A Library to Support Learning Power Systems Modeling with OpenModelica and OMEdit

Francesco Casella  
(francesco.casella@polimi.it)  

Adrien Guironnet  
(adrien.guironnet@rte-france.com)
Scope of the PowerGrids Library

• Modelling of power transmission and distribution systems
• Scale: from small academic examples to full pan-european models
• Quasi-static E/M behaviour of transmission lines → phasors
• Balanced 3-phase systems
Scope of the PowerGrids Library

• Modelling of power transmission and distribution systems
• Scale: from small academic examples to full pan-european models
• Quasi-static E/M behaviour of transmission lines → phasors
• Balanced 3-phase systems
• Dynamic phenomena: 0.1 to 10 s
  – Inertia of rotating synchronous generators
  – Internal electrical dynamics of synchronous generators
  – Governors, AVR's, PSSs
  – Islanding transients
  – Always close to nominal (50/60 Hz) frequency
Scope of the PowerGrids Library

- Modelling of power transmission and distribution systems
- Scale: from small academic examples to full pan-european models
- Quasi-static E/M behaviour of transmission lines → phasors
- Balanced 3-phase systems
- Dynamic phenomena: 0.1 to 10 s
  - Inertia of rotating synchronous generators
  - Internal electrical dynamics of synchronous generators
  - Governors, AVRPs, PSSs
  - Islanding transients
  - Always close to nominal (50/60 Hz) frequency
- Full open-source paradigm
  - Modelica language
  - Open source tools (although commercial ones are also fine)
  - Open source solvers
  - Full access to all the details, no hidden/secret/proprietary
Typical Design Rationale of Modelica Libraries

- Models are written by seasoned Modelica experts, with many years experience
- Abstraction, replaceable classes, multiple inheritance are used cleverly
  - to achieve generality
  - to avoid repetitions
  - to minimize the Modelica source code size
Typical Design Rationale of Modelica Libraries

• Models are written by seasoned Modelica experts, with many years experience
• Abstraction, replaceable classes, multiple inheritance are used cleverly
  – to achieve generality
  – to avoid repetitions
  – to minimize the Modelica source code size
• End users are normally expected to use the models via a GUI, not to interact with the model at the source code level.
• Example: PowerSystems library
Typical Design Rationale of Modelica Libraries

• Models are written by seasoned Modelica experts, with many years experience
• Abstraction, replaceable classes, multiple inheritance are used cleverly
  – to achieve generality
  – to avoid repetitions
  – to minimize the Modelica source code size
• End users are normally expected to use the models via a GUI, not to interact with the model at the source code level.
• Example: PowerSystems library

Libraries are easy to use
Extensive Modelica training to adapt existing models or develop new ones
Design Rationale of the PowerGrids library – I

• Power systems engineers are fairly conservative, compared to other sectors of engineering

• They are used to very powerful and mature commercial domain-specific simulation tools
Design Rationale of the PowerGrids library – I

• Power systems engineers are fairly conservative, compared to other sectors of engineering

• They are used to very powerful and mature commercial domain-specific simulation tools

• In some cases, they are used to write their own simulation software
  – Procedural approach
  – Fortran/C/C++ code
  – No model-solver separation
  – Very hard work, mostly going into low-level details
  – Very difficult to understand, develop, and maintain
Design Rationale of the PowerGrids library – I

• Power systems engineers are fairly conservative, compared to other sectors of engineering

• They are used to very powerful and mature commercial domain-specific simulation tools

• In some cases, they are used to write their own simulation software
  – Procedural approach
  – Fortran/C/C++ code
  – No model-solver separation
  – Very hard work, mostly going into low-level details
  – Very difficult to understand, develop, and maintain

• Model/solver separation is often a cultural shock

• A few dedicated Modelica enthusiasts are found, but selling Modelica to the entire community is a very difficult task

• Usually, people have more important things to do than learning Modelica!
Declarative modelling makes life easier

It allows to broaden the scope of modelling beyond state-of-the-art tools

Models of innovative equipment
Multi-domain modelling possible
Declarative modelling makes life easier

It allows to broaden the scope of modelling beyond state-of-the-art tools

Models of innovative equipment
Multi-domain modelling possible

Modelica should be used to make source code easy to understand, develop, and maintain

Keep the learning curve for domain experts as low and smooth as possible
Use the power of Modelica to make the source code easier to understand not to make it arcane or obscure!
Library design: Quantities

• Use Complex variables for phasors
  – The original equations are written using complex numbers
  – Modelica tools should handle them with zero performance penalty!

• Use SI units for connectors and basic physical models
  – Scaling performed automatically via \textit{nominal} attribute
  – Avoid the confusion of having p.u. vars with multiple base quantities
  – Use p.u. \textit{locally} when textbook equations also do for better clarity (e.g. synchronous machine models)
Library Design: Ports and Base Classes

• PortAC: basic object bound to connector current & voltage
  – Contains start values of voltage and P/Q from power flow
  – Defines local base quantities, p.u. quantities, and auxiliary variables
  – Defined once and for all, used by all models consistently
Library Design: Ports and Base Classes

• PortAC: basic object bound to connector current & voltage
  – Contains start values of voltage and P/Q from power flow
  – Defines local base quantities, p.u. quantities, and auxiliary variables
  – Defined once and for all, used by all models consistently

• Base Classes:
  – OnePortAC (generators, loads)
  – OnePortACdqPU (includes Park transformation and per-uniting)
  – TwoPortAC (transmission lines, transformers, phase shifters)

