Modelling and Simulation of Power Systems with Grid-Connected Converters in OpenModelica

OpenModelica Workshop 2022

Lluc Figueras Llerins
Vinicius Albernaz Lacerda
Eduardo Prieto Araujo

Adrien Guironnet
Quentin Cossart

Outline

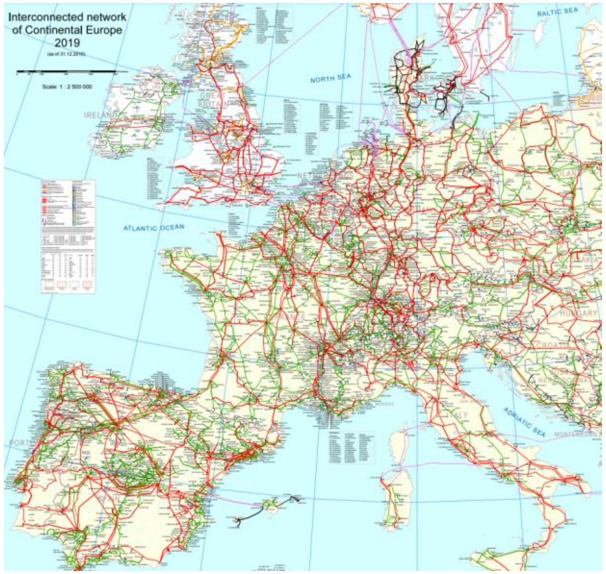
1. New challenges in power systems simulation
2. Test case and Modelica models implementation
3. OpenModelica vs Simulink simulation performance
4. Conclusions and future work

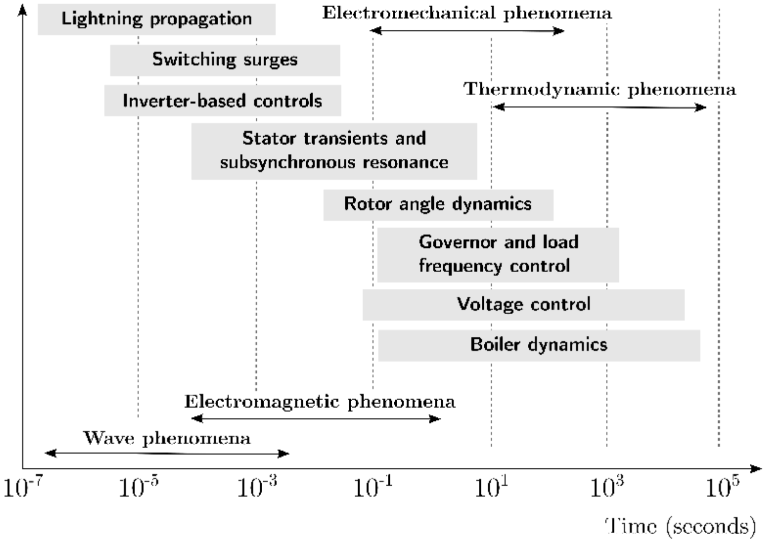
New challenges

The power system is evolving from a well-known and predictable behaviour to a more complex, unpredictable and numerically-driven system.

- Increasing penetration of power electronic devices
- Open and flexible simulation environments crucial to deal with new challenges
- Complex systems → increasing computational burden

Complete interoperability between power system actors require common modelling and simulation tools





Traditional approach and current situation

Closed architecture and proprietary tools

- Use imperative programming languages (FORTRAN, C, C++...).
- Efficient but large sets of hard-coded data unnaccessible to users.
- Internal model representation difficult to understand and share.
- Strong coupling between the solver and the model.
- Solver- and implementation-dependent black-box models.
- Simulation results inconsistent between tools (different assumptions, simplifications, modelling philosophy)

Traditional approach and current situation

Data exchange between TSOs thanks to the Common Grid Model Exchange Standard (CGMES).

- Sufficient for static load flow studies and parametric data exchange.

However,

- Unsuitable to exchange dynamic models for time-domain simulations.
- The mathematical representation of the model is not shared explicitly.
- Different software give different results for the same data set.
- Final model implementation and mathematical solving methods are inaccessible black-box models.

➡ **Both data and equations exchange appears to be the only viable solution in the long term**

Traditional approach and current situation

- Limited collaboration, interoperability, portability, transparency and flexibility

