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

1 Introduction

Power systems are undergoing major changes due to the increasing penetration of renewable energies, to the booming development of high-voltage direct current lines and to the difficulty to build new AC lines, that lead power system operators to design complex automata to fully use the grid potential. All these changes are deeply affecting the power system dynamics that are evolving from a well-known behaviour, mainly driven by the synchronous generator dynamics, to a more uncertain and complex behaviour driven by a combination of power-electronic based components without inertia, complex system-wide controls and traditional machines.

In this context, the Transmission System Operators (TSO) ability to assess the system stability is questioned and recent events have shown the need for a new frame for time-domain stability studies (ENTSO-E SG SPD Report, a; ENTSO-E System Protection and Dynamics WG Report; Yan et al., 2018). In order to accompany this deep and global change in a secure way, power system actors have to take adapted and coordinated decisions. This demands a transition from current closed tools towards flexible and transparent power system simulation approaches, enabling all players to run collaborative studies in a straightforward and easy way.

Despite previous efforts led by the European Network of Transmission System Operators for Electricity (ENTSO-E) association with the development of a standard exchange format - Common Grid Model Exchange Standard (CGMES) - different software tools still give different results for the same data set, even for small networks (ENTSO-E SG SPD Report, b). Without having access neither to the final modelling implementation nor to the mathematical methods used for the solution, it remains very arduous to understand the reasons of the results difference, and thus to agree upon common actions at the pan-european level. Having open-source shared models will be a first step towards a better understanding of the overall power system behaviour and ease the power system actors cooperation.

Modelica appears as a good candidate to build such models and initiatives at the European level are currently conducted to promote this vision – the necessity to construct and share a reference Modelica library – using the results from several previous efforts (Winkler, 2017). Indeed, the European projects Pegase and iTesla have respectively validated the usability of Modelica for power system modelling on elementary components (Pegase; Chieh et al., 2011) and proved the possibility to get results from Modelica models that are similar to those obtained with existing power system simulation software (iTesla; Bogodorova et al., 2013; León et al., 2015). Due to their intended goal, the library inherited from these projects consists mainly in a direct porting of several tools models that stick to a procedural way of modelling (Fortran-style code), which is difficult to interpret, extend, and maintain.

Two open-source libraries have also been developed in the Modelica community for electromechanical power system modelling: one is the Modelica.Electrical.QuasiStationary library, which uses complex phasors and contains only very basic component models, thus not specifically designed for large scale power system models. The other one is the PowerSystems library, whose design is fundamentally based on the concept of replaceable phase system, which can have an arbitrary number of independent voltage and current components, and an arbitrary number of reference phases. The very high level of abstraction, coupled with an

extensive use of inheritance, allows to accommodate a very wide scope in a unified framework spanning DC, one- and three-phase AC in both quasi-static and dynamic regimes. The end result is a code base which is very general, elegant and concise; however, the price to pay for these features is that the code is quite difficult to comprehend for non-Modelica experts. For a more detailed review of open-source Modelica libraries in this field, the interested reader is referred to (Winkler, 2017).

The authors are also aware of the existence of commercial Modelica libraries for power system, and also worked on internal developments in this field that were mainly intended for experimentation purposes, see, e.g., (Casella et al., 2016). However, they won't be discussed here, since a key requirement of the next-generation pan-european power system modelling tool is the accessibility and open-source nature of the modelling code.

All the previously mentioned works contribute to demonstrate the added value of using Modelica for power systems models, but a more generic and high-level reflection, especially on the modelling side, is required to ensure the widest possible adoption of Modelica by the power system actors, i.e. Transmission System and Distribution System Operators. This is the main objective of the prototype library introduced in this paper and could be achieved by relying on the following library features:

- the library architecture was carefully designed to take full advantage of the declarative modelling approach of Modelica, which allows to write models that are close to their textbook descriptions, without at the same time being too difficult to understand for Modelica beginners because of the extensive use of advanced object-oriented features;
- specific power system models have been written in order to be easily understood, reused and potentially extended by power system experts without much experience in the Modelica language, rather than by expert Modelica developers;
- the library has been designed to eventually evolve into an industrial-grade tool.

The paper is organized as follows. Section 2 defines the library scope while section 3 states the necessary requirements to embrace this scope. In section 4, the architecture and design of the PowerGrids library are discussed, showing how they can fulfil the requirements. Section 5 is a presentation of the prototype library models and their validation, both at an individual level and at a system level, is detailed in section 6. Section 7 is devoted to conclusions and future perspectives of this work.

2 Scope

The scope of the library described in this paper is to build electro-mechanical power transmission and distribution system models, possibly spanning entire countries or even the whole pan-european system, though of course smaller system models for regional studies or for didactic purposes should also be covered.

The electro-magnetic behaviour of transmission lines, transformers and loads is assumed to always be close to sinusoidal steady-state, so that the relationships between three-phase voltage and current systems can be represented by phasors, while fast electro-magnetic transients in those components involving current and voltage state variables are neglected.

