & sorting
  - 2 min removing simple equations



Usable, but not very practical yet



Further optimizations required in the backend and code generation phases

Initialization

- Extensive use of homotopy to make steady-state initialization possible (incl. decouplers at the inlet of each component)

Initialization

- Extensive use of homotopy to make steady-state initialization possible (incl. decouplers at the inlet of each component)
- Forward initialization in on-design conditions
  - Succeeds in Dymola
  - Succeeds in OMC if tearing is kept and tearingSelect is used

Initialization

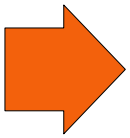
- Extensive use of homotopy to make steady-state initialization possible (incl. decouplers at the inlet of each component)
- Forward initialization in on-design conditions
  - Succeeds in Dymola
  - Succeeds in OMC if tearing is kept and tearingSelect is used
- Backward initialization in on-design conditions
  - Succeeds in Dymola
  - Fails in OMC

Initialization

- Extensive use of homotopy to make steady-state initialization possible (incl. decouplers at the inlet of each component)
- Forward initialization in on-design conditions
  - Succeeds in Dymola
  - Succeeds in OMC if tearing is kept and tearingSelect is used
- Backward initialization in on-design conditions
  - Succeeds in Dymola
  - Fails in OMC
- Backward initialization in off-design conditions
  - Works in Dymola only down to about 90% load
  - Fails in OMC

Initialization

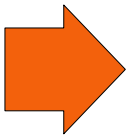
- Extensive use of homotopy to make steady-state initialization possible (incl. decouplers at the inlet of each component)
- Forward initialization in on-design conditions
  - Succeeds in Dymola
  - Succeeds in OMC if tearing is kept and tearingSelect is used
- Backward initialization in on-design conditions
  - Succeeds in Dymola
  - Fails in OMC
- Backward initialization in off-design conditions
  - Works in Dymola only down to about 90% load
  - Fails in OMC



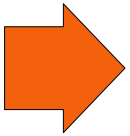
Numerically challenging problem!

Initialization

- Extensive use of homotopy to make steady-state initialization possible (incl. decouplers at the inlet of each component)
- Forward initialization in on-design conditions
  - Succeeds in Dymola
  - Succeeds in OMC if tearing is kept and tearingSelect is used
- Backward initialization in on-design conditions
  - Succeeds in Dymola
  - Fails in OMC
- Backward initialization in off-design conditions
  - Works in Dymola only down to about 90% load
  - Fails in OMC



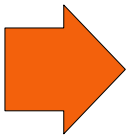
Numerically challenging problem!



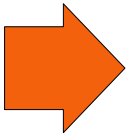
Dymola: dead end because of IP issues

Initialization

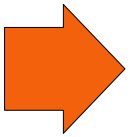
- Extensive use of homotopy to make steady-state initialization possible (incl. decouplers at the inlet of each component)
- Forward initialization in on-design conditions
  - Succeeds in Dymola
  - Succeeds in OMC if tearing is kept and tearingSelect is used
- Backward initialization in on-design conditions
  - Succeeds in Dymola
  - Fails in OMC
- Backward initialization in off-design conditions
  - Works in Dymola only down to about 90% load
  - Fails in OMC



Numerically challenging problem!



Dymola: dead end because of IP issues



OMC: we can still look under the hood

Simulation

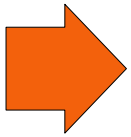
- Dymola simulates the system without fuss

Simulation

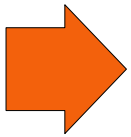
- Dymola simulates the system without fuss
- OpenModelica is extremely inefficient (about 100 s per time step) due to convergence issues on nonlinear systems involving derivatives (even at steady state!)

Simulation

- Dymola simulates the system without fuss
- OpenModelica is extremely inefficient (about 100 s per time step) due to convergence issues on nonlinear systems involving derivatives (even at steady state!)



Further investigation required



Most likely a scaling issue

Outlook

OMC is (almost) capable of handling the most challenging thermal power plant model of my entire career

Outlook

OMC is (almost) capable of handling the most challenging thermal power plant model of my entire career

With some further optimization, it could match or even surpass Dymola's performance (e.g. handling backward init to 20% load)

Case Study #4:

Power System Modelling with PowerGrids 2.0.0

Power Grids 1.0.0 (now 1.0.3)

Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library

Andrea Bartolini[1] Francesco Casella[2] Adrien Guironnet[3]

[1]Dynamica s.r.l., Italy, andrea.bartolini@dynamica-it.com
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy, francesco.casella@polimi.it
[3]RTE, France, adrien.guironnet@rte-france.com

Abstract

This paper presents the PowerGrids library, which is aimed at the modelling of large-scale power transmission and distribution system. The requirements and design principles of the library are extensively discussed, as well as some key implementation details. The library represents a prototype implementation of the presented requirements and design guidelines, and will form the basis for the future development of an industrial-grade open-source library to be used by European TSOs and DSOs.

