Dynaωo: A Suite of Power System Simulation Tools using Modelica and the Open Modelica Compiler

A. Guironnet et al., Virtual 2021 Open Modelica Workshop

http://www.dynawo.org
https://github.com/dynawo/dynawo
rte-dynawo@rte-france.com

Outline

- Introduction to power system simulations
- Introduction to Dynaωo
- Dynaωo and OpenModelica
- Conclusions

01

Introduction to power system simulations

Transmission System Operators

- "Entities operating independently from the other electricity market players and responsible for the bulk transmission of electric power on the main high voltage electric network"
  - ➢ Non discriminatory and transparent access to the electricity grid
  - ➢ Safe operation and maintenance of the system
  - ➢ Grid infrastructure development

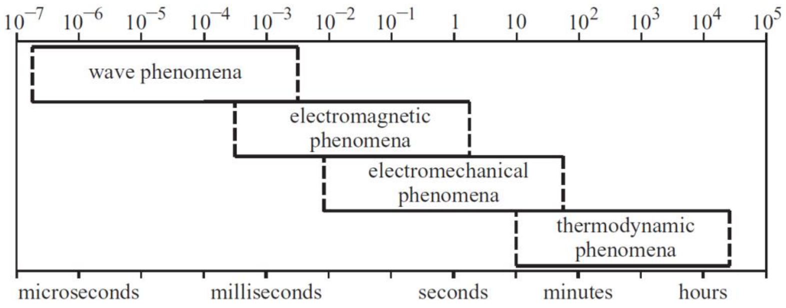
- RTE – French Transmission System Operator
  - ➢ In charge of the largest European network (more than 100 000 kms of EHV and HV lines – 400 to 63 kV, 2 600 substations, peak load served > 100 GW).
  - ➢ Ensuring a stable and secure operation means:
    - ❖ *Adequacy* – Acceptable steady-state (thermal overloads, voltage values for materials)
    - ❖ *Stability* – Stable and possible transition between two operating points

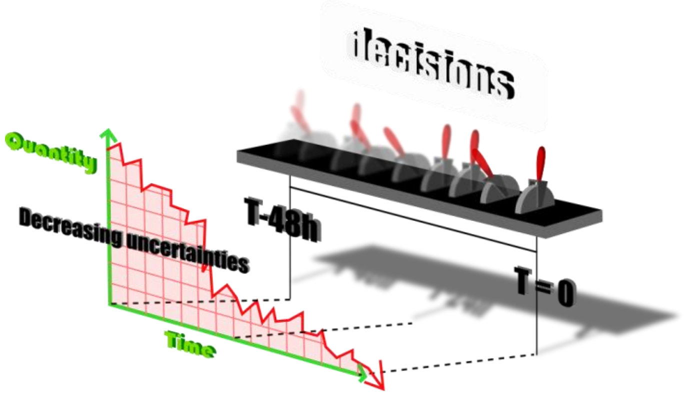
      Dynamic stability (transient, voltage, small-signal, frequency, etc.) ensured by time-domain simulations.

Power System Simulations

- Done at different time-scale on a regular basis to ensure adequacy and stability
  - Static and dynamic security assessment (simulating all network contingency every 15')
  - Day, week and month ahead assessment with static simulations (steady-state calculation, short-circuit calculation) as well as time-domain simulations (voltage or transient studies) on different scenarios.
  - Planning studies (from a few years to 20 years ahead studies).
  - Design ad'hoc stability studies (insertion of new components – HVDC links, offshore wind parks, etc.)

- Analysis of the system during transitions or at steady-state following a transition
  - Triggered by the normal evolution of the system (load change, production scheduled change, etc.) or by sudden change (generator tripping, short-circuit, etc.)
  - Involves a large range of phenomena with different time constants.





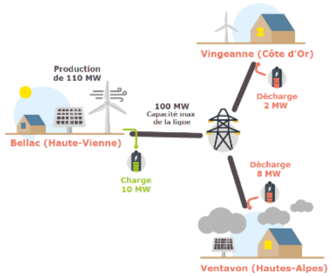
Challenges and needs

- A system evolving at a very high pace due to a global demand for cleaner energy
  - ➢ Massive integration of Renewable Energy Sources (RES).
  - ➢ High-Voltage Direct Current (HVDC) links boom
  - ➢ Deep evolution of the consumption uses (active consumers, electric vehicle, microgrids, etc.)

- A complete switch from an easy-to-predict and physically-driven system to a more complex, unpredictable and numerically-driven system
  - ➢ Forces System Operators to find efficient and complex ways to control it
  - ➢ Leads to the development of advanced special protection, control and regulation schemes deeply modifying the system behavior, its dynamics and its stability.

⇒All of this advocates for <u>more collaboration</u>, <u>more transparency</u>

and <u>more flexibility.</u>



Traditional approach and current situation

- Closed and proprietary power system simulation tools are the norm and use:
  - ➢ An internal representation of the system that can’t be easily shared
  - ➢ Programming language to develop and offer closed models
  - ➢ Closed numerical methods to solve the mathematical problems

- Most of them are based on legacy code developed for classical AC systems
  - ➢ Strong and very low level hypothesis on the system’s behavior
  - ➢ Constraints on the way elements can interact and can be connected
  - ➢ Introducing a new methodology or a new technology demands to modify a large part of the tool

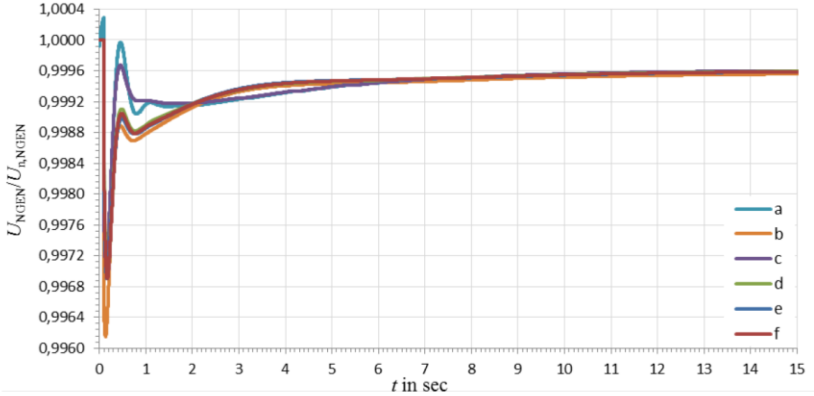
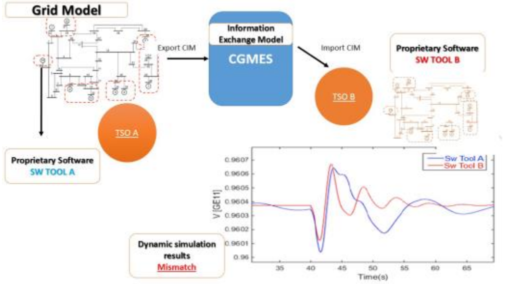


FIG. 5-3: RESPONSE OF TERMINAL VOLTAGE OF THE MACHINE IN TEST CASE 2



=> The traditional approach and the legacy simulation tools are at odds with the previously introduced needs with limited possibilities for collaboration, limited transparency and limited flexibility.