The library described in this paper currently only considers balanced three-phase systems, though it would be possible to extend the approach to deal with unbalanced systems by introducing the direct, inverse and zero sequence representation. The system dynamic behaviour is generated by the mechanical inertia of rotating synchronous generators and by their internal electromagnetic transients, represented by lumped-parameter models, which have much larger time constants than the transmission lines, in the range 0.1-10 s, as well as by the dynamics of all continuous-time and event-based control systems.

The library should allow to efficiently simulate dynamic islanding transients, whereby a synchronous system can be split into two or more synchronous islands by the opening of some circuit breakers, which could operate at significantly different frequency for significant amounts of time, and possibly be re-synchronized afterwards.

The system can be assumed to always operate at frequencies close to the reference value (50 Hz for Europe), since power systems cannot operate reliably when the system frequency deviates by more than a few Hz from the reference. Simulations need to be reliably initialized at steady-state or sufficiently close to it also in the case of very large-scale systems.

Last, but not least, the models from this library are expected to be used in two ways. One is to use them to build complete power system models in Modelica, using Modelica tools to turn them into simulation code. However, even though encouraging experiments have been carried out using Modelica tools to handle system models with thousands of generators, lines, and loads, as reported in (Casella et al., 2017), current Modelica compiler technology, based on the expansion of models down to individual scalar equations, still cannot handle realistic pan-european grid models within acceptable limits in terms of code generation time (a few minutes at most) and of executable simulation code size (a few MBytes).

A major breakthrough is thus needed in the Modelica compiler technology to avoid the burden of code duplication relative to components that are instantiated hundreds or thousands of times in the system model. In the meantime, it should be possible to also use models from the library within domain-specific simulation tools, such as RTE's Dynaωo (Guironnet et al., 2018), that require only one instance of C-code for each model appearing multiple times in the system, build the residuals and Jacobians for

DOI 10.3384/ecp19157627

the DAE solver using said C-code and ad-hoc algorithms, and finally use DAE open-source solvers to compute the simulation results.

3 Requirements

The modelling approach for the physical components represented in the library should be fully declarative and equation-based, with no compromises. The code describing the physical behaviour of components should be as close as possible to the original formulation of the equations that are found in textbooks or technical reports. This allows to achieve four important objectives:

1. the models are largely self-documenting and can easily be traced back to authoritative sources;

2. it is immediately clear to a domain expert by just reading the code what a given model actually represents and what the modelling assumptions are;

3. it is easy to turn any new component model who is devised by experts in terms of basic physical equations on the paper into the model code;

4. it is easy to adapt or customize existing models to fit new and possibly unexpected simulation scenarios in the future.

The model-solver separation principle should always guide the code development. Models should be written to be clear, compact, elegant and easily understood, without any need to explicitly structure them in a way which is oriented to their solution, as it is common practice in traditional Fortran, Matlab, C, or C++ based models.

Basic data types, model building blocks and base classes should be readily available in the library, to guide modellers that are not Modelica experts in the development of new models that fully exploit the power of the Modelica language, avoiding them the hassle of taking care of the tedious and repetitive parts of the modelling effort, and allowing them to focus on the core parts of their models.

For controllers such as Automatic Voltage Regulators (AVR), governors or Power System Stabilizers (PSS), which are usually specified in terms of block diagrams, the same representation should be used in the library, so that again the model is as close as possible to the original source (e.g. block diagrams taken from IEEE standards or recommendations), self-documented and immediately recognizable by a domain expert.

On the other hand, an essential point is that the actual implementation of blocks, either in terms of equations or of lower-level block diagrams, must be fully accessible, so that the exact behaviour of the each block is clearly and unambiguously defined. This is often a weak point in closed-source simulation tools, in which the behaviour of some non-trivial blocks such as anti-windup controllers, that can actually be implemented in subtly different ways, may be different from one tool to the other, leading to different behaviour of the same system model depending on which tool it is run (ENTSO-E SG SPD Report, b).

It is assumed that the results of a power-flow calculation are available, specifying the voltage modulus and phase and the entering active and reactive power flow at each three-phase port of each component. These could be computed by a separate tool, or possibly by a corresponding Modelica power-flow model, see section 5.5.

Last, but not least, it is important to point out who is going to write the code of the more sophisticated component models. Most Modelica libraries are written by Modelica experts and are expected to be used more or less out of the box by domain experts and practitioners. In the case of the library discussed in this paper, instead, one fundamental goal is to develop an open library that is readily accepted by the community of transmission and distribution systems operators, as well as by students in this field, also for developing new models. This requires the existing source code to be easily read and understood, and new models to be easily developed or adapted, by people who are domain experts but are not seasoned Modelica library developers, having had only some basic training in Modelica.

The most commonly used language in this area are Fortran, C/C++ and possibly Matlab or Python, and people are normally used to a procedural approach to modelling, so the shift to the a-causal, declarative approach of Modelica already requires a significant effort to become second nature to the modeller. From this point of view, providing basic data types, objects and templates (base classes), as well as some fully worked out reference model implementations can be very useful to make the learning curve less steep.