Keywords: Power System Modelling, Power Generation and Transmission, Pan-European Power System

What's New?

- PowerGrids 2.0.0 new features
  - Significant refactoring of interfaces, based on experience with 1.0.x
  - Uses MSL 4.0.0

What's New?

- PowerGrids 2.0.0 new features
  - Significant refactoring of interfaces, based on experience with 1.0.x
  - Uses MSL 4.0.0
  - Embedded power flow

What's New?

- PowerGrids 2.0.0 new features
  - Significant refactoring of interfaces, based on experience with 1.0.x
  - Uses MSL 4.0.0
  - Embedded power flow
  - Vastly improved user interface
  - Benefits from recent advances in OMEdit
    - Parameter-dependent enabling of parameter input fields
    - DynamicSelect visualization of simulation results on diagrams

What's New?

- PowerGrids 2.0.0 new features
  - Significant refactoring of interfaces, based on experience with 1.0.x
  - Uses MSL 4.0.0
  - Embedded power flow
  - Vastly improved user interface
  - Benefits from recent advances in OMEdit
    - Parameter-dependent enabling of parameter input fields
    - DynamicSelect visualization of simulation results on diagrams
- Currently on https://github.com/PowerGrids/PowerGrids/tree/develop will be released soon

What's New?

- PowerGrids 2.0.0 new features
  - Significant refactoring of interfaces, based on experience with 1.0.x
  - Uses MSL 4.0.0
  - Embedded power flow
  - Vastly improved user interface
  - Benefits from recent advances in OMEdit
    - Parameter-dependent enabling of parameter input fields
    - DynamicSelect visualization of simulation results on diagrams
- Currently on https://github.com/PowerGrids/PowerGrids/tree/develop will be released soon
- New developments towards modelling of large-scale systems in OMC (> 1000 buses)

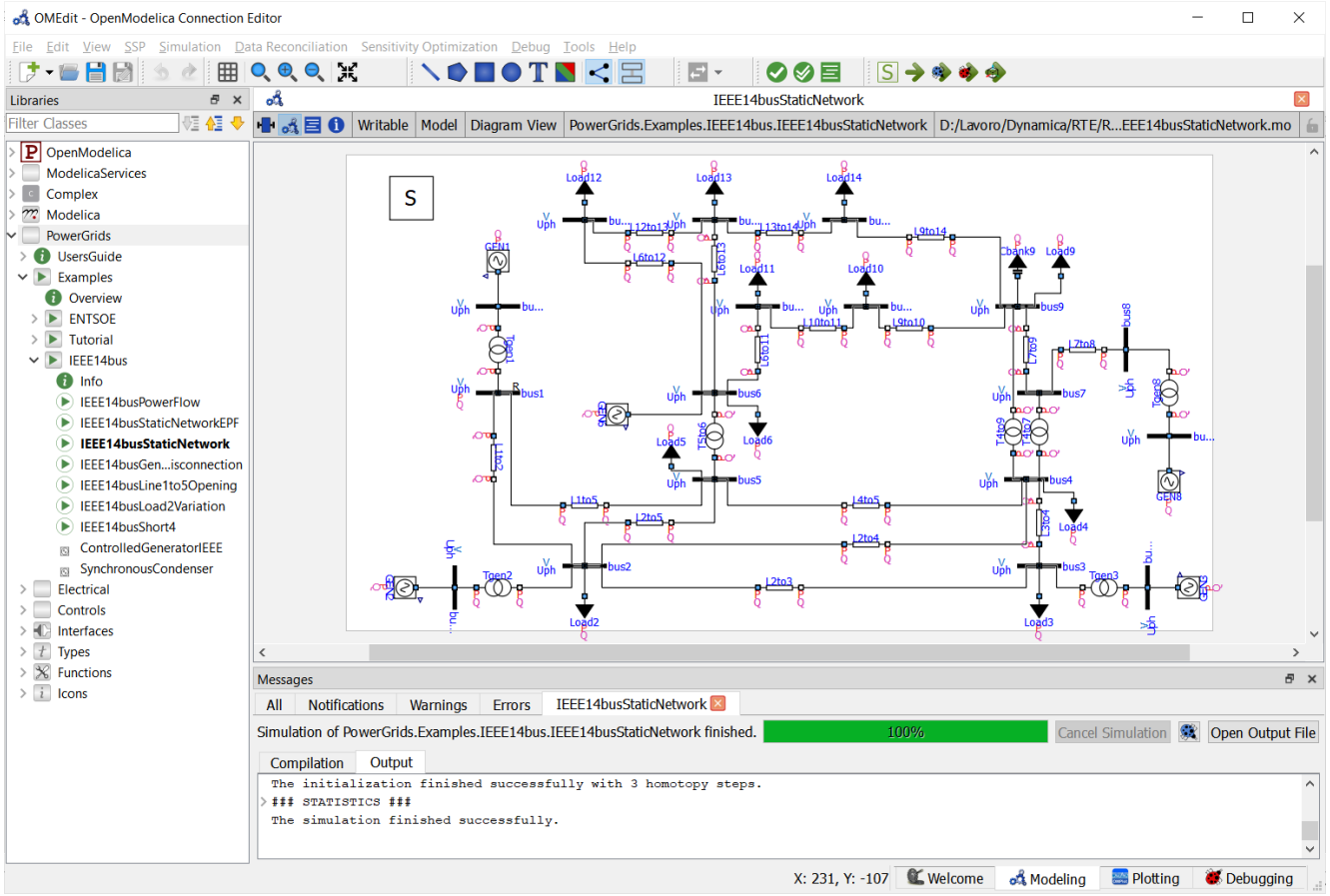
What's New?