02

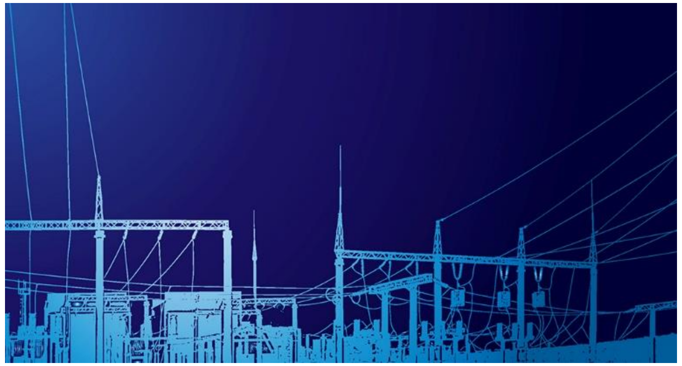
Introduction to Dynaωo

The Dynawo initiative

- A complete perspective change in terms of power system simulations tools approach
  - ➢ Withdrawal of legacy closed simulation tools development
  - ➢ Switch to a transparent and open-source approach
  - ➢ Switch to the use of a high-level modeling language
  - ➢ Switch to a strict separation between modeling and solving parts

⇒ <u>A very strong commitment by RTE for a new vision for power system simulation tools</u>

- ❖ To build, share and develop new solutions with all interested partners (TSO, DSO, RSC, universities, etc.)
- ❖ To set-up a new standard for simulations with easy exchanges, discussions and collaborations



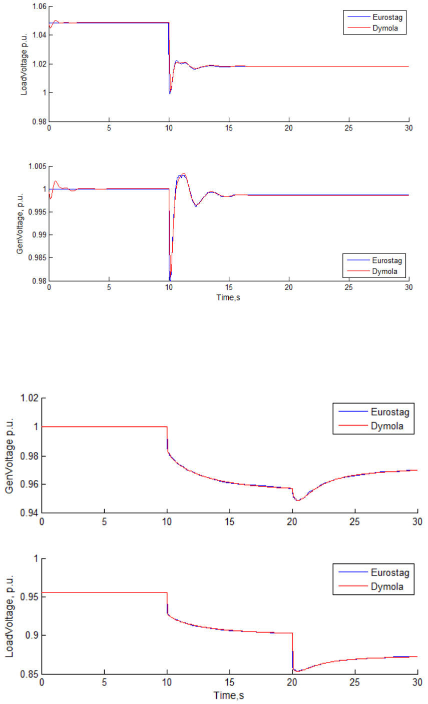
http://www.dynawo.org – rte-dynawo@rte-france.com

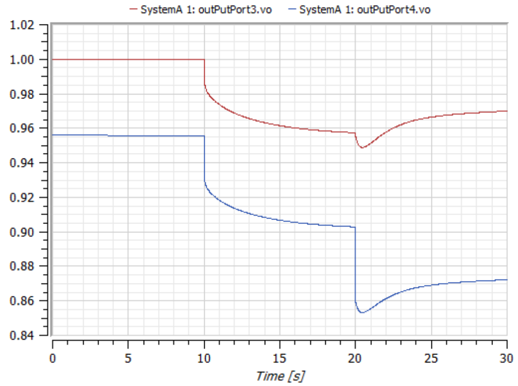
The exploratory phase

- Conducted through two R&D European projects to assess the feasibility of using the Modeling language for power system modeling
  - Pegase(1): first very simple models to get knowledge on the language and to understand its main features, pros and cons.
  - iTesla(2, 3,4): development and validation of a Modelica library (iPSL) that enables to obtain identical results than closed legacy tools (Eurostag, PSS/E, etc.)

PEGASE

iTesla
Innovative Tools for Electrical System Security within Large Areas





http://www.dynawo.org – rte-dynawo@rte-france.com

The exploratory phase

- Conducted through two R&D European projects to assess the feasibility of using the Modeling language for power system modeling
  - Pegase(1): first very simple models to get knowledge on the language and to understand its main features, pros and cons.
  - iTesla(2, 3,4): development and validation of a Modelica library (iPSL) that enables to obtain identical results than closed legacy tools (Eurostag, PSS/E, etc.)

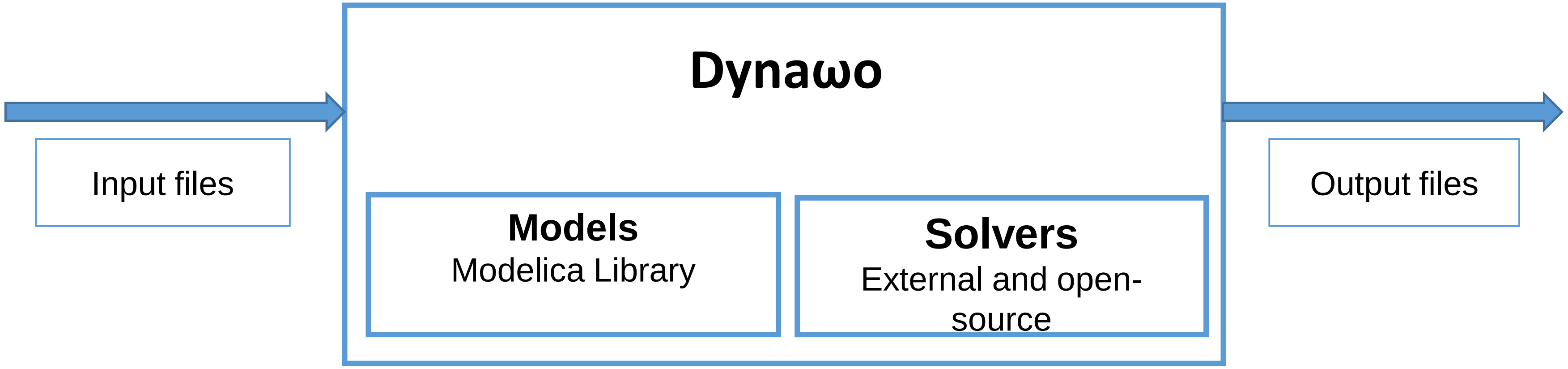
⇒ Both projects enable to prove that Modelica can be used to do power system modeling

⇒ The next steps have consisted in:

- Finding a way to bypass the language and tools limitations for an operational use
- Switching to a real declarative approach in modeling

The Dynaωo approach

- An hybrid C++/Modelica open-source suite of simulation tools(5) based on two core principles:
  - Using as much as possible a high-level modeling language (Modelica) for the modeling side
  - A strict separation between the modeling and solving parts


Input files

Dynaωo

Models
Modelica Library

Solvers
External and open-source

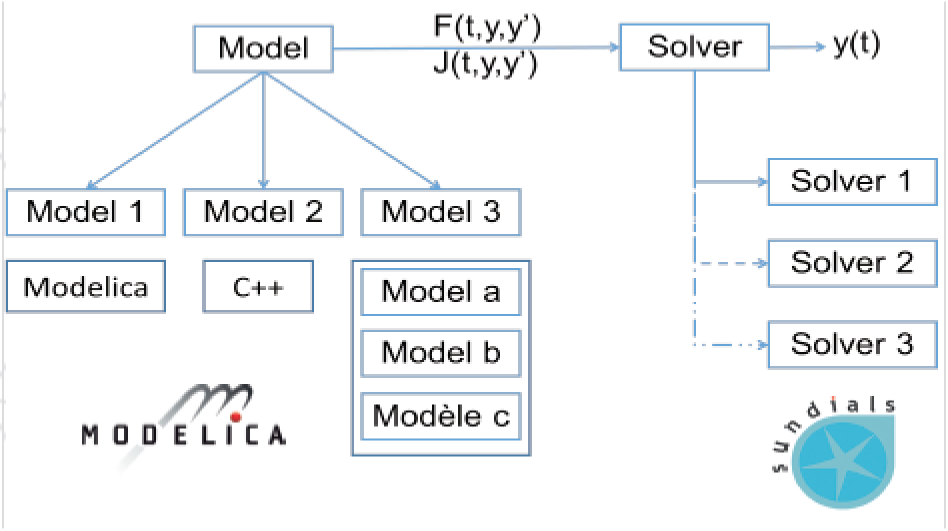
Output files

⇒ In order to ensure flexibility, transparency and quality without degrading the performances compared to classical power system simulation tools.

http://www.dynawo.org – rte-dynawo@rte-france.com

The Dynawo approach

- An hybrid C++/Modelica open-source suite of simulation tools(5) based on two core principles:
  - ➢ Using as much as possible a high-level modeling language (Modelica) for the modeling side
  - ➢ A strict separation between the modeling and solving parts



Model

F(t,y,y')

J(t,y,y')

Solver

y(t)

Model 1

Model 2

Model 3

Modelica

C++

Model a

Model b

Modèle c

Solver 1

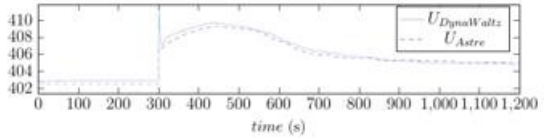
Solver 2

Solver 3

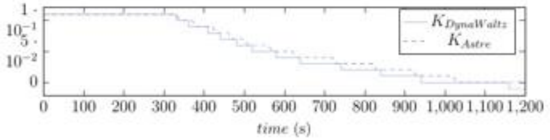
⇒ In order to ensure <u>flexibility, transparency and quality without degrading the performances</u> compared to classical power system simulation tools.