↓

- Necessity of a common language for dynamic models

↓

Development of common standarized languages
Development of Open Source Software

OpenModelica

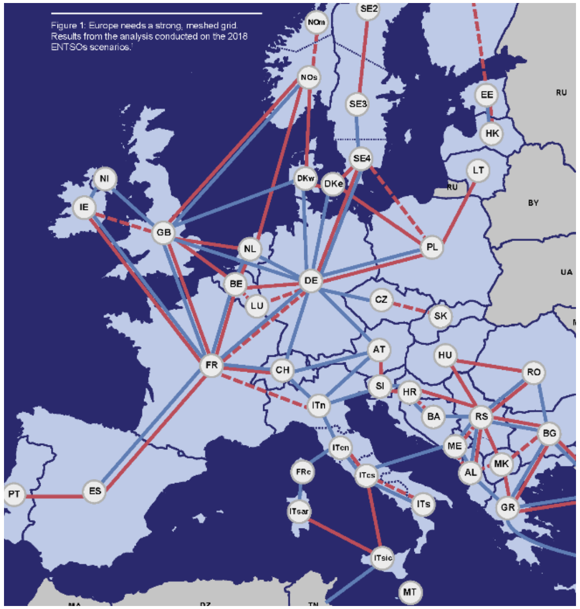


Figure 1: Europe needs a strong, meshed grid. Results from the analysis conducted on the 2018 ENTSOs scenarios.

Modelica

Promising approach for power system modelling and simulation

Increasing interest in the power system community to use Modelica as the common standardized language

- Excellent candidate to provide an open standard implementation
- Disconnect the dependency between the power system tool and the power system model
- Decoupling of the solver and the model
- Environments to transform Modelica code into executable simulation code.
  - Open-source: **OpenModelica, Dynawo**.

OpenModelica DYNAWO

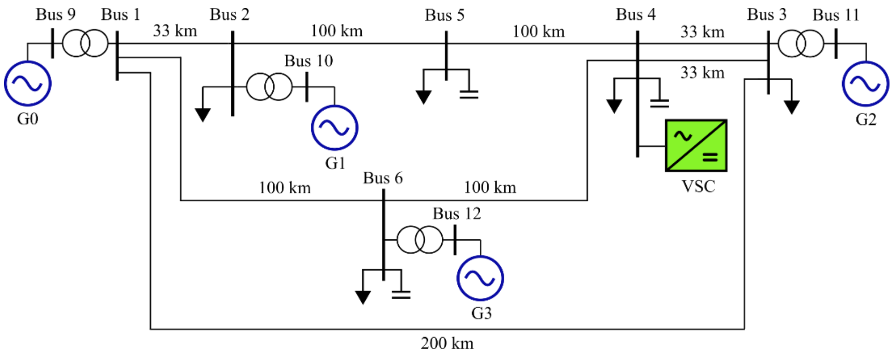
Test case: CIGRE HV transmission benchmark system

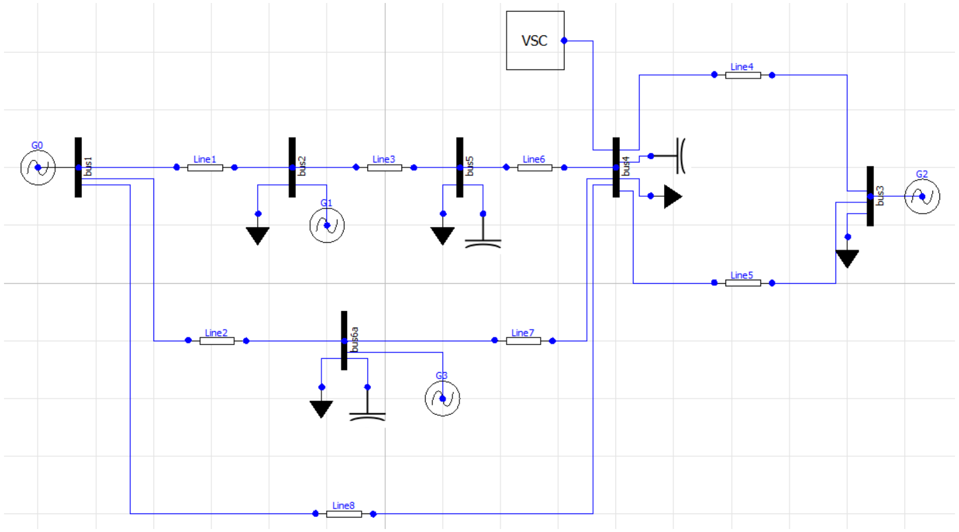
Assess OpenModelica's

- Robustness
- Flexibility
- Accuracy
- Computational performance

EMT-type modelling and simulation

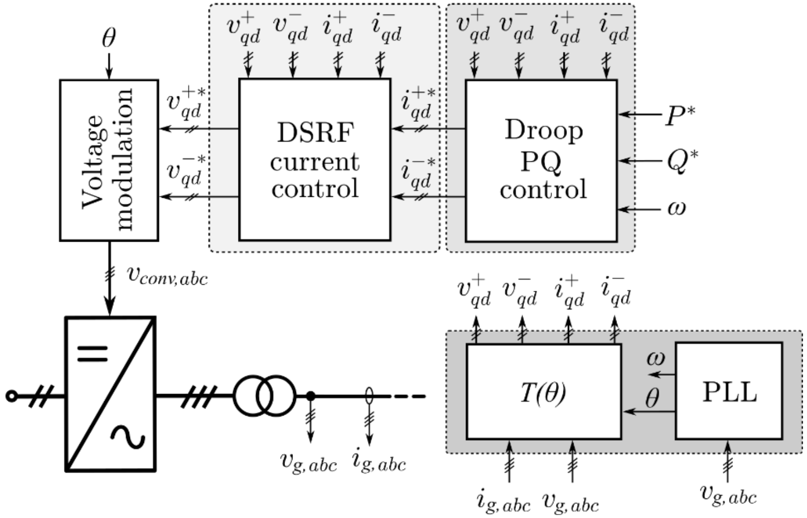
- Components defined by their differential equations → high-level accuracy
- Represent non-linearities
- Frequency dependant effects
- Unbalanced networks