- PowerGrids 2.0.0 new features
  - Significant refactoring of interfaces, based on experience with 1.0.x
  - Uses MSL 4.0.0
  - Embedded power flow
  - Vastly improved user interface
  - Benefits from recent advances in OMEdit
    - Parameter-dependent enabling of parameter input fields
    - DynamicSelect visualization of simulation results on diagrams
- Currently on https://github.com/PowerGrids/PowerGrids/tree/develop will be released soon
- New developments towards modelling of large-scale systems in OMC (> 1000 buses)

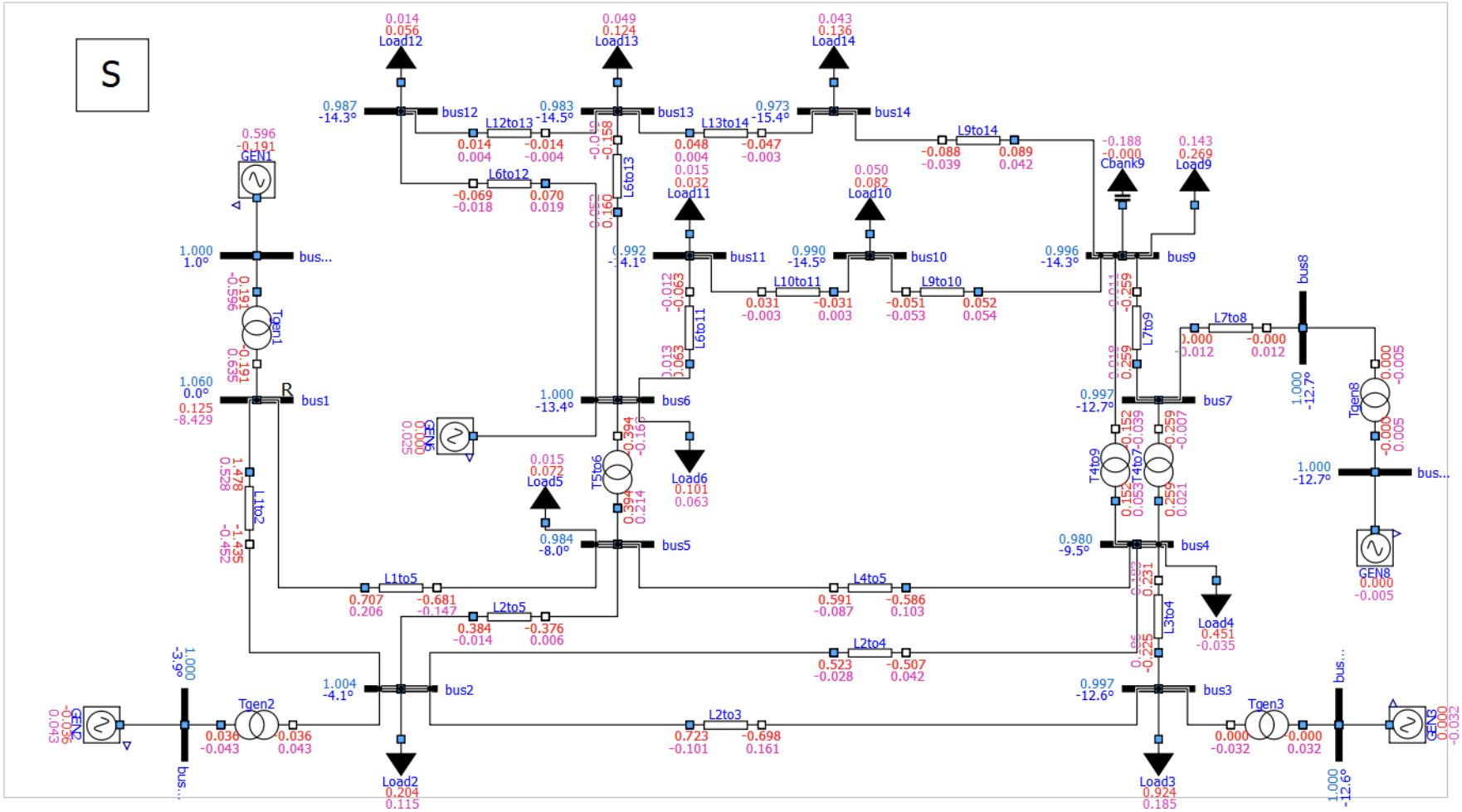
- Many thanks to:
  - **RTE** for the financial support to the development of OMC
  - Prof. **Massimo Ceraolo** (Univ. Pisa) for the financial support to the development of PowerGrids 2.0.0 and for the (relentless!) end-user feedback