• All low-level details (scaling, initialization, definitions of commonly used variables, etc.) are handled by the base classes, designed by Modelica experts
Library Design: Ports and Base Classes

- **PortAC**: basic object bound to connector current & voltage
  - Contains start values of voltage and P/Q from power flow
  - Defines local base quantities, p.u. quantities, and auxiliary variables
  - Defined once and for all, used by all models consistently

- **Base Classes**:
  - OnePortAC (generators, loads)
  - OnePortACdqPU (includes Park transformation and per-uniting)
  - TwoPortAC (transmission lines, transformers, phase shifters)

- All low-level details (scaling, initialization, definitions of commonly used variables, etc.) are handled by the base classes, designed by Modelica experts

Domain experts can focus on high-level equation-based modelling with minimal effort
Live Demo with OMEdit

https://www.github.com/powergrids
Tutorial on PowerGrids with OMEdit

Tomorrow morning @MODPROD Workshop
Solver Issue: Initialization - I

- Steady-state initialization is required
  - Nonlinear equations involved
  - Convergence is potentially problematic
Solver Issue: Initialization - I

• Steady-state initialization is required
  – Nonlinear equations involved
  – Convergence is potentially problematic

• Theoretical foundation (Casella, Bachmann 2019, submitted to AMC)
  – Only variables influencing the Jacobian of the initialization problem need to be given a good initial guess
  – Other variables get the same value after the first Newton iteration
Solver Issue: Initialization - I

• Steady-state initialization is required
  – Nonlinear equations involved
  – Convergence is potentially problematic

• Theoretical foundation (Casella, Bachmann 2019, submitted to AMC)
  – Only variables influencing the Jacobian of the initialization problem need to be given a good initial guess
  – Other variables get the same value after the first Newton iteration

• Library design: all components with nonlinear equations have
  – Parameters to set port values obtained from the power flow problem
  – Binding/initial equations to compute all other required start values
Solver Issue: Initialization - I

• Steady-state initialization is required
  – Nonlinear equations involved
  – Convergence is potentially problematic

• Theoretical foundation (Casella, Bachmann 2019, submitted to AMC)
  – Only variables influencing the Jacobian of the initialization problem need to be given a good initial guess
  – Other variables get the same value after the first Newton iteration

• Library design: all components with nonlinear equations have
  – Parameters to set port values obtained from the power flow problem
  – Binding/initial equations to compute all other required start values

Guarantee of convergence once power flow solution is known
Solver Issue: Initialization - II

• The convergence can be destroyed by tearing!
  – Linear variable is selected as tearing variable
  – It has no meaningful value
  – Some nonlinear variables are computed in the torn section as a function of it
Solver Issue: Initialization - II

• The convergence can be destroyed by tearing!
  – Linear variable is selected as tearing variable
  – It has no meaningful value
  – Some nonlinear variables are computed in the torn section as a function of it

BOOM!
Solver Issue: Initialization - II

- The convergence can be destroyed by tearing!
  - Linear variable is selected as tearing variable
  - It has no meaningful value
  - Some nonlinear variables are computed in the torn section as a function of it

- Solution 1: No tearing at all (+ sparse solver)
- Solution 2: “Smart” tearing
  - Take into account indirect influence of tearing variables on torn variables
  - Avoid the loss of strategic nonlinear variable start values
Solver Issue: Initialization - II

- The convergence can be destroyed by tearing!
  - Linear variable is selected as tearing variable
  - It has no meaningful value
  - Some nonlinear variables are computed in the torn section as a function of it

- Solution 1: No tearing at all (+ sparse solver)
- Solution 2: “Smart” tearing
  - Take into account indirect influence of tearing variables on torn variables
  - Avoid the loss of strategic nonlinear variable start values

- Open problems
  - How to make sure the correct solver setup is automatically obtained?
  - Are new standarized annotations required?
Solver Issue: DAE mode

- The DAEs describing power systems with phasors are sparse (local connections)
- The corresponding ODEs instead are dense (acceleration of each generator instantaneously depends on the angle of all other generators)
- Causalization and solution with an ODE solver is not a good idea
Solver Issue: DAE mode

- The DAEs describing power systems with phasors are sparse (local connections)
- The corresponding ODEs instead are dense (acceleration of each generator instantaneously depends on the angle of all other generators)
- Causalization and solution with an ODE solver is not a good idea

DAE mode should be automatically selected when this structure is detected
Solver Issue: DAE mode

- The DAEs describing power systems with phasors are sparse (local connections)
- The corresponding ODEs instead are dense (acceleration of each generator instantaneously depends on the angle of all other generators)
- Causalization and solution with an ODE solver is not a good idea

DAE mode should be automatically selected when this structure is detected

Efficient event detection and handling (currently based on causalized equations) needs new research
Conclusions

• The PowerGrids library allows to perform phasor-based power system simulation using OMC-OMEdit

• The toolchain is complete, including power flow and graphical editor and is 100% open source free software
Conclusions

• The PowerGrids library allows to perform phasor-based power system simulation using OMC-OMEdit

• The toolchain is complete, including power flow and graphical editor and is 100% open source free software

• Excellent for teaching purposes

• The source code of component models is easily understood and written also by Modelica novices
Conclusions

• The PowerGrids library allows to perform phasor-based power system simulation using OMC-OMEdit

• The toolchain is complete, including power flow and graphical editor and is 100% open source free software

• Excellent for teaching purposes

• The source code of component models is easily understood and written also by Modelica novices

• Could also be expanded for serious use on large-scale systems when better support for such systems is provided by OpenModelica
Thank you for your kind attention!