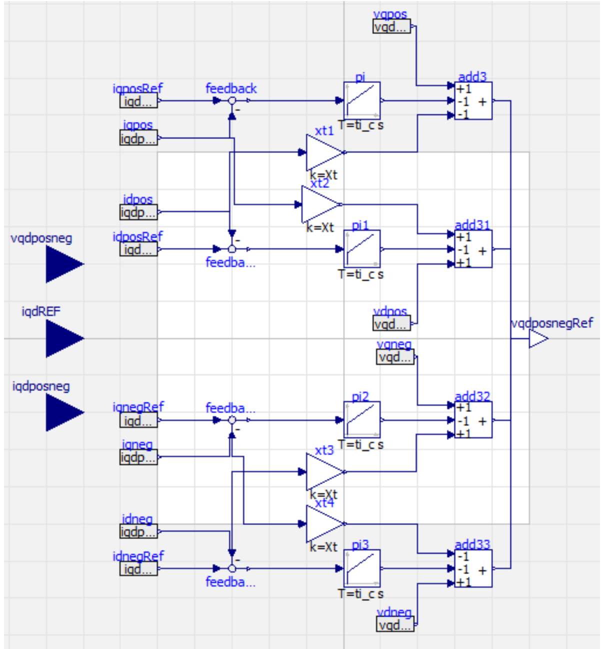


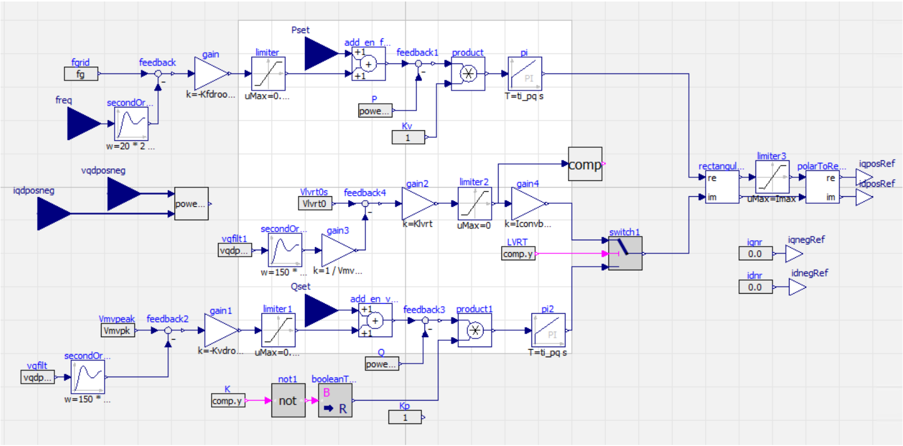


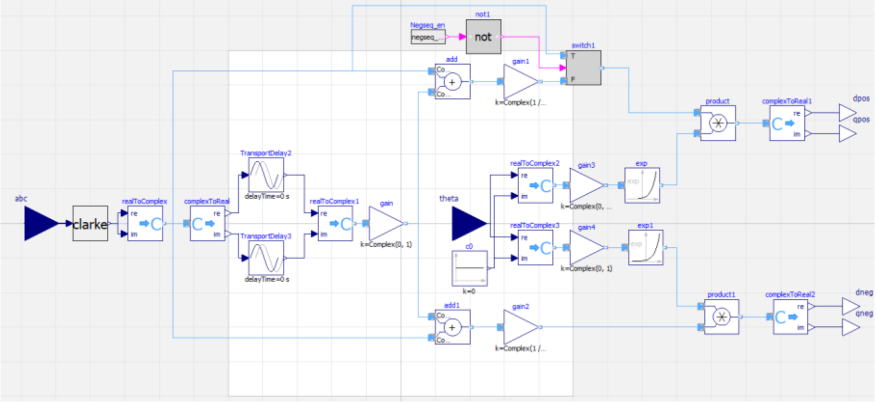
Adaptation of the CIGRE European HV transmission network benchmark system