Most importantly, advanced Modelica features should then be used judiciously as long as they can actually make reading and writing the code easier, by using basic types and classes which are already defined in the library. On the other hand, very elaborate inheritance-based library architectures, possibly involving replaceable classes and multiple inheritance, should rather be avoided, as they could make understanding the code difficult for people without a lot of experience and practice in using Modelica, ending up in a steep barrier to the acceptance of the library, and ultimately hindering its diffusion in the potential user community.

4 Design of the PowerGrids Library

In this section, the basic architecture of the PowerGrids library is presented in a bottom-up fashion, showing all the basic data types and base classes that can be used to described actual models in a very compact and clear way, as will be demonstrated in Section 5.

4.1 Complex Variables

Following the declarative modelling paradigm, phasors should be represented by complex variables, in order to

have a compact and immediately understood formulation of equations, as in textbooks. Complex numbers are available in Modelica since 2013 and should be used to their full extent. The explicit expansion of equations into their real and imaginary parts is tedious, ugly and error prone, makes the model unnecessarily obscure, and should thus be left to the Modelica compiler, not to the modeller.

In fact, Modelica tools still turn out sometimes to be a bit crude in handling Complex numbers in the most efficient way, but this should by no means be a reason to work around these tool limitations by expanding models to scalar values manually. To the contrary, this should be a good reason to push Modelica tool vendors to improve their handling so that there is no performance penalty whatsoever when they are employed instead of writing manually expanded equations using Real numbers.

4.2 Units and Scaling

Physical variables, in particular connector variables, are always defined given in SI units. This guarantees consistency at system and model level, avoids the need of introducing conversion factors in physical equations, and also allows to use unit checking to spot modelling errors resulting in dimensionally inconsistent equations. As it is the standard practice, engineering units can be used for convenience in the user interface, for parameter input and for plotting, by setting the `displayUnit` attributes according to the typical values found in high-voltage transmission systems, i.e., kV for voltage, kA for current, MW for active power, MVA for complex apparent power and MVAR for reactive power.

Use of per-unit variables is restricted to those cases in which their use makes the equations more compact and thus easier to write and understand, such as models of synchronous machines. In this case, the reference textbooks use per-unit quantities when writing the equations, so the application of the principle that the Modelica code should be as close as possible to the textbook equation suggests to also write the Modelica code in the same way, using component-specific (not system-wide) base quantities declared as parameters.

If model equations using SI units were solved directly, the numerical accuracy of the solution would be severely hampered because of badly scaled problems, since the typical model will contain some variables with order of magnitude 1 (the p.u. variables), some others (currents and voltage variables) around 10^3–10^4, and some others (power variables) in the range 10^8–10^9. In fact, one of the reasons why p.u. variables and engineering units are used explicitly in traditional power system simulation software is to ensure the good numerical conditioning of the problem. However, this can make the code a bit confusing and possibly lead to modelling mistakes, due to mix-up of per-unit variables and parameters using different base quantities (system-wide and component-specific) with variables and parameters in engineering units.

The proper way to address this problem in a declarative way in Modelica is to avoid introducing explicitly the scaled variables in the model, to define the nominal power and voltage as parameters of each component, and then to set the `nominal` attribute for all physical variables to a value that can be derived from them, e.g.

```
Modelica.SIunits.Current I(nominal = SNom/VNom);
```

When generating the simulation code, the Modelica tool uses the `nominal` attribute as a scaling factor for the variables, and is also able to automatically derive a scaling factor for equation residuals using the nominal values of the variables and the values of the Jacobian, see (Casella and Braun, 2017) for further details. The end result is equivalent to the use of per-unit variables of traditional power system simulation codes. However, according to the model-solver separation principle, this process is totally transparent to the modeller.

4.3 Connectors and System Object

AC components in power trasmission and distribution systems interact through 3-phase connections, which are assumed to be balanced in the context of this work, since this is the most commonly used assumption for large-scale system studies. The efficient simulation of such systems when they are close to sinusoidal steady-state requires a representation of 3-phase systems leading to almost constant variables in that case.

To achieve this objective, both the Modelica.Electrical.QuasiStationary and the PowerGrids libraries represent 3-phase systems relative to a rotating frame of reference, whose angle is defined by a source component and propagated through connectors to all the components of a synchronous system; the former library uses complex phasors, while the latter uses a dq0 decomposition. In order to handle this properly, overconstrained connectors need to be used, whereby the Modelica tool analyzes the connection graph and automatically removes the redundant equations involving the reference angle that are generated when meshed grids are built.

This design is very elegant and fully object-oriented, but as of Modelica 3.4 it has the fundamental limitation that the topology of the graph is statically determined at compile time and cannot be changed at runtime. It is thus possible to model systems containing multiple synchronous systems (e.g. two AC grids connected by an HVDC line with AC/DC interfaces), but it is not possible to efficiently model systems that are split in two or more disconnected synchronous islands during a simulation, e.g. because of the opening of circuit breakers that connect different areas in the system, as well as their re-synchronization and re-connection. As the scope of the library discussed in this paper should include this kind of simulations, it is not possible to use connectors to propagate the reference frame information.