The experimentation phase

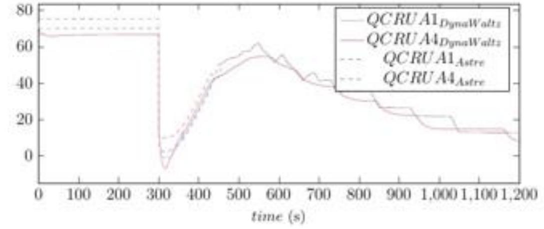
- Conducted through internal efforts and collaborations with power system and Modelica experts to validate and industrialize the Dynawo approach for long-term and transient stability studies.
  - ➢ <u>Quality</u> assessed by thorough validations against legacy closed tools (both for long-term and transient stability studies).
  - ➢ <u>Transparency(6)</u> guaranteed by a concrete switch to declarative modeling
  - ➢ <u>Performances similar(7)</u> than current simulation tools thanks to an unique approach

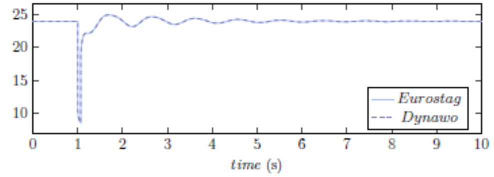


(a) Tension du poste TRLPP7 en kV

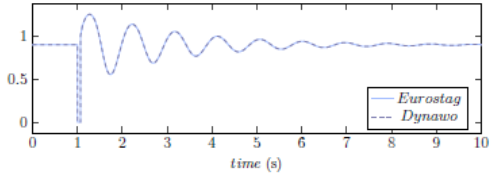


(b) Niveau RST de la zone TRLPP7

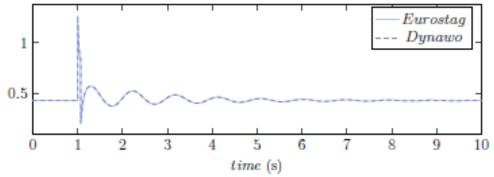




(a) Tension aux bornes du générateur (kV)

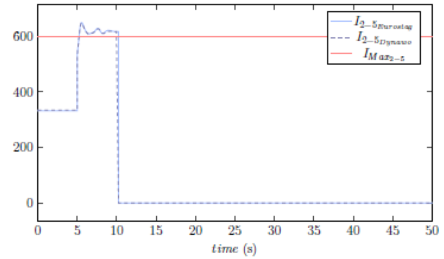


(b) Puissance active du générateur (p.u)

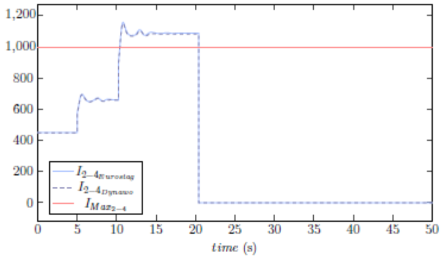


(c) Puissance réactive du générateur (p.u)

FIGURE 10 – Défaut triphasé



(a) Courant sur la ligne 2-5 (A)



(b) Courant sur la ligne 2-4 (A)

FIGURE 24 – Fonctionnement de l'automate de délestage ampèremétrique

The experimentation phase

- Conducted through internal efforts and collaborations with power system and Modelica experts to validate and industrialize the Dynaωo approach for long-term and transient stability studies.
  - ➢ Quality assessed by thorough validations against legacy closed tools (both for long-term and transient stability studies).
  - ➢ Transparency(6) guaranteed by a concrete switch to declarative modeling
  - ➢ Performances similar(7) than current simulation tools thanks to an unique approach

Network	Scenario	Variables nb	Solver	Sim. Time
IEEE14	Generator disconnection (gen2) at t= 100 s. tStop = 1200 s.	365	BE order1 h = 1 s	**0.300 s**
IEEE14	Non impedant fault – 1/1.1 s at gen2 tStop = 20 s	400	IDA BDF order2 1e-4 accuracy	**0.190 s**
IEEE57	Generator disconnection (gen 12) at t = 100 s. tStop = 1200s	467	BE order1 h = 1 s	**0.292 s**
IEEE57	Non impedant fault – 1/1.1 s at gen 12 tStop = 20 s	793	IDA BDF order2 1e-4 accuracy	**1.35 s**
French Regional Snapshot	Generator disconnection on a nuclear unit (t = 100 s). tStop = 1200 s	~ 35 000	BE order1 h = 1 s	**6.5 s**

The experimentation phase

- Conducted through internal efforts and collaborations with power system and Modelica experts to validate and industrialize the Dynawo approach for long-term and transient stability studies.
  - Quality assessed by thorough validations against legacy closed tools (both for long-term and transient stability studies).
  - Transparency(6) guaranteed by a concrete switch to declarative modeling
  - Performances similar(7) than current simulation tools thanks to an unique approach

⇒ The promising results obtained, the robustness of the approach and the potential it offers conduct to:

- A parallel run currently going on for voltage stability simulation tool with an operational use expected end of 2021 / beginning of 2022
- An extension of the project to renew a large part of RTE simulation tools using the Dynawo approach.

The extension phase

- An approach declined to build and propose a complete and consistent suite of simulation tools
  - *DynaFlow*(8) for calculating the correct steady-state by using a time-domain approach to properly take into account the interactions between controllers.
    - Proof-of-concept validated and industrialization under progress.
  - *DySym* for short-circuit calculations.
    - Proof-of-concept envisioned for the end of the year.
  - *DynaWaltz* for long-term stability studies
    - Parallel run ongoing and operational use expected next year.
  - *DynaSwing* for transient stability studies
    - Positive proof-of-concept and industrialization under progress.
  - *DynaWave*(9), a « quasi-EMT » approach, for stability studies and system design with a high penetration of Power Electronics.
    - First efforts done to envision a proof-of-concept next year.

The extension phase

<table>
<tr><th rowspan="3">Simulation tool</th><th>DynaFlow</th><th>DySym</th><th>DynaWaltz</th><th>DynaSwing</th><th>DynaWave</th></tr>
<tr><td>Steady-state simulation</td><td>Short-circuit calculation</td><td>Long-term stability</td><td>Transient stability</td><td>Fast dynamics calculation (quasi-EMT)</td></tr>
<tr><td colspan="5">Common high-level objects and APIs</td></tr>
<tr><th>Compiler</th><td colspan="5">One single compiler (OpenModelica Compiler)</td></tr>
<tr><th rowspan="2">Modelling choices
(Modelica based)</th><td colspan="5">Common models for « slow » dynamics objects</td></tr>
<tr><td>Simplified models</td><td>Simplified three-phase models</td><td colspan="2">Phasor models</td><td>« Quasi-EMT » models</td></tr>
<tr><th rowspan="2">Solver</th><td colspan="5">Common low-level numerical parts (LU decomposition, algebraic solvers)</td></tr>
<tr><td>Simplified solver</td><td>Specific DAE solver</td><td>Simplified solver</td><td>Specific DAE solver</td><td>Specific DAE solver</td></tr>
</table>

Mid-term to long-term objectives

- An approach declined to build and propose a complete and consistent suite of simulation tools
  - *DynaFlow* - Operational use foreseen in 2022/2023
  - *DySym* - Operational use foreseen in 2023/2024
  - *DynaWaltz* - Operational use foreseen in 2021/2022
  - *DynaSwing* - Operational use foreseen in 2023/2024
  - *DynaWave* - Operational use foreseen in 2025/2026

- Long-term research works for use in EMT simulations (PhD in cooperation with EP Montréal(10)), multi-domain simulations and robustness and performance improvements (PhD in cooperation with CUT for numerical methods)