2L-VSC Modelica implementation

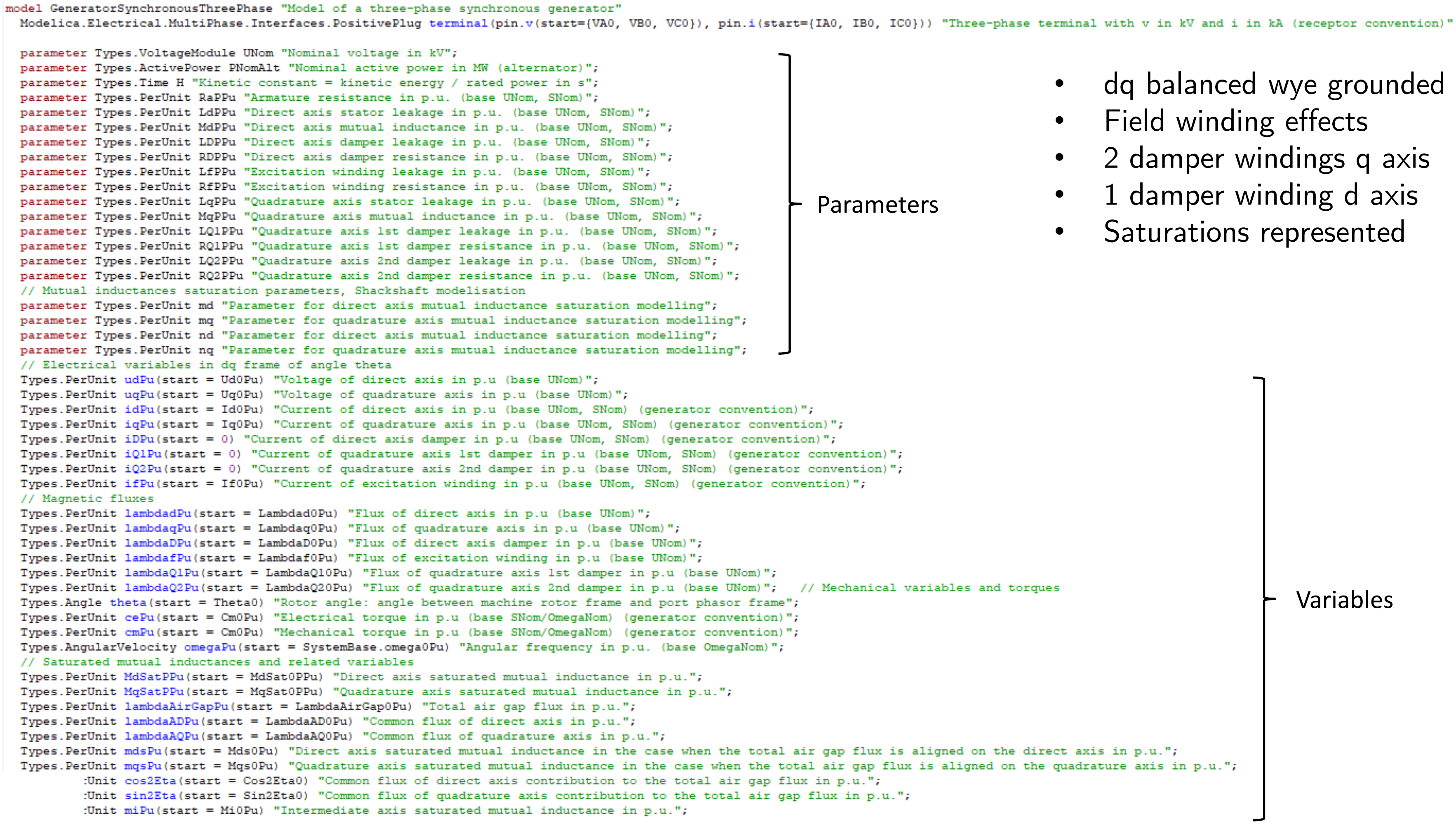








Synchronous machine Modelica implementation



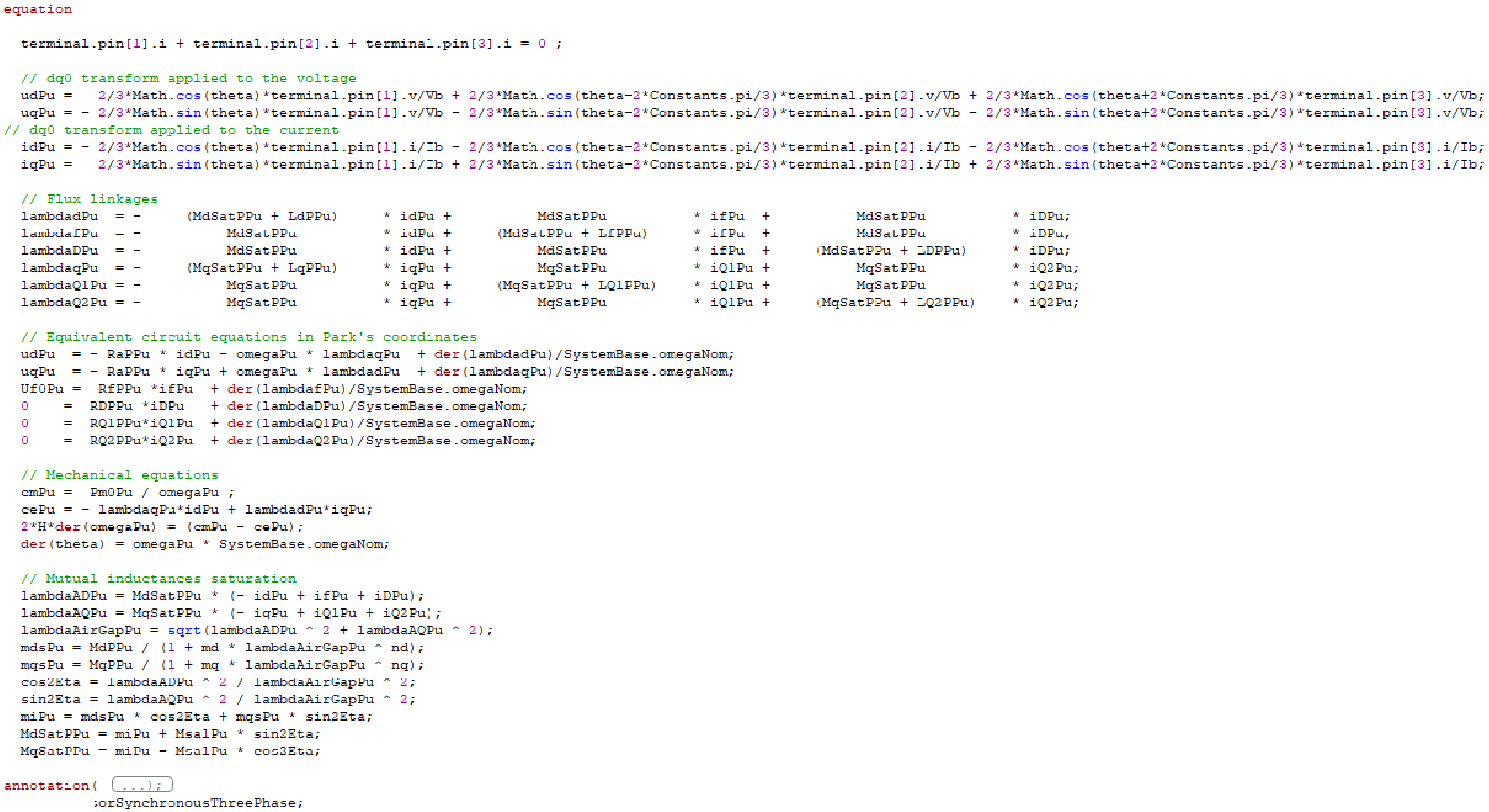
```
model GeneratorSynchronousThreePhase "Model of a three-phase synchronous generator"
  Modelica.Electrical.MultiPhase.Interfaces.PositivePlug terminal(pin.v(start={VA0, VB0, VC0}), pin.i(start={IA0, IB0, IC0})) "Three-phase terminal with v in kV and i in kA (receptor convention)"

  parameter Types.VoltageModule UNom "Nominal voltage in kV";
  parameter Types.ActivePower PNomAlt "Nominal active power in MW (alternator)";
  parameter Types.Time H "Kinetic constant = kinetic energy / rated power in s";
  parameter Types.PerUnit RaPPu "Armature resistance in p.u. (base UNom, SNom)";
  parameter Types.PerUnit LdPPu "Direct axis stator leakage in p.u. (base UNom, SNom)";
  parameter Types.PerUnit MdPPu "Direct axis mutual inductance in p.u. (base UNom, SNom)";
  parameter Types.PerUnit LDPPu "Direct axis damper leakage in p.u. (base UNom, SNom)";
  parameter Types.PerUnit RDPPu "Direct axis damper resistance in p.u. (base UNom, SNom)";
  parameter Types.PerUnit LfPPu "Excitation winding leakage in p.u. (base UNom, SNom)";
  parameter Types.PerUnit RfPPu "Excitation winding resistance in p.u. (base UNom, SNom)";
  parameter Types.PerUnit LqPPu "Quadrature axis stator leakage in p.u. (base UNom, SNom)";
  parameter Types.PerUnit MqPPu "Quadrature axis mutual inductance in p.u. (base UNom, SNom)";
  parameter Types.PerUnit LQ1PPu "Quadrature axis 1st damper leakage in p.u. (base UNom, SNom)";
  parameter Types.PerUnit RQ1PPu "Quadrature axis 1st damper resistance in p.u. (base UNom, SNom)";
  parameter Types.PerUnit LQ2PPu "Quadrature axis 2nd damper leakage in p.u. (base UNom, SNom)";
  parameter Types.PerUnit RQ2PPu "Quadrature axis 2nd damper resistance in p.u. (base UNom, SNom)";