Therefore, the AC connector in the PowerGrids library is defined as

DOI 10.3384/ecp19157627

```
connector TerminalAC
  Types.ComplexVoltage v;
  flow Types.ComplexCurrent i;
end TerminalAC;
```

where `v` is the phase-to ground, RMS voltage phasor, and `i` is the line RMS current phasor. Both phasors describe the magnitude and phase of the three-phase balanced system with respect to a reference rotating frame, that needs to be the same for all connectors in each connection set. There are then three possible options regarding reference systems, which are selected in the system object and then accessed via the inner-outer mechanism.

The first option can be chosen when the system is connected to an infinite bus, which prevents the frequency of all voltages and currents to drift significantly away from 50 Hz, lest synchronism is lost. In this case, all components use the same reference, which rotates at the fixed frequency of 50 Hz (or 60 Hz for US and Japan, the reference frequency being defined in the system object).

The second option can be chosen in the case of a single system of components operating in synchronism, except for possible electro-mechanical swings, whose frequency can drift away from 50 (or 60) Hz for long intervals, e.g., if secondary frequency control is lacking or not effective enough for some reason. In this case the frequency output of one component, usually the angular frequency of a large synchronous machine rotor in the system, is connected to the reference frequency input of the system object, from which all other system components access it via the inner/outer mechanism. System models with multiple synchronous islands (e.g. two AC grids connected by an HVDC line) can be modelled by including each island in a sub-model with its own system object and an outer connector, which will then in turn be connected together with the HVDC line.

The third option allow to handle dynamically splitting and re-synchronizing synchronous sub-islands in the system efficiently. In this case, the system object contains a description of the grid topology (nodes and branches), it receives the connection status of all branch components (lines, transformers) in the grid by means of boolean inputs, and outputs the reference frequency and activation status to all nodes in the system (machines and loads).

During initialization, and every time one such input changes, e.g. because of a line trip, the system object runs a topological analysis on the grid, determines what are the synchronously operating islands, selects one node of each island as the source of the reference frequency following some criterion, and outputs the reference frequency values to all the nodes of the island accordingly. If newly formed islands end up operating steadily at different frequency, all their phasors will be nearly constant, speeding up variable step-size solvers; this would not be the case if the second option was selected, because only the phasors of the island containing the single reference generator would be constant, while all others would end up rotating at constant, non-zero speed.

In case islands are formed that would break the simulation, e.g. because they have loads but no generators, the activation signals of all the nodes become false, so that the corresponding models can be switched to an "off state", e.g. zero current for loads, zero power for generators.

The third option is currently not yet implemented in the library, but is made possible by the library architecture; it was successfully experimented within the feasibility study reported by (Casella et al., 2017). A proper implementation is not trivial, because it needs to include efficient algorithms for topology analysis, that will be run online each time an event corresponding to a breaker opening or closing is triggered. Such algorithms should be implemented efficiently as external C functions, possibly porting them from the code-base of existing power system simulation tools.

Setting up a model to work with the third option requires a fairly large amount of signal connections between node and branch components of the grid and the system object, which is not really consistent with a fully object-oriented modelling approach. On the other hand, large grid models will most likely not be built manually using a GUI, but rather generated automatically from grid description such as the XML-based CIM standard (R.Viruez et al., 2017), so this is not necessarily a big issue. It would be in fact be interesting to extend the overconstrained connector semantics of Modelica to also handle run-time topological changes of the connection graph, in order to pass the reference phase/frequency information through the connectors in fully object-oriented way, but relying on this mechanism would make the library described in this paper depending on a non-standard experimental feature of the language for a very long time, which is out of question. The discussion of such a mechanism for future improvements of Modelica would be nevertheless interesting, but goes beyond the scope of this paper.

DC connections, as will be needed for HVDC links, can be represented by using the standard connectors of the Modelica.Electrical.Analog library.

4.4 Ports and Common Base Classes

AC components are connected via `TerminalAC` connectors, which contain the minimum amount of independent quantities needed to represent the physical interaction of two or more components by means of connection equations, namely, the phase-to-ground and line current RMS phasors. In fact, there are many other related variables that could be used both for modelling and monitoring purposes: the phase-to-phase voltage phasor, the per-unit voltage and current phasors, the voltage and current modulus and phase angle, the active, reactive, and complex power flowing through the connector, etc.

All these quantities and the equations relating them to the phase-to-ground voltage and line current phasors are thus defined once and for all in the `PortAC` model, so they don't need to be re-defined every time a new component model is developed. In fact, since some of these

```
partial model OnePortAC
  parameter Types.Voltage UNom;
  parameter Types.ApparentPower SNom;
  parameter Boolean portVariablesPu = true;
  parameter Boolean portVariablesPhases = false;
  parameter Boolean generatorConvention = false;
  parameter Types.Voltage UStart = UNom;
  parameter Types.Angle UPhaseStart = 0;
  parameter Types.ActivePower PStart = SNom;
  parameter Types.ReactivePower QStart = 0;

  PowerGrids.Interfaces.TerminalAC terminal;
  PortAC port(
    final v = terminal.v, final i = terminal.i,
    final UNom = UNom, final SNom = SNom,
    final portVariablesPu = portVariablesPu,
    ...
    final PStart = PStart,
    final QStart = QStart,
    ...);
  outer Electrical.System system;
end OnePortAC;
```

Figure 1. Code of OnePortAC base class

quantities are not always needed in all models, they are conditionally defined based on boolean parameters. For example, `portVariablesPu` activates the definition of per-unit port variables, as well as the corresponding binding equations.