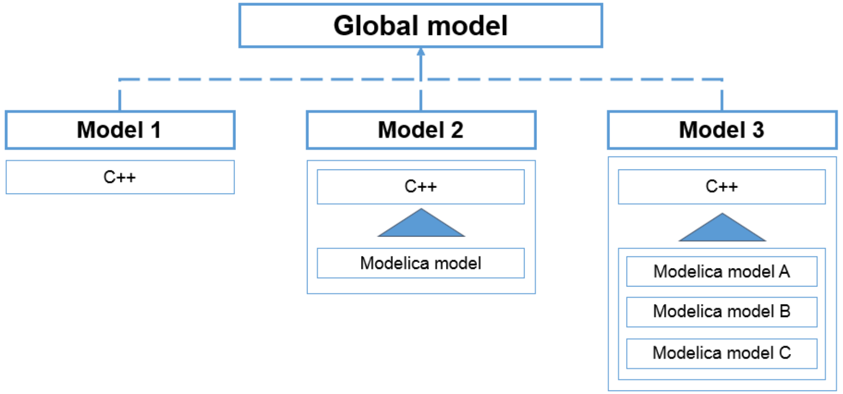
03

Dynawo and OpenModelica

A tool based on the OpenModelica Compiler

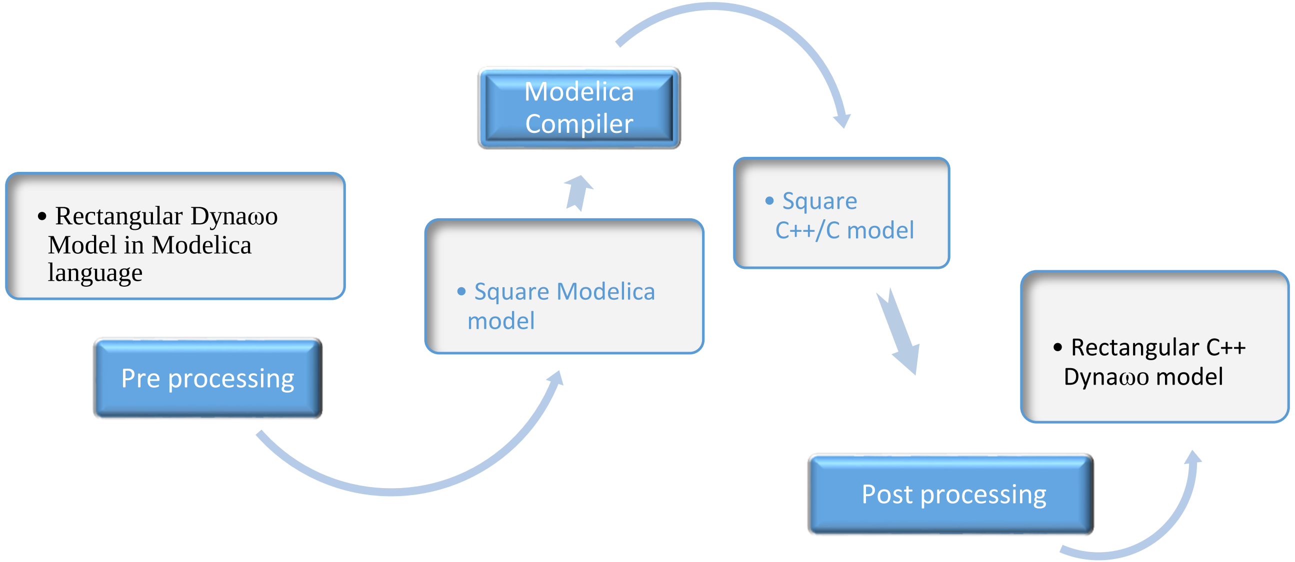
- RTE’s strategical choice to build upon existing open-source solution tools for Modelica compilation
  - In order to share efforts, to get inspirations and to be able to discuss with different domains experts as well as Modelica experts.
  - Ended up in the choice of OpenModelica as the best solution after an extensive analysis of the opportunities provided by the different open-source Modelica tools back in 2014/2015

- The OpenModelica Compiler in combination with Python scripts is used to precompile the models library to give them an identical structure than C++ models that are instantiated at run-time by Dynaωo.



A tool based on the OpenModelica Compiler

- The OpenModelica Compiler in combination with Python scripts is used to precompile the models library to give them an identical structure than C++ models that are instantiated at run-time by Dynawo.
  - ➢ Getting rid of the performances issue by pre-compiling most of the models beforehands (during the tool compilation) and only instantiating them at run-time
  - ➢ Compiling only once each kind of model and then instantiate them as many times as needed for one simulation.
  - ➢ Done through a customized pipeline around the OpenModelica Compiler (only requesting to identify the external variables)



- Rectangular Dynaωo Model in Modelica language

Pre processing

- Square Modelica model

Modelica Compiler

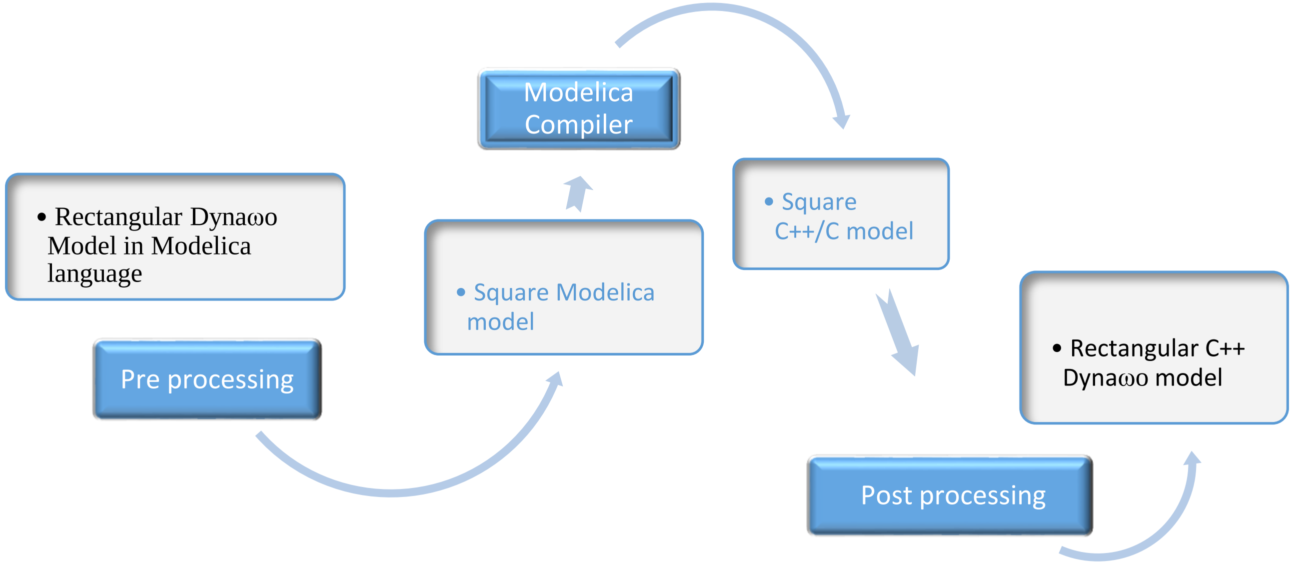
- Square C++/C model

Post processing

- Rectangular C++ Dynaωo model

A tool based on the OpenModelica Compiler

- The OpenModelica Compiler in combination with Python scripts is used to precompile the models library to give them an identical structure than C++ models that are instantiated at run-time by Dynawo.