  // Mutual inductances saturation parameters, Shackshaft modelisation
  parameter Types.PerUnit md "Parameter for direct axis mutual inductance saturation modelling";
  parameter Types.PerUnit mq "Parameter for quadrature axis mutual inductance saturation modelling";
  parameter Types.PerUnit nd "Parameter for direct axis mutual inductance saturation modelling";
  parameter Types.PerUnit nq "Parameter for quadrature axis mutual inductance saturation modelling";
  // Electrical variables in dq frame of angle theta
  Types.PerUnit udPu(start = Ud0Pu) "Voltage of direct axis in p.u (base UNom)";
  Types.PerUnit uqPu(start = Uq0Pu) "Voltage of quadrature axis in p.u (base UNom)";
  Types.PerUnit idPu(start = Id0Pu) "Current of direct axis in p.u (base UNom, SNom) (generator convention)";
  Types.PerUnit iqPu(start = Iq0Pu) "Current of quadrature axis in p.u (base UNom, SNom) (generator convention)";
  Types.PerUnit iDPu(start = 0) "Current of direct axis damper in p.u (base UNom, SNom) (generator convention)";
  Types.PerUnit iQ1Pu(start = 0) "Current of quadrature axis 1st damper in p.u (base UNom, SNom) (generator convention)";
  Types.PerUnit iQ2Pu(start = 0) "Current of quadrature axis 2nd damper in p.u (base UNom, SNom) (generator convention)";
  Types.PerUnit ifPu(start = If0Pu) "Current of excitation winding in p.u (base UNom, SNom) (generator convention)";
  // Magnetic fluxes
  Types.PerUnit lambdadPu(start = Lambdad0Pu) "Flux of direct axis in p.u (base UNom)";
  Types.PerUnit lambdaqPu(start = Lambdaq0Pu) "Flux of quadrature axis in p.u (base UNom)";
  Types.PerUnit lambdaDPu(start = LambdaD0Pu) "Flux of direct axis damper in p.u (base UNom)";
  Types.PerUnit lambdafPu(start = Lambdaf0Pu) "Flux of excitation winding in p.u (base UNom)";