Power transmission and distribution system models are usually represented by single-line diagrams, whereby components are only connected to the ground internally where needed (e.g. for shunt admittances in transmission lines), but there is no need to show the connections to ground explicitly at the system level. Hence, each port corresponds to a single connector, not to a pair of connectors, as in the case of DC component models.

It is then possible to define two base classes, one for one-port components such as generators, load, and capacitor banks, that correspond to nodes in single-line connection diagrams, and the other for two port-components such as transformers and transmission lines. These base classes contain the terminal connector(s), the instance(s) of the port(s), and the start values of port voltage phase and angle as well as of the active and reactive power flowing into the port, which will be the basis for initialization, see Section 4.5. Fig. 1 shows the implementation of `OnePortAC`, abridged for conciseness.

All one- and two-port AC components can then be written by extending from these two base classes, directly using port variables such as the active power entering the port `port.P`, the per-unit voltage phasor `port.vPu` or the voltage phase angle `port.UPhase` in the model equations.

It is then possible to define the `OnePortACdqPU` base model, which extends `OnePortAC` by adding the equations that define Park's transformation between voltages and currents in the port rotating frame of reference and the direct and quadrature per-unit voltages `vdPu,vqPu` and currents `idPu,iqPu` in the machine rotating frame of reference. This allows to directly write the machine models using per-unit variables in the rotor frame of reference, as it is normally done in textbooks, without bothering about defining Park's transformation, which is already provided by the base class of the library.

This design is extremely user-friendly even for a novice Modelica developer, because it is very straightforward. It provides many of the variables that are usually involved in power system component models, and ultimately allows to focus on writing the actual model in a declarative way, without wasting time on coding standard and well-known basic definitions and transformations times and again.

4.5 Strategies for Initialization

The traditional strategy for initialization in power system simulation tools is to have dedicated initialization models to calculate initial values for the system states based on boundary values. Indeed, TSOs only have observability on their own network, which means that the initial values available are the bus values - voltage, active and reactive flows into the bus - and not the set point values for the different injections (machines, loads, static VAR compensators, etc.). It is then necessary to derive these initial values from the bus values: the causality of this problem is opposite to the one from the time-domain problem in which the fixed set points enable to calculate the bus values. In current power system software, the approach is to manually derive the sequence of computations required to solve the system for the initial values starting from the boundary values.

This strategy, that was also followed in the design of the iPSL library, has the advantage of splitting the system-wide initialization problem into a large number of small initialization problems, which are solved locally for each component connected to the grid, and is historically proven to be reliable also on very large-scale systems. On the other hand, it follows a traditional procedural approach, which is particularly inconvenient because it basically requires to re-write all the model equations two times, one for the simulation and one for the initialization, the only difference being the causality of the solution.

The PowerGrids library implements two declarative initialization strategies, based on the use of initial equations, that can be selected from the system object. The starting point in both cases are the initial values of the voltage magnitude and angle, active and reactive power at the component ports, see Fig. 1. These values can be the result of a power-flow computation carried out with a specialized tool, or they could be computed running a Modelica power-flow model, see Section 5.5, or they could come from the TSO/DSO online grid monitoring system.

A key provision in both cases is the computation of start values for a subset of variables that show up in a nonlinear fashion in all models, e.g., the machine angle,vwhich is the argument of sine and cosine functions, and the direct and quadrature values of rotor current and voltage,

DOI 10.3384/ecp19157627

which are multiplied together to get the active and reactive power, in synchronous machine models. This is performed in a declarative way by writing as many initial equations as necessary to compute those start values (which are `fixed =false` parameters) from the port start values.

It is important to notice that the number of such equations is a small fraction of the total equations number in the model, and also that it is not necessary to write them down explicitly in the way they will be solved. It is thus easy to check that they are correct by inspection, because they are basically the same that show up in the **`equation`** section, except that they have the start values as unknowns instead of the model variables, and that they lack all derivative terms (quasi-static initialization).

The first strategy replicates the traditional one, albeit in a declarative way. In this case, initial equations fix the currents values that are injected into the grid by the generators, according to the results of the power-flow calculations. The dependency analysis carried out by the Modelica tool on the initialization problem, resulting in the Block-Lower-Triangular (BLT) transformation, determines that, starting from those values, it is possible to solve the network equations, including transformers, transmission lines and loads, to determine the corresponding voltages. The corresponding problem is a very large, sparse system of nonlinear equations; the convergence of the iterative solver is guaranteed by the start values that have previously been computed based on the power flow.