PowerGrids 2.0.0 + OpenModelica

- A fully open-source integrated environment for power system simulation
  - Usable for teaching and research
  - Integrated power flow computation for initialization
  - GUI for building and editing models and for simulation results display
  - State of the art numerical solvers (IDA + Kinsol + KLU)



Using DynamicSelect Diagrams



Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done
  - Array-preserving pre- and post optimization: done

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done
  - Array-preserving pre- and post optimization: done
  - Scalarized causalization with array-based code generation: done

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done
  - Array-preserving pre- and post optimization: done
  - Scalarized causalization with array-based code generation: done
  - Support of Complex operator records: done

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done
  - Array-preserving pre- and post optimization: done
  - Scalarized causalization with array-based code generation: done
  - Support of Complex operator records: done
  - Support of daeMode: almost done

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done
  - Array-preserving pre- and post optimization: done
  - Scalarized causalization with array-based code generation: done
  - Support of Complex operator records: done
  - Support of daeMode: almost done
  - Support of sparse solvers for initialization: almost done

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done
  - Array-preserving pre- and post optimization: done
  - Scalarized causalization with array-based code generation: done
  - Support of Complex operator records: done
  - Support of daeMode: almost done
  - Support of sparse solvers for initialization: almost done
  - Index reduction: To-do (but not required for phasor-based models!)

Native Support for Large-Scale Models in OMC

- Enabling technology: array-preserving code generation
- New Frontend: done
  - Can flatten models without unrolling arrays
  - Can collect individual model instances into arrays
- New Backend: work in progress
  - Basic infrastructure: done
  - Array-preserving pre- and post optimization: done
  - Scalarized causalization with array-based code generation: done
  - Support of Complex operator records: done
  - Support of daeMode: almost done
  - Support of sparse solvers for initialization: almost done
  - Index reduction: To-do (but not required for phasor-based models!)
- Code generation: done
- Simulation runtime: done, using IDA+Kinsol+KLU