  Types.PerUnit lambdaQ1Pu(start = LambdaQ10Pu) "Flux of quadrature axis 1st damper in p.u (base UNom)";
  Types.PerUnit lambdaQ2Pu(start = LambdaQ20Pu) "Flux of quadrature axis 2nd damper in p.u (base UNom)";   // Mechanical variables and torques
  Types.Angle theta(start = Theta0) "Rotor angle: angle between machine rotor frame and port phasor frame";
  Types.PerUnit cePu(start = Cm0Pu) "Electrical torque in p.u (base SNom/OmegaNom) (generator convention)";
  Types.PerUnit cmPu(start = Cm0Pu) "Mechanical torque in p.u (base SNom/OmegaNom) (generator convention)";
  Types.AngularVelocity omegaPu(start = SystemBase.omega0Pu) "Angular frequency in p.u. (base OmegaNom)";
  // Saturated mutual inductances and related variables
  Types.PerUnit MdSatPPu(start = MdSat0PPu) "Direct axis saturated mutual inductance in p.u.";
  Types.PerUnit MqSatPPu(start = MqSat0PPu) "Quadrature axis saturated mutual inductance in p.u.";
  Types.PerUnit lambdaAirGapPu(start = LambdaAirGap0Pu) "Total air gap flux in p.u.";
  Types.PerUnit lambdaADPu(start = LambdaAD0Pu) "Common flux of direct axis in p.u.";
  Types.PerUnit lambdaAQPu(start = LambdaAQ0Pu) "Common flux of quadrature axis in p.u.";
  Types.PerUnit mdsPu(start = Mds0Pu) "Direct axis saturated mutual inductance in the case when the total air gap flux is aligned on the direct axis in p.u.";
  Types.PerUnit mqsPu(start = Mqs0Pu) "Quadrature axis saturated mutual inductance in the case when the total air gap flux is aligned on the quadrature axis in p.u.";
          :Unit cos2Eta(start = Cos2Eta0) "Common flux of direct axis contribution to the total air gap flux in p.u.";
          :Unit sin2Eta(start = Sin2Eta0) "Common flux of quadrature axis contribution to the total air gap flux in p.u.";
          :Unit miPu(start = Mi0Pu) "Intermediate axis saturated mutual inductance in p.u.";
```