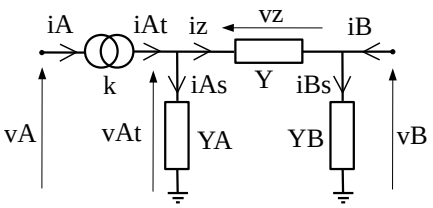
Once that is solved, both currents and voltages are known at the generators' boundaries, so the BLT analysis will automatically determine that the initialization problems of each generator can be solved independently. These will be automatically solved with the opposite causality with respect to the simulation problem, again relying on the computed start values to ensure the convergence of implicit nonlinear equations, ultimately computing all the initial state values, as well as consistent values of the governor and voltage regulators set-points, which are given by `fixed=false` parameters.

The second strategy instead is to directly initialize the whole system in steady-state, by adding initial equations in the generators stating that the derivatives of all internal states are zero. In this case, a very large nonlinear system of equations will emerge from the BLT transformation of the initialization problem, including all the components of the system. Solver convergence can again be facilitated by the start values mentioned previously, together with the use of homotopy to deal with the controller and governor saturations within the generator models, by first solving a problem without controller saturations and then by gradually introducing them, by means of the `homotopy()` operator.

This strategy can be used effectively to ensure a perfect steady-state initialization without the need of any relaxation transient, in case the models used for the power-flow analysis are not exactly consistent with the models used in the Modelica system, e.g. because of simplifications in the loads. It is however hardly feasible in the presence of discrete system-wide automata acting, e.g., on tap changers or phase shifters depending on the values of certain voltages, currents, or power flows, because the solution of the ensuing nonlinear mixed Real-Integer problem could be infeasible.

It is also important to make sure that there are no system-level singularity or ill-conditioning. For example, if the system is not connected to an infinite bus, some primary frequency control needs to be present in the generator models. Otherwise, the steady-state initialization problem will be both over- and under-determined, since on one hand the system frequency will not be well-defined, while on the other hand the excess active power resulting from the mismatch in the balance between the generators' output, the line losses, and the load consumption, will have nowhere to go.

Figure 2. PiNetwork



5 Prototype Library Models

5.1 Transmission Lines and Transformers

The base class `PiNetwork` defines a generic pi-network with one series admittance Y, two shunt admittances at each port Y_A and Y_B, and an ideal transformer with complex tranformatio ratio k at port A, as shown in Fig. 2. It extends `TwoPortAC` by adding the following 7 equations:

```
equation
// Kirchhoff's laws
  iAt = iz + iAs;
  iBs = iz + iB;
  vAt = vz + vB;
// Constitutive equations
  vAt = k * vA;
  iA  = CM.conj(k)*iAt;
  iAs = YA * vAt;
  iBs = YB * vB;
  iz = Y * vz;
```

From this base model, it is possible to derive many different models through inheritance. A transmission line is obtained by making **`final`** `k = 1` and by setting all the admittances to be equal to parameters, which are kept consant through the simulation; this is a commonly adopted assumption, since the system frequency doesn't change much from the reference value. A transmission line with breakers can instead be obtained by keeping Y, Y_A and Y_B as time-varying variables, whose values are determined by **`when`** clauses depending on the status of boolean breaker

opening signals. A fixed ratio transformer is obtained by making Y, Y_A, Y_B, and k equal to parameters, while transformers with tap changers and phase shifters can be modelled by making the real part or the phase of k the result of discrete equations modelling the control logic.

In the latter case, while the description of the physical behaviour is described declaratively in an **equation** section, the control logic is described in an **algorithm** section, which is more suited to describe the control logic and state machine that govern the component's behaviour.

Inheritance is used consistently in this case in order to factor out common features, avoid code duplication and keep the code base lean. However, the design is straightforward also for model developers who are Modelica newbies, since there is only one line of inheritance, whereby each child object adds some specialization to define a more specialized object, whose scope and definition is easy to understand and corresponds to commonly used concepts in power systems.

5.2 Loads

Load models can be implemented in a straighforward way by extending the `OnePortAC` base class and by adding the equations for the active and reactive power flows. For example, the following model defines a voltage-dependent PQ load:

```
U_URef = port.U / URef;
port.P = PRef*U_URef^alpha;
port.Q = QRef*U_URef^ beta;
```

`PRef` and `QRef` can be determined by parameters, or by time-varying variables equal to an expression that is assigned with a binding equation when instantiating the component, or by an input connector.

5.3 Synchronous Machine

Synchronous machines are a key component of power transmission systems. The PowerGrids library contains a full-fledged 4-windings machine model, to check how far it is possible to push the declarative approach with non-trivial models. Magnetic saturation was not included for lack of time, but it could be added easily to the model.

The corresponding iPSL library model, a port of the Eurostag machine model, is really difficult to understand: the formulation heavily leans towards the procedural, and would require a lots of extra documentation to explain why each equation is written in the way it is. Considering also the initialization model, which is essential in order to use the model, the model spans over 400 lines of code. Once the original equations have been rewritten and partially solved towards their solution to write the code, reverse-engineering them back to their original formulation is conceptually (as well as practically) impossible, because some information was lost in the process. Therefore, porting the iPSL library model into the PowerGrids model was not really possible.