Current Test Results with New Backend

Model	Verified	Simulate	Total buildModel	Parsing	Frontend	Backend	SimCode	Templates	Compile
PowerGrids.Controls.Test.TestDeadBand (sim)	0.00 (1/8 failed)	0.01	1.61	1.62	0.00	0.00	0.00	0.01	1.59
PowerGrids.Controls.Test.TestDerivativeLag (sim)	0.01 (1/35 failed)	0.01	1.59	1.70	0.01	0.01	0.00	0.01	1.55
PowerGrids.Controls.Test.TestDiscontinuousDeadBand (sim)	0.00 (1/8 failed)	0.01	1.64	1.62	0.00	0.00	0.00	0.01	1.62
PowerGrids.Controls.Test.TestFirstOrderWithNonWindupLimiter		0.00	0.16	1.73	0.01	0.13	0.00	0.02	0.00
PowerGrids.Controls.Test.TestIntegratorWithNonWindupLimiter		0.00	0.04	1.68	0.01	0.03	0.00	0.00	0.00
PowerGrids.Controls.Test.TestLeadLag (sim)	0.01 (1/38 failed)	0.01	1.64	1.67	0.01	0.01	0.00	0.01	1.60
PowerGrids.Controls.Test.TestLeadLagWithNonWindupLimiter		0.00	0.16	1.70	0.01	0.13	0.00	0.02	0.00
PowerGrids.Controls.Test.TestLeadMOrderLag (sim)	1.03 (301/1346 failed)	0.18	2.65	1.67	0.01	0.18	0.02	0.05	2.39
PowerGrids.Controls.Test.TestLimiterWithLag		0.00	0.03	1.64	0.01	0.02	0.00	0.00	0.00
PowerGrids.Controls.Test.TestPIWithNonWindupLimiter		0.00	0.03	1.68	0.01	0.02	0.00	0.00	0.00
PowerGrids.Controls.Test.TestRampTrackingFilter		0.00	0.00	1.72	0.00	0.00	0.00	0.00	0.00
PowerGrids.Electrical.Test.LoadImpedancePQ (sim)	0.03 (228 verified)	0.08	2.47	1.52	0.03	0.17	0.01	0.03	2.23
PowerGrids.Electrical.Test.OneBusImpedanceOneLoad (sim)	0.00 (82 verified)	0.02	2.04	1.58	0.02	0.02	0.00	0.14	1.86
PowerGrids.Electrical.Test.OneBusImpedanceOneVariableLoad (sim)	0.02 (160 verified)	0.05	2.13	1.58	0.02	0.15	0.01	0.02	1.94
PowerGrids.Electrical.Test.OneBusImpedanceVariableVoltageOneLoad (sim)	0.01 (89 verified)	0.02	1.94	1.61	0.02	0.02	0.01	0.14	1.76
PowerGrids.Electrical.Test.OneBusOneLoad (sim)	0.00 (82 verified)	0.02	2.09	1.61	0.02	0.01	0.00	0.15	1.90
PowerGrids.Electrical.Test.OneBusTransmissionLineOneLoad (sim)	0.00 (138 verified)	0.05	2.21	1.60	0.03	0.16	0.01	0.02	1.99
PowerGrids.Electrical.Test.OneBusTransmissionLineShuntOneLoad (sim)	0.00 (138 verified)	0.05	2.23	1.61	0.03	0.16	0.01	0.03	2.02
PowerGrids.Electrical.Test.SynchronousMachine4Windings (sim)	0.13 (5/148 failed)	0.29	2.26	1.60	0.03	0.15	0.01	0.02	2.05
PowerGrids.Electrical.Test.SynchronousMachine4WindingsAccurate (sim)	0.03 (5/148 failed)	0.10	2.19	1.62	0.02	0.16	0.01	0.02	1.97
PowerGrids.Electrical.Test.SynchronousMachine4WindingsExact (sim)	0.05 (15/148 failed)	0.09	2.21	1.58	0.02	0.15	0.01	0.02	2.01
PowerGrids.Electrical.Test.SynchronousMachine4WindingsNoLoad (sim)	0.48 (5/148 failed)	1.06	2.17	1.62	0.03	0.15	0.01	0.02	1.97
PowerGrids.Electrical.Test.SynchronousMachine4WindingsNoLoadAccurate (sim)	0.47 (5/148 failed)	1.35	2.18	1.62	0.02	0.14	0.01	0.02	1.99
PowerGrids.Electrical.Test.SynchronousMachine4WindingsNoLoadExact (sim)	0.48 (5/148 failed)	1.38	2.21	1.60	0.02	0.14	0.01	0.02	2.02
PowerGrids.Electrical.Test.SynchronousMachine4WindingsPowerSwing		0.00	2.16	1.53	0.03	0.15	0.01	0.02	1.94
PowerGrids.Electrical.Test.TestExciterRectifierRegulationCharacteristicIEEE (sim)		0.01	1.74	1.57	0.01	0.00	0.00	0.01	1.71
PowerGrids.Electrical.Test.TestExciterVoltageTransducerIEEE		0.00	0.03	1.53	0.01	0.02	0.00	0.00	0.00
PowerGrids.Electrical.Test.TestOneBusTransmissionLineWithBreakersOneLoad		0.00	0.19	1.61	0.03	0.16	0.00	0.00	0.00
PowerGrids.Electrical.Test.TestTapChangerLogicInterval		0.00	0.03	1.49	0.01	0.02	0.00	0.00	0.00
PowerGrids.Electrical.Test.TestTapChangerLogicMax		0.00	0.03	1.52	0.01	0.02	0.00	0.00	0.00