Parameters

Variables

- dq balanced wye grounded
- Field winding effects
- 2 damper windings q axis
- 1 damper winding d axis
- Saturations represented

Synchronous machine Modelica implementation



```
equation

  terminal.pin[1].i + terminal.pin[2].i + terminal.pin[3].i = 0 ;

  // dq0 transform applied to the voltage
  udPu =   2/3*Math.cos(theta)*terminal.pin[1].v/Vb + 2/3*Math.cos(theta-2*Constants.pi/3)*terminal.pin[2].v/Vb + 2/3*Math.cos(theta+2*Constants.pi/3)*terminal.pin[3].v/Vb;
  uqPu = - 2/3*Math.sin(theta)*terminal.pin[1].v/Vb - 2/3*Math.sin(theta-2*Constants.pi/3)*terminal.pin[2].v/Vb - 2/3*Math.sin(theta+2*Constants.pi/3)*terminal.pin[3].v/Vb;
// dq0 transform applied to the current
  idPu = - 2/3*Math.cos(theta)*terminal.pin[1].i/Ib - 2/3*Math.cos(theta-2*Constants.pi/3)*terminal.pin[2].i/Ib - 2/3*Math.cos(theta+2*Constants.pi/3)*terminal.pin[3].i/Ib;
  iqPu =   2/3*Math.sin(theta)*terminal.pin[1].i/Ib + 2/3*Math.sin(theta-2*Constants.pi/3)*terminal.pin[2].i/Ib + 2/3*Math.sin(theta+2*Constants.pi/3)*terminal.pin[3].i/Ib;

  // Flux linkages
  lambdadPu  = -     (MdSatPPu + LdPPu)     * idPu +          MdSatPPu         * ifPu  +          MdSatPPu          * iDPu;
  lambdafPu  = -          MdSatPPu          * idPu +     (MdSatPPu + LfPPu)    * ifPu  +          MdSatPPu          * iDPu;
  lambdaDPu  = -          MdSatPPu          * idPu +          MdSatPPu         * ifPu  +     (MdSatPPu + LDPPu)     * iDPu;
  lambdaqPu  = -     (MqSatPPu + LqPPu)     * iqPu +          MqSatPPu         * iQ1Pu +          MqSatPPu          * iQ2Pu;
  lambdaQ1Pu = -          MqSatPPu          * iqPu +     (MqSatPPu + LQ1PPu)   * iQ1Pu +          MqSatPPu          * iQ2Pu;
  lambdaQ2Pu = -          MqSatPPu          * iqPu +          MqSatPPu         * iQ1Pu +     (MqSatPPu + LQ2PPu)    * iQ2Pu;

  // Equivalent circuit equations in Park's coordinates
  udPu  = - RaPPu * idPu - omegaPu * lambdaqPu  + der(lambdadPu)/SystemBase.omegaNom;
  uqPu  = - RaPPu * iqPu + omegaPu * lambdadPu  + der(lambdaqPu)/SystemBase.omegaNom;
  Uf0Pu =  RfPPu *ifPu  + der(lambdafPu)/SystemBase.omegaNom;
  0    =  RDPPu *iDPu   + der(lambdaDPu)/SystemBase.omegaNom;
  0    =  RQ1PPu*iQ1Pu  + der(lambdaQ1Pu)/SystemBase.omegaNom;
  0    =  RQ2PPu*iQ2Pu  + der(lambdaQ2Pu)/SystemBase.omegaNom;

  // Mechanical equations
  cmPu =  Pm0Pu / omegaPu ;
  cePu = - lambdaqPu*idPu + lambdadPu*iqPu;
  2*H*der(omegaPu) = (cmPu - cePu);
  der(theta) = omegaPu * SystemBase.omegaNom;

  // Mutual inductances saturation
  lambdaADPu = MdSatPPu * (- idPu + ifPu + iDPu);
  lambdaAQPu = MqSatPPu * (- iqPu + iQ1Pu + iQ2Pu);
  lambdaAirGapPu = sqrt(lambdaADPu ^ 2 + lambdaAQPu ^ 2);
  mdsPu = MdPPu / (1 + md * lambdaAirGapPu ^ nd);
  mqsPu = MqPPu / (1 + mq * lambdaAirGapPu ^ nq);
  cos2Eta = lambdaADPu ^ 2 / lambdaAirGapPu ^ 2;
  sin2Eta = lambdaAQPu ^ 2 / lambdaAirGapPu ^ 2;
  miPu = mdsPu * cos2Eta + mqsPu * sin2Eta;
  MdSatPPu = miPu + MsalPu * sin2Eta;
  MqSatPPu = miPu - MsalPu * cos2Eta;

annotation( ...);
          :orSynchronousThreePhase;
```

Test case: CIGRE HV transmission benchmark system

Initialization

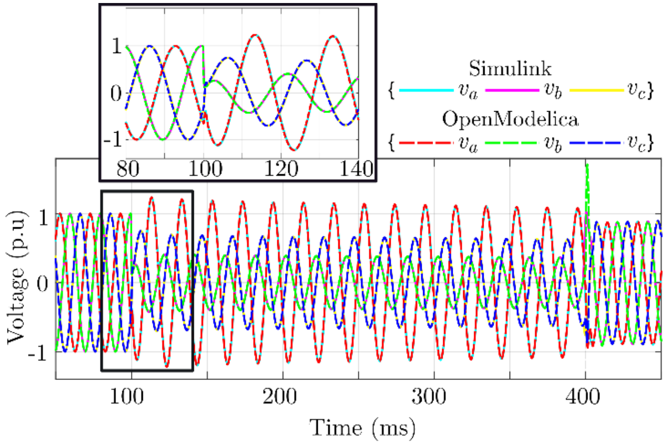
- Initial steady-state (or sufficiently approximate) values need to be provided to make initialization stable.
- Since there is index reduction going on, all of the states in the system must be initialized adequately.
- Dynamic power grid models require the results of a load-flow to set up initial values
- As OpenModelica lacks a power flow computation tool → Simulink's load flow values entered manually