The right approach to write the model is instead to get back to the original sources, in this case the theory manual of the Eurostag software, which unfortunately lacks some crucial details, and eventually to the classic book (Kundur, 1994). The machine model is built by extending from the `OnePortACdqPU` base class, that pre-defines all the variables down to the per-unit voltages and currents on the rotor direct and quadrature axes, and then by just adding the equations taken from the textbook that define the relationship between currents and magnetic field, the relationship between magnetic field, speed, and voltage, and the mechanical power balance on the machine rotor.

```
equation
// Flux linkages
lambdadPu = (MdPu+LdPu)*idPu + MdPu*ifPu + MdPu*iDPu;
lambdafPu = MdPu*idPu+(MdPu+LfPu+mrcPu)*ifPu+(MdPu+mrcPu)*iDPu;
lambdaDPu = MdPu*idPu+(MdPu+mrcPu)*ifPu+(MdPu+LDPu+mrcPu)*iDPu;
lambdaqPu = (MqPu+LqPu)*iqPu+MqPu*iQ1Pu+MqPu*iQ2Pu;
lambdaQ1Pu = MqPu*iqPu+(MqPu+LQ1Pu)*iQ1Pu +MqPu*iQ2Pu;
lambdaQ2Pu = MqPu*iqPu+MqPu*iQ1Pu+(MqPu+LQ2Pu)*iQ2Pu;
// Equivalent circuit equations in Park's coordinates
if neglectTransformerTerms then
  udPu = raPu*idPu - omegaPu*lambdaqPu;
  uqPu = raPu*iqPu + omegaPu*lambdadPu;
else
  udPu = raPu*idPu-omegaPu*lambdaqPu+der(lambdadPu)/omegaBase;
  uqPu = raPu*iqPu+omegaPu*lambdadPu+der(lambdaqPu)/omegaBase;
end if;
ufPu =  rfPu *ifPu  + der(lambdafPu)/omegaBase;
0    =  rDPu *iDPu  + der(lambdaDPu)/omegaBase;
0    =  rQ1Pu*iQ1Pu + der(lambdaQ1Pu)/omegaBase;
0    =  rQ2Pu*iQ2Pu + der(lambdaQ2Pu)/omegaBase;
// Mechanical equations
der(theta) = (omegaPu - omegaRefPu) * omegaBase;
2*H*der(omegaPu) =
  (CmPu*PNom/SNom-CePu) - DPu*(omegaPu-omegaRefPu);
CePu = lambdaqPu*idPu - lambdadPu*iqPu;
PePu = CePu*omegaPu;
PmPu = CmPu*omegaPu;
omega = omegaPu*omegaBase;
```

Figure 3. Equation section of the synchronous machine model

The ensuing code, reported Fig. 3 is self-documenting: a reader familiar with synchronous machine theory will immediately recognize the model and understand exactly what kind of behaviour it represents.

A further important point is worth discussing. The equations shown in Fig. 3 use physical parameters (inductances, resistances) which are hardly accessible. External parameters that can be experimentally determined are usually given instead: some per-unit inductances and some time constants. External parameters can be computed explicitly from internal, see (Kundur, 1994) with different degrees of approximation, but not the other way round.

One can then extend the internally-parameterized model, make the internal parameters **final** and with `fixed = false`, add the external parameters, and finally add the initial equations relating the external parameters to the internal ones as stated in the textbook. The Modelica tool will then automatically generate the code to solve those equations backwards, computing the internal parameters from the external one.

The iPSL/Eurostag library model, instead, contains procedural code to carry out this operation, which is difficult to understand unless accompanied by extensive documentation, and a lot more difficult and error-prone to write.

It turns out that these equations are easily solved if the approximate relationships are used, but they can give con-

DOI 10.3384/ecp19157627

vergence problem if the more accurate relationships are used. This issue is brilliantly solved by the use of the `homotopy()` operator, whereby the approximate relationships are used for the simplified model and the accurate ones are used for the actual model.

5.4 Controllers

Controllers such as AVRs, PSSs, and governors, are defined in IEEE standards or guidelines, e.g. (IEEE PES-TR1), (IEEE Std 421.5-2016) by means of block diagrams. The `PowerGrids.Controls` library contains the implementation of basic building blocks which are not already available in the Modelica Standard library, either directly by means of equations, or by means of lower-level block diagrams, if this is the way the block behaviour is specified in the original document. The idea as usual is to keep the Modelica representation of component behaviour as close as possible to the original source documents.

A few representative controller models are also implemented from the above-mentioned sources, e.g., the IEEE ST4B Automatic Voltage Regulator, the IEEE PSS2A and PSS2B Power System Stabilizers, and the IEEE TGOV1 turbine governor.

5.5 Power Flow

In case one wants to run a transmission system model without the need of a separate tool to compute the power-flow, one option is to set the power-flow problem up using Modelica, and to solve it with a Modelica tool. The `PowerGrids.Electrical.PowerFlow` package contains models built for this purpose. Some of them, e.g. the transmission line and the infinite bus, extend the regular models by just changing some appropriate default parameter values. Others, like the PV generators, the PQ load, and the slack node, are easily built by extending from `OnePortAC` and simply adding one or two equations to determine the corresponding variables.