Next Steps

- Feb-Mar 2024: continuous-time models should be handled
- Tests will be carried out with ScalablePowerGrids on exemplary power transmission models with size up to 20,000 buses

Next Steps

- Feb-Mar 2024: continuous-time models should be handled
- Tests will be carried out with ScalablePowerGrids on exemplary power transmission models with size up to 20,000 buses
- 2024-2025:
  - Efficient handling of events

Next Steps

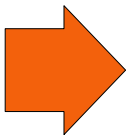
- Feb-Mar 2024: continuous-time models should be handled
- Tests will be carried out with ScalablePowerGrids on exemplary power transmission models with size up to 20,000 buses
- 2024-2025:
  - Efficient handling of events
  - Code optimizations
    (e.g. removing non-observable variables and equations)

Next Steps

- Feb-Mar 2024: continuous-time models should be handled
- Tests will be carried out with ScalablePowerGrids on exemplary power transmission models with size up to 20,000 buses
- 2024-2025:
  - Efficient handling of events
  - Code optimizations
    (e.g. removing non-observable variables and equations)
  - More efficient new backend algorithms for the scalarized parts
    (causalization, Jacobians)

Next Steps

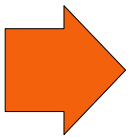
- Feb-Mar 2024: continuous-time models should be handled
- Tests will be carried out with ScalablePowerGrids on exemplary power transmission models with size up to 20,000 buses
- 2024-2025:
  - Efficient handling of events
  - Code optimizations
    (e.g. removing non-observable variables and equations)
  - More efficient new backend algorithms for the scalarized parts
    (causalization, Jacobians)



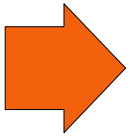
Goal: run national- and continental-size power system models out of the box in OMC

Next Steps

- Feb-Mar 2024: continuous-time models should be handled
- Tests will be carried out with ScalablePowerGrids on exemplary power transmission models with size up to 20,000 buses
- 2024-2025:
  - Efficient handling of events
  - Code optimizations
    (e.g. removing non-observable variables and equations)
  - More efficient new backend algorithms for the scalarized parts
    (causalization, Jacobians)



Goal: run national- and continental-size power system models out of the box in OMC



Support the challenges posed by the energy transition on the power systems in the EU and beyond

Conclusions

- OpenModelica is now capable of tackling challenging, state-of-the-art problems in energy, mechatronics, and other areas

Conclusions

- OpenModelica is now capable of tackling challenging, state-of-the-art problems in energy, mechatronics, and other areas

- OpenModelica enables fully open source solutions
  - for open scientific publications including the simulation code
  - for open collaboration (e.g. among ENTSO-E partners)

Conclusions

- OpenModelica is now capable of tackling challenging, state-of-the-art problems in energy, mechatronics, and other areas

- OpenModelica enables fully open source solutions
  - for open scientific publications including the simulation code
  - for open collaboration (e.g. among ENTSO-E partners)

- Shortcoming and issues can be handled together with end-users thanks to the open-source model

Conclusions

- OpenModelica is now capable of tackling challenging, state-of-the-art problems in energy, mechatronics, and other areas

- OpenModelica enables fully open source solutions
  - for open scientific publications including the simulation code
  - for open collaboration (e.g. among ENTSO-E partners)

- Shortcoming and issues can be handled together with end-users thanks to the open-source model

- Work in progress to further optimize and streamline performance and numerical robustness

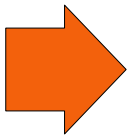
Conclusions

- OpenModelica is now capable of tackling challenging, state-of-the-art problems in energy, mechatronics, and other areas

- OpenModelica enables fully open source solutions
  - for open scientific publications including the simulation code
  - for open collaboration (e.g. among ENTSO-E partners)

- Shortcoming and issues can be handled together with end-users thanks to the open-source model

- Work in progress to further optimize and streamline performance and numerical robustness



Bright future ahead!

Thank you for your kind attention!