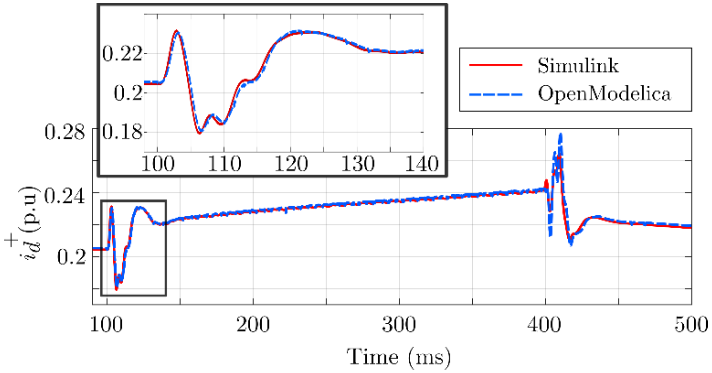
Providing load flow init values → steady-state at 50 s
Without providing any initialization scheme → simulation fails at initialization

Test case: CIGRE HV transmission benchmark system

Comparison methodology

- Four events simulated to study the system behaviour
- Analysis of the interaction between the power converter and the grid.
- Results obtained in OM compared against same model in Simulink.
- Simulations run on a desktop with AMD Ryzen Threadripper 2950X 16-Core 3.50 GHz Processor and 32 GB of RAM under Windows 10 64-bit.





OpenModelica vs Simulink simulation performance

- Three-phase fault at bus 6 – 50/50.3 s, tStop = 70 s.
- Integration methods used in OpenModelica and Simulink are essentially different → fixed and variable time step.
- If using variable-time step solver in Simulink based on BDF ➡ OM outperforms Simulink with the ratio of **5:1** (OM system with disconnected VSC)

Characteristics	**OM system with connected VSC**	**OM system with disconnected VSC**	**Simulink system**
nb variables	4320	3128	-
Solver	DASSL (order 5)	DASSL (order 5)	Backward Euler
Tolerance	1e-4	1e-4	-
Δt max	25 µs	25 µs	25 µs
Δt min	0.12 µs	0.13 µs	25 µs
Steps taken	39,056,194	778,739	-
ODE function calls	73,551,404	1,537,703	-
J-evaluations	14,188,205	183,749	-
J-evaluation time (s)	11,428	131	-
Total simulation time (s)	25,449	992	2133

Costly Jacobian evaluation

Due to oscillations in VSC DSRF control

OpenModelica outperforms Simulink with the ratio of **2.15:1**

OM 1.18.0 vs OM 1.18.1 simulation performance

- Three-phase fault at bus 6 – 50/50.3 s, tStop = 70 s.

Characteristics	**OM 1.17**	**OM 1.18**	**Simulink system**
nb variables	4320	3128	-
Solver	DASSL (order 5)	DASSL (order 5)	Backward Euler
Tolerance	1e-4	1e-4	-
Δt max	25 µs	25 µs	25 µs
Δt min	0.12 µs	0.112 µs	25 µs
Steps taken	39,056,194	33,371,799	-
ODE function calls	73,551,404	62,970,161	-
J-evaluations	14,188,205	12,170,676	-
J-evaluation time (s)	11,428	9,721	-
Total simulation time (s)	25,449	20,537	2133

20% speed up

Experience in using OM for power systems with power converters

- ✓ OpenModelica has demonstrated an excellent potential for EMT-type modelling and simulation for future power systems.
- ✓ Modelica implementation of such models is easy and straightforward due to the native properties of the language: declarative and equation-based.
- ✓ Decoupling the models from numerical solvers offers outstanding flexibility compared to Simulink.
- ✓ Results confirmed a notable overall agreement between OM and Simulink
- ❖ In general terms, OM performance for the studied system is not satisfactory compared to Simulink.
- ❖ The lack of a load flow computation tool is a major barrier for power system simulation.

Conclusions and future work

- OpenModelica is a promising environment for power system simulation

⇨ **Collaboration, flexibility, transparency and interoperability**

- Remarkable opportunities afforded by this new approach to software development

- Future works include:
  - Testing larger transmission power networks
  - Implementing initialization routines to avoid explicitly entered load flow data
  - Analyse systems with a higher share of RES.
  - Linearization capabilities for converter control structures

Q & A