- Rectangular Dynawo Model in Modelica language

Pre processing

- Square Modelica model

Modelica Compiler

- Square C++/C model

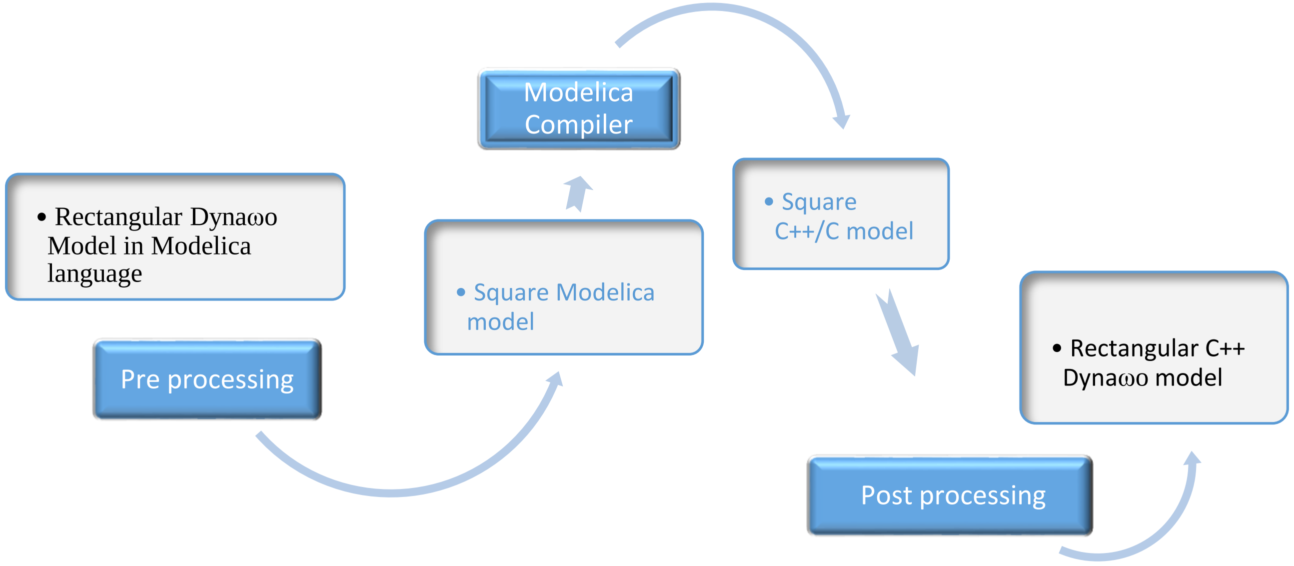
- Rectangular C++ Dynawo model

Post processing

```
model Line "AC power line - PI model"

/*
  Equivalent circuit and conventions:

               I1                  I2
   (terminal1) -->-------R+jX-------<-- (terminal2)
                     |            |
                    G+jB         G+jB
                     |            |
                    ---          ---
*/

  import Dynawo.Connectors;
  import Dynawo.Electrical.Controls.Basics.SwitchOff;

  extends SwitchOff.SwitchOffLine;
  extends AdditionalIcons.Line;

  Connectors.ACPower terminal1 annotation( ...);
  Connectors.ACPower terminal2 annotation( ...);

  parameter Types.PerUnit RPu "Resistance in p.u (base SnRef)";
  parameter Types.PerUnit XPu "Reactance in p.u (base SnRef)";
  parameter Types.PerUnit GPu "Half-conductance in p.u (base SnRef)";
  parameter Types.PerUnit BPu "Half-susceptance in p.u (base SnRef)";

protected
  parameter Types.ComplexImpedancePu ZPu (re = RPu, im = XPu) "Line impedance";
  parameter Types.ComplexAdmittancePu YPu (re = GPu, im = BPu) "Line half-admittance";

  Types.ActivePowerPu P1Pu "Active power on side 1 in p.u. (base SnRef)";
  Types.ReactivePowerPu Q1Pu "Reactive power on side 1 in p.u. (base SnRef)";
  Types.ActivePowerPu P2Pu "Active power on side 2 in p.u. (base SnRef)";
  Types.ReactivePowerPu Q2Pu "Reactive power on side 2 in p.u. (base SnRef)";

equation

  if (running.value) then
    ZPu * (terminal2.i - YPu * terminal2.V) = terminal2.V - terminal1.V;
    ZPu * (terminal1.i - YPu * terminal1.V) = terminal1.V - terminal2.V;
  else
    terminal1.i = Complex (0);
    terminal2.i = Complex (0);
  end if;

  P1Pu = ComplexMath.real(terminal1.V * ComplexMath.conj(terminal1.i));
  Q1Pu = ComplexMath.imag(terminal1.V * ComplexMath.conj(terminal1.i));
  P2Pu = ComplexMath.real(terminal2.V * ComplexMath.conj(terminal2.i));
  Q2Pu = ComplexMath.imag(terminal2.V * ComplexMath.conj(terminal2.i));

annotation(preferredView = "text", ...);
end Line;
```

A tool based on the OpenModelica Compiler

- The OpenModelica Compiler in combination with Python scripts is used to precompile the models library to give them an identical structure than C++ models that are instantiated at run-time by Dynaωo.



- Rectangular Dynaωo Model in Modelica language

Pre processing

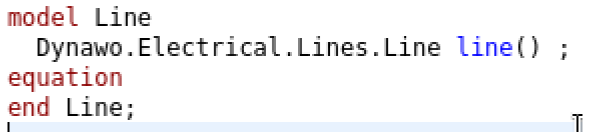
- Square Modelica model

Modelica Compiler

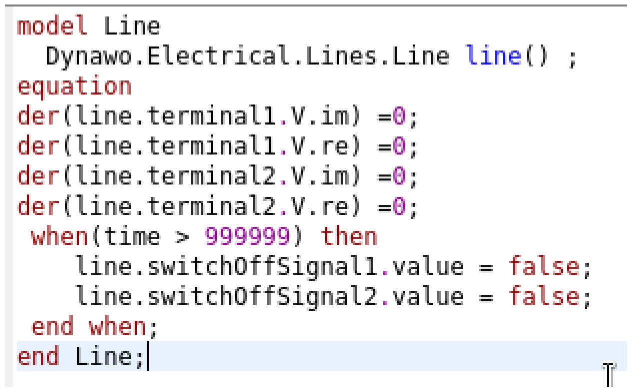
- Square C++/C model

- Rectangular C++ Dynaωo model

Post processing



```
model Line
  Dynawo.Electrical.Lines.Line line() ;
equation
end Line;
```

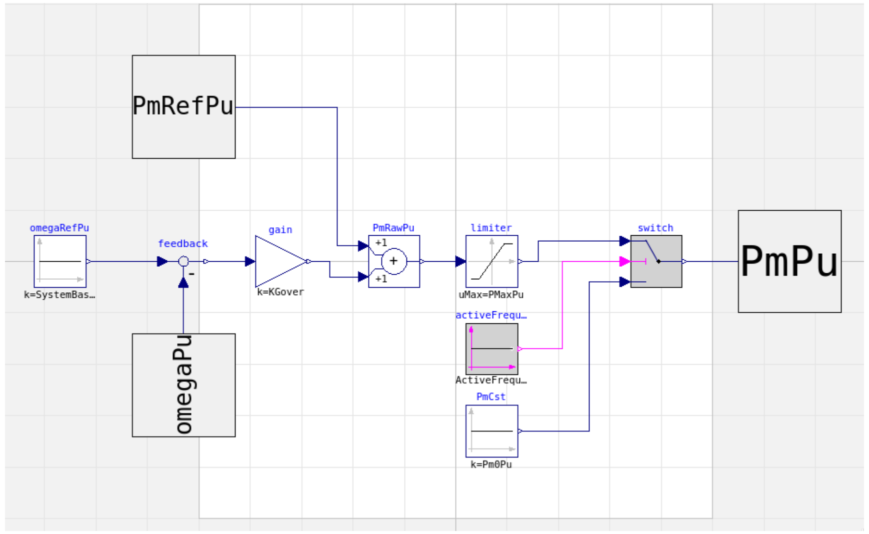


```
model Line
  Dynawo.Electrical.Lines.Line line() ;
equation
der(line.terminal1.V.im) =0;
der(line.terminal1.V.re) =0;
der(line.terminal2.V.im) =0;
der(line.terminal2.V.re) =0;
 when(time > 999999) then
    line.switchOffSignal1.value = false;
    line.switchOffSignal2.value = false;
 end when;
end Line;
```

Key features – Aliasing

- Aliasing is a key feature in OMC for large-scale power system simulations.
  - A lot of connect or equality equations exist in power system models
  - Appearing in control structures or between the different parts of a component.

- Generator example
  - Instantiated around <u>500 times</u> on a French test case
  - Comprises the physical synchronous machine plus a governor model, a voltage regulation model, an under-voltage automaton, a step-up transformer and a reactive power control loop (for secondary voltage control).
  - <u>187 continuous variables, 53 of them are alias</u>
  - <u>82 discrete variables, 12 of them are alias</u>



Key features – Aliasing

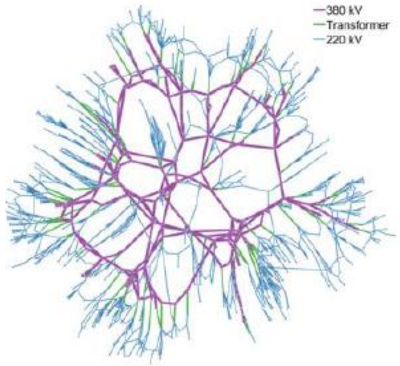
- Aliasing is a key feature in OMC for large-scale power system simulations.
  - A lot of connect or equality equations exist in power system models
  - Appearing in control structures or between the different parts of a component.

- Load example
  - Instantiated around 9 000 times on a French test case
  - Comprises the alpha-beta load model plus two transformers plus two tap-changers
  - 37 continuous variables, 14 of them are alias
  - 43 discrete variables, 8 of them are alias

```
model LoadTwoTransformersTapChangers
  Dynawo.Electrical.Loads.LoadAlphaBeta load() ;
  Dynawo.Electrical.Controls.Transformers.TapChanger tapChangerD() ;
  Dynawo.Electrical.Controls.Transformers.TapChanger tapChangerT() ;
  Dynawo.Electrical.Transformers.TransformerVariableTap transformerD() ;
  Dynawo.Electrical.Transformers.TransformerVariableTap transformerT() ;
equation
  connect(load.switchOffSignal1,transformerD.switchOffSignal1) ;
  connect(load.switchOffSignal2,transformerD.switchOffSignal2) ;
  connect(load.terminal,transformerD.terminal2) ;
  connect(tapChangerD.UMonitored,transformerD.U2Pu) ;
  connect(transformerD.switchOffSignal1,tapChangerD.switchOffSignal1) ;
  connect(transformerD.switchOffSignal2,tapChangerD.switchOffSignal2) ;
  connect(tapChangerD.tap,transformerD.tap) ;
  connect(tapChangerT.UMonitored,transformerT.U2Pu) ;
  connect(transformerT.switchOffSignal1,tapChangerT.switchOffSignal1) ;
  connect(transformerT.switchOffSignal2,tapChangerT.switchOffSignal2) ;
  connect(tapChangerT.tap,transformerT.tap) ;
  connect(transformerD.switchOffSignal1,transformerT.switchOffSignal1) ;
  connect(transformerD.switchOffSignal2,transformerT.switchOffSignal2) ;
  connect(transformerD.terminal1,transformerT.terminal2) ;
der(transformerT.terminal1.V.im) =0;
der(transformerT.terminal1.V.re) =0;
 when(time > 999999) then
    load.PRefPu.value = 0;
    load.QRefPu.value = 0;
    tapChangerD.locked = false;
    tapChangerD.switchOffSignal2.value = false;
    tapChangerT.locked = false;
    tapChangerT.switchOffSignal1.value = false;
 end when;
end LoadTwoTransformersTapChangers;
```

Key features – dae Mode

- Dae Mode is a key feature in OMC for large-scale power system simulations (11).
  - ➢ Power system is by nature a very sparse system
  - ➢ Power system models, except in the Electro-Magnetical Transient (EMT) domains, have large sets of algebraic equations



- The dae mode option in the Open Modelica compiler(12,13) enables to:
  - ➢ Avoid a few compilation failures
  - ➢ Preserve the system sparsity

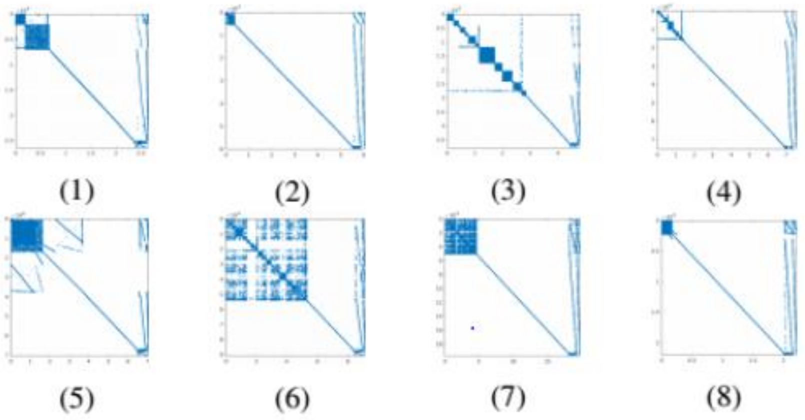
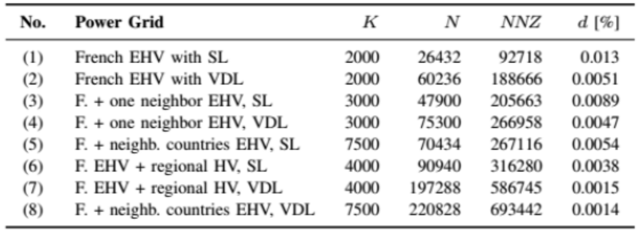
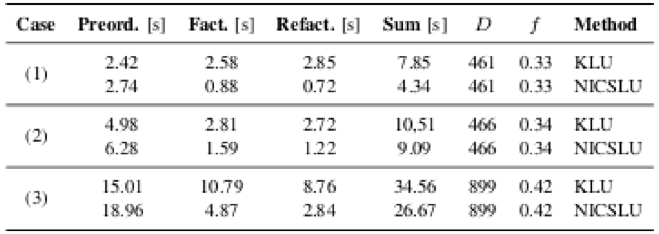


Fig. 1: Matrix sparsity patterns



No.	Power Grid	K	N	NNZ	d [%]
(1)	French EHV with SL	2000	26432	92718	0.013
(2)	French EHV with VDL	2000	60236	188666	0.0051
(3)	F. + one neighbor EHV, SL	3000	47900	205663	0.0089
(4)	F. + one neighbor EHV, VDL	3000	75300	266958	0.0047
(5)	F. + neighb. countries EHV, SL	7500	70434	267116	0.0054
(6)	F. EHV + regional HV, SL	4000	90940	316280	0.0038
(7)	F. EHV + regional HV, VDL	4000	197288	586745	0.0015
(8)	F. + neighb. countries EHV, VDL	7500	220828	693442	0.0014

TABLE I: Characteristics of squared matrices with size $N \times N$, K nodes, sorted by nonzeros NNZ, and with density factor $d = \frac{NNZ}{N \cdot N}$ in %



Case	Preord. [s]	Fact. [s]	Refact. [s]	Sum [s]	D	f	Method
(1)	2.42	2.58	2.85	7.85	461	0.33	KLU
	2.74	0.88	0.72	4.34	461	0.33	NICSLU
(2)	4.98	2.81	2.72	10,51	466	0.34	KLU
	6.28	1.59	1.22	9.09	466	0.34	NICSLU
(3)	15.01	10.79	8.76	34.56	899	0.42	KLU
	18.96	4.87	2.84	26.67	899	0.42	NICSLU

TABLE III: Accumulated execution times for the listed steps of the variable time step solver, with D LU decompositions and a factorization ratio $f = \frac{\#\text{Fact.}}{\#\text{Refact.}}$

Perspectives – Symbolic Jacobian in dae Mode

- No Symbolic Jacobian provided by the Open Modelica Compiler in dae mode
  - ➢ Forces to use an automatic differentiation algorithm in Dynaωo to compute the Jacobian for Modelica models
  - ➢ Currently done with Adept, leading to a large additional cost (residual re-evaluation with specific types – *2 to *3 compared to a classical evaluation, no use of sparsity information in the evaluation part, etc.)
  - ➢ Unstability in terms of nnz elements, leading to a larger number of complete LU decompositions compared to what one would expect.
  - ➢ Forces to try to use methods on the numerical side to avoid Jacobian evaluations (inexact Newton method)

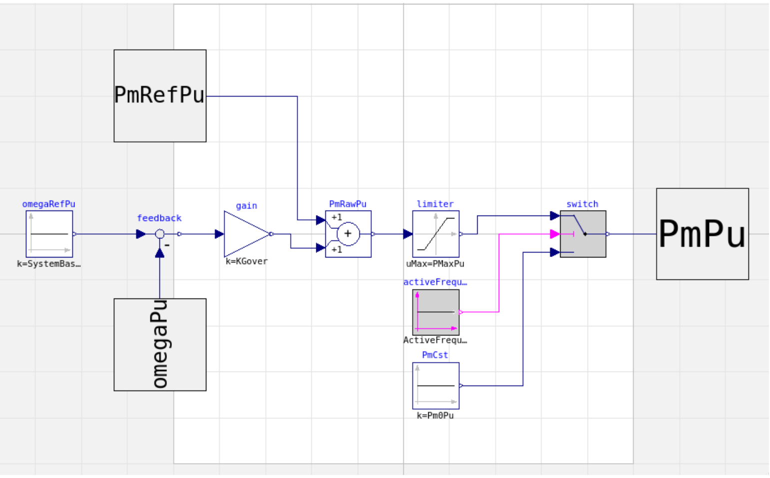
⇒Having such a feature would be a game changer for the operational use of any Modelica-based solution.

⇒It will also open up more possibilities related to reuse of constant parts or similar improvements(14).

Perspectives – Observers and advanced aliasing

- One key aspect for performance improvement is to further reduce the number of variables
    - Advanced aliasing can be developed to further reduce the number of variables
    - “Observers” can be introduced to separate between the variables necessary to the system resolution and the variables that are not necessary to the system resolution

- Advanced aliasing can certainly be applied on control structures for example
    - Gain block can be replaced by an aliased structure
    - Other elementary operations can certainly be replaced too
    - Up to a few hundreds or thousands variables on a French test case over 55 000 variables.

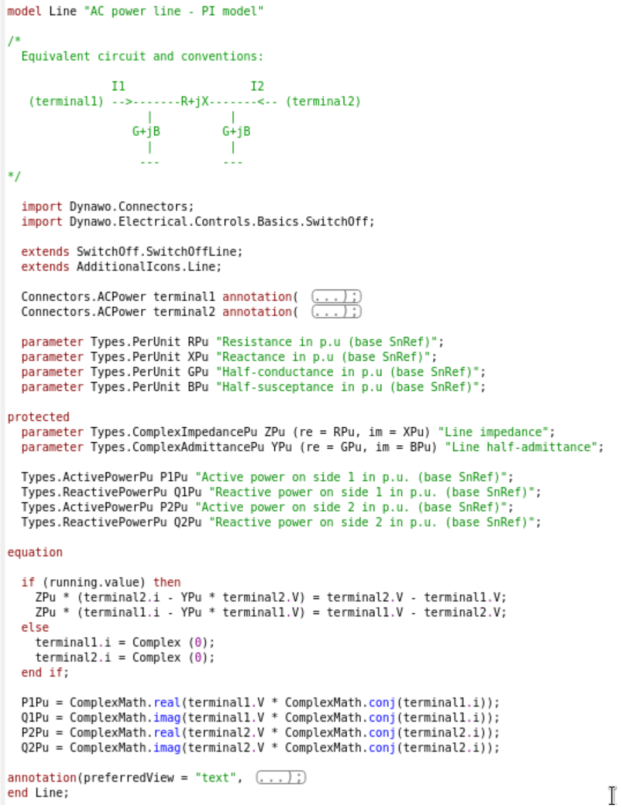


http://www.dynawo.org – rte-dynawo@rte-france.com

Perspectives – Observers and advanced aliasing

- One key aspect for performance improvement is to further reduce the number of variables
  - ➢ Advanced aliasing can be developed to further reduce the number of variables
  - ➢ “Observers” can be introduced to separate between the variables necessary to the system resolution and the variables that are not necessary to the system resolution

- “Observers” are very common structure in power system models
  - ➢ Generally speaking, the internal control structures are modeled using a p.u. basis (enabling to have common control parameter values whatever the size of the generators).
  - ➢ End-users want to have SI units.
  - ➢ A lot of conversions from p.u. to SI are done on the interesting values.
  - ➢ Some of the variables in SI are not used in any other equations than the SI <-> p.u. equation.
  - ➢ It is possible to exclude these variables and equations from the main problem.



```
model Line "AC power line - PI model"

/*
  Equivalent circuit and conventions:

               I1                  I2
   (terminal1) -->-------R+jX-------<-- (terminal2)
                    |           |
                  G+jB         G+jB
                    |           |
                   ---         ---
*/

  import Dynawo.Connectors;
  import Dynawo.Electrical.Controls.Basics.SwitchOff;

  extends SwitchOff.SwitchOffLine;
  extends AdditionalIcons.Line;

  Connectors.ACPower terminal1 annotation( ...);
  Connectors.ACPower terminal2 annotation( ...);

  parameter Types.PerUnit RPu "Resistance in p.u (base SnRef)";
  parameter Types.PerUnit XPu "Reactance in p.u (base SnRef)";
  parameter Types.PerUnit GPu "Half-conductance in p.u (base SnRef)";
  parameter Types.PerUnit BPu "Half-susceptance in p.u (base SnRef)";

protected
  parameter Types.ComplexImpedancePu ZPu (re = RPu, im = XPu) "Line impedance";
  parameter Types.ComplexAdmittancePu YPu (re = GPu, im = BPu) "Line half-admittance";

  Types.ActivePowerPu P1Pu "Active power on side 1 in p.u. (base SnRef)";
  Types.ReactivePowerPu Q1Pu "Reactive power on side 1 in p.u. (base SnRef)";
  Types.ActivePowerPu P2Pu "Active power on side 2 in p.u. (base SnRef)";
  Types.ReactivePowerPu Q2Pu "Reactive power on side 2 in p.u. (base SnRef)";

equation

  if (running.value) then
    ZPu * (terminal2.i - YPu * terminal2.V) = terminal2.V - terminal1.V;
    ZPu * (terminal1.i - YPu * terminal1.V) = terminal1.V - terminal2.V;
  else
    terminal1.i = Complex (0);
    terminal2.i = Complex (0);
  end if;

  P1Pu = ComplexMath.real(terminal1.V * ComplexMath.conj(terminal1.i));
  Q1Pu = ComplexMath.imag(terminal1.V * ComplexMath.conj(terminal1.i));
  P2Pu = ComplexMath.real(terminal2.V * ComplexMath.conj(terminal2.i));
  Q2Pu = ComplexMath.imag(terminal2.V * ComplexMath.conj(terminal2.i));

annotation(preferredView = "text", ...);
end Line;
```

Perspectives – Observers and advanced aliasing

- One key aspect for performance improvement is to further reduce the number of variables
  - Advanced aliasing can be developed to further reduce the number of variables
  - “Observers” can be introduced to separate between the variables necessary to the system resolution and the variables that are not necessary to the system resolution

- “Observers” are very common structure in power system models
  - Generally speaking, the internal control structures are modeled using a p.u. basis (enabling to have common control parameter values whatever the size of the generators for example).
  - End-users want to have SI units.
  - A lot of conversions from p.u. to SI are done on the interesting values.
  - Some of the variables in SI are not used in any other equations than the SI <-> p.u. equation.
  - It is possible to exclude these variables and equations from the main problem.

  - Generator model: <u>47 variables out of 187</u>
  - Load model: <u>7 variables out of 37</u>

Perspectives – Efficient run-time code

- The code generated by the OpenModelica compiler can be further enhanced to take advantage of common structures
  - ➢ Mutualizing the when and if condition evaluations
  - ➢ Especially on large-scale and long-term simulations (large number of calls)

```
  //Transition to "Locked" (possible from any state and prioritary)
  when (not running.value) or locked then
    state = State.Locked;
    tap.value = pre(tap.value);
    tTapUp = Constants.inf;
    tTapDown = Constants.inf;
  //Transition to "WaitingToMoveDown" (possible from any state except down states)
  elsewhen lookingToDecreaseTap and (pre(state) == State.Standard or pre(state) == State.MoveUp1 or pre(state) == State.MoveUpN or pre(state) == State.WaitingToMoveUp or pre(state) ==
State.Locked) and running.value and not(locked) then
    state = State.WaitingToMoveDown;
    tap.value = pre(tap.value);
    tTapUp = Constants.inf;
    tTapDown = Constants.inf;
  //Transition to "WaitingToMoveUp" (possible from any state except up states)
  elsewhen lookingToIncreaseTap and (pre(state) == State.Standard or pre(state) == State.MoveDown1 or pre(state) == State.MoveDownN or pre(state) == State.WaitingToMoveDown or pre(state)
== State.Locked) and running.value and not(locked) then
    state = State.WaitingToMoveUp;
    tap.value = pre(tap.value);
    tTapUp = Constants.inf;
    tTapDown = Constants.inf;
  //Transition to "Standard" (possible from any state)
  elsewhen valueUnderStop and pre(state) <> State.Standard and running.value and not(locked) then
    state = State.Standard;
    tap.value = pre(tap.value);
    tTapUp = Constants.inf;
    tTapDown = Constants.inf;
  //Transition to "MoveDown1" (only possible from WaitingToMoveDown)
  elsewhen pre(state) == State.WaitingToMoveDown and time - tValueAboveMaxWhileRunning>= t1st and pre(tap.value) > tapMin then
    state = State.MoveDown1;
    tap.value = pre(tap.value) - 1;
    tTapUp = pre(tTapUp);
    tTapDown = time;
    Timeline.logEvent1(TimelineKeys.TapDown);
  //Transition to "MoveUp1" (only possible from WaitingToMoveUp)
  elsewhen pre(state) == State.WaitingToMoveUp and time - tValueAboveMaxWhileRunning>= t1st and pre(tap.value) < tapMax then
    state = State.MoveUp1;
    tap.value = pre(tap.value) + 1;
    tTapUp = time;
    tTapDown = pre(tTapDown);
    Timeline.logEvent1(TimelineKeys.TapUp);
  //Transition to "MoveDownN" (only possible from MoveDown1 or MoveDownN)
  elsewhen (pre(state) == State.MoveDown1 or pre(state) == State.MoveDownN) and time - pre(tTapDown) >= tNext and pre(tap.value) > tapMin then
    state = State.MoveDownN;
    tap.value = pre(tap.value) - 1;
    tTapUp = pre(tTapUp);
    tTapDown = time;
    Timeline.logEvent1(TimelineKeys.TapDown);
  //Transition to "MoveUpN" (only possible from MoveUp1 or MoveUpN)
  elsewhen (pre(state) == State.MoveUp1 or pre(state) == State.MoveUpN) and time - pre(tTapUp) >= tNext and pre(tap.value) < tapMax then
    state = State.MoveUpN;
    tap.value = pre(tap.value) + 1;
    tTapUp = time;
    tTapDown = pre(tTapDown);
    Timeline.logEvent1(TimelineKeys.TapUp);
  end when;

annotation(preferredView = "text");
end BaseTapChangerPhaseShifter_MAX;
```

```
/*
equation index: 453
type: WHEN

when {$whenCondition27} then
  tapChangerT._tTapDown = 9.999999999999999e+59;
end when;
*/
void LoadTwoTransformersTapChangers_eqFunction_453(DATA *data, threadData_t *threadData)
{
  TRACE_PUSH
  const int equationIndexes[2] = {1,453};
  if((data->localData[0]->booleanVars[19] /* $whenCondition27 DISCRETE */ && !data->simulationInfo->booleanVarsPre[19] /* $whenCondition27 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = 9.999999999999999e+59;
  }
  else if((data->localData[0]->booleanVars[18] /* $whenCondition26 DISCRETE */ && !data->simulationInfo->booleanVarsPre[18] /* $whenCondition26 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = 9.999999999999999e+59;
  }
  else if((data->localData[0]->booleanVars[17] /* $whenCondition25 DISCRETE */ && !data->simulationInfo->booleanVarsPre[17] /* $whenCondition25 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = 9.999999999999999e+59;
  }
  else if((data->localData[0]->booleanVars[16] /* $whenCondition24 DISCRETE */ && !data->simulationInfo->booleanVarsPre[16] /* $whenCondition24 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = 9.999999999999999e+59;
  }
  else if((data->localData[0]->booleanVars[15] /* $whenCondition23 DISCRETE */ && !data->simulationInfo->booleanVarsPre[15] /* $whenCondition23 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = data->localData[0]->timeValue;
  }
  else if((data->localData[0]->booleanVars[14] /* $whenCondition22 DISCRETE */ && !data->simulationInfo->booleanVarsPre[14] /* $whenCondition22 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = data->simulationInfo->realVarsPre[32] /* tapChangerT.tTapDown DISCRETE */;
  }
  else if((data->localData[0]->booleanVars[13] /* $whenCondition21 DISCRETE */ && !data->simulationInfo->booleanVarsPre[13] /* $whenCondition21 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = data->localData[0]->timeValue;
  }
  else if((data->localData[0]->booleanVars[12] /* $whenCondition20 DISCRETE */ && !data->simulationInfo->booleanVarsPre[12] /* $whenCondition20 DISCRETE */ /* edge */))
  {
    data->localData[0]->realVars[32] /* tapChangerT.tTapDown DISCRETE */ = data->simulationInfo->realVarsPre[32] /* tapChangerT.tTapDown DISCRETE */;
  }
  TRACE_POP
}
```

04

Conclusions and perspectives

Conclusions and perspectives

- Dynaωo - An hybrid C++/Modelica open-source suite of simulation tools
  - ➢ A very strong strategical choice from RTE to push for the development of open-source solutions for power system simulations
  - ➢ A mature project evolving to a suite of simulation tools to renew the whole range of operational software in the next few years
  - ➢ An approach based on flexibility, transparency, quality and acceptable performances that gain interest in the power system community

- Dynaωo – A simulation tool built upon the OpenModelica compiler.
  - ➢ Focus on the simulation time and the features enabling to speed it up
  - ➢ Key features already available in the OpenModelica compiler (aliasing, common subexpression elimination or dae mode)
  - ➢ Additional features could reduce the gap with programming language approaches and ensure the long-term approach long-term viability
  - ➢ Will to make OpenModelica in general providing new features and offering even higher quality.

Questions

http://www.dynawo.org – rte-dynawo@rte-france.com

References

- (1) Power system modeling in Modelica for time-domain simulation, A. Chieh, P. Panciatici, and J.Picard, IEEE Powertech 2011.
- (2) Unambiguous power system dynamic modeling and simulation using Modelica tools, L. Vanfretti, W. Li, T. Bogodorova and P. Panciatici, IEEE PES GM 2013
- (3) A Modelica power system library for phasor time-domain simulation, T. Bogodorova, M. Sabate, G. Leon, L. Vanfretti, M. Halat, J.-B. Heyberger and P. Panciatici, IEEE ISGT Europe 2013
- (4) A tool to ease Modelica-based Dynamic Power System Simulations, R. Viruez, S. Machado, L-M. Zamarreno, G. Leon, F. Beaude, S. Petitrenaud, J-B Heyberger, Modelica Conference 2017.
- (5) Towards an open-source solution using Modelica for Time Domain simulation of Power Systems, A. Guironnet, M. Saugier, S. Petitrenaud, F. Xavier and P. Panciatici, IEEE ISGT Europe 2018.
- (6) Towards Pan-European Power Grid Modelling in Modelica: Design Principles and A Prototype for a Reference Power System Library, F. Casella, A. Bartolini and A. Guironnet, Modelica Conference 2019
- (7) Speed-up of Modelica-based large scale simulations, the example of the open-source tool Dynaωo, A. Guironnet and F. Xavier, IEEE PES GM 2020
- (8) A Novel Approach for the calculation of steady states in transmission systems using simplified time-domain simulation, Q. Cossart, M. Chiaramello, A. Guironnet and P. Panciatici, 2021

References

- (9) An open-source implementation of Grid-Forming converters using Modelica, Q. Cossart, F. Rosière, A. Guironnet and M. Saugier, Accepted for IEEE ISGT Europe 2020.
- (10) Simulation of Electromagnetic transients with Modelica, Accuracy and Performance Assessments for Transmission Line Models, A. Masoom, T. Ould-Bachir, J. Mahseredjian, A. Guironnet and N. Ding, PSCC 2020
- (11) A comparative analysis of LU decomposition methods for power system simulations, L. Schumacher, L. Razik, A. Monti, G. Bureau and A. Guironnet, IEEE Powertech 2019.
- (12) Experience on the use of the DAE mode in industrial power system simulations, K.Abdelhak, B. Bachmann, F. Rosière and A. Guironnet, Presentation at the 2020 OpenModelica Workshop, https://openmodelica.org/images/M_images/OpenModelicaWorkshop_2020/PresentationOMDAE.pdf
- (13) Contributions to the efficient and parallel Jacobian evaluation and its application in OpenModelica, W. Braun, M. Schroschk, V. Ruge, A. Heuermann and B. Bachmann, America Modelica conference, 2020