This is not meant to be a replacement of existing power-flow tools, particularly for very large-scale models that may include additional outer loops around the inner power flow calculations, such as PV/PQ node switching, distributed slack node or automata simulations. However, the ability of solving smaller power-flow problems, whose results are required to initialize the dynamic models, could be interesting for smaller-size studies, for example by students that do not have access to commercial power flow tools.

6 Model Verification and Validation

Each component of the PowerGrids library has been verified in simple simulations with known analytical solution.

Specifically, the transmission line, transformer, and load models were verified in a number of simple cases with few non-zero parameters, for which the manual computation of voltages and power flows is straightforward.

The synchronous machine model was verified by successfully replicating the results of a worked-out exercise in (Kundur, 1994), checking that all internal parameters are computed correctly, and that the steady-state behaviour corresponds to the values reported in the book. The dynamic behaviour of the synchronous machine was validated in a simple system also including a transformer, a transmission line, and an infinite bus, starting in steady state and applying step changes to the mechanical power input and to the excitation voltage input.

All the above-mentioned test cases were replicated using the iPSL library and collected in the companion library `PowerGridsIPSLValidation`. The results of all tests matched with very good accuracy.

The test system described in (ENTSO-E SG SPD Report, b), which also includes a governor, an automatic voltage reguator, and a power system stabilizer following IEEE standards, was built with the PowerGrids library. All the test results of the report were reproduced satisfactorily.

The testing activity (as well as the library development) was carried out using the OpenModelica tool, using both the standard ODE mode, where the system model is causalized by the tool and then integrated with an ODE solver, and the experimental DAE mode (see (Braun et al., 2017)) that directly solves the DAEs. Note that, although the DAE equations are sparse, the ODE equations are not, due to the instantaneous interaction among all the generators induced by the phasor-based, quasi-static grid model, making DAE mode integration mandatory for systems with more than a few generators.

Finally, the suitability of the library to describe large-scale systems was tested with an ad-hoc scalable test grid model. The model contains a rectangular grid of $N \times M$ nodes, where each node is connected to its four neighbours by equal transmission lines; a synchronous generator is connected via a step-up transformer to each node, and also feeds a local load. The grid is initialized with a trivial power flow, whereby the active and reactive power of each generator is consumed by the local load, so that the grid is unloaded and all the grid nodes have the same voltage.

Tests were carried out successfully with OpenModelica on a Xeon 2650 CPU with 72 GB of RAM, using the IDA DAE solver and the Kinsol sparse nonlinear solver (both using KLU as sparse linear solver), on systems of size up to $N = 64$ and $M = 64$, which correspond to about 4000 nodes, 750.000 differential equations, and one million initial equations. However, the simulation performance above 500 nodes scales quite badly with the size, most likely due to the high number of cache misses caused by the large size of the executable code (over 100 MB). Above this size, Modelica tools capable of exploiting arrays when generating simulation code would be required. Research is currently starting in this area, but no such tool is yet available at the time of this writing.

7 Conclusion and Future Work

This paper lays out the requirements for an electro-mechanical power generation and transmission system

modelling library. Such a library could be used by TSOs and DSOs for daily activity up to and including pan-european network stability assessment.

The prototype library PowerGrids presented in this paper was developed according to these requirements using the full power of the Modelica language, namely equation-based and fully declarative models, model-solver separation, complex variables, SI units and clear textbook references, while keeping the source code of library accessible for non-Modelica experts, which is not the case with power system existing libraries. The authors believe that the approach taken in the design of this library maximizes the likelihood that experts in power transmission systems in both industry and academia make the transition to Modelica for their modelling work, avoiding the need of a too steep learning curve that could hinder it.

This prototype library also offers an option for students and academics to work with the same data and models. It provides them a direct and transparent way to the used equations thus facilitates the potential discussions between modelling expert and helps improving the general quality of power system models available.

The PowerGrids library will serve as a basis for a larger and more consequent work to provide a reference library for electro-mechanical and electro-magnetic grid models which will ease a wider acceptance of Modelica in the power system community, particularly for network stability studies. This library development will hopefully be conducted in a Horizon 2020 European research project currently under preparation.

It is planned to release the PowerGrids library as open source during the year 2019.

References

T. Bogodorova, M. Sabate, G. León, L. Vanfretti, M. Halat, J.-B. Heyberger, and P. Panciatici. A modelica power-system library for phasor time-domain simulation. In *Proc. 4th IEEE PES ISGT Europe*, 2013.

Willi Braun, Francesco Casella, and Bernhard Bachmann. Solving large-scale Modelica models: new approaches and experimental results using OpenModelica. In *Proc. 12th International Modelica Conference*, pages 557–563, Prague, Czech Republic, May 15–17 2017. doi:10.3384/ecp17132557.

Francesco Casella and Willi Braun. On the importance of scaling in equation-based modelling. In *8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2017*, pages 3–7, Wessling, Germany, Dec 1 2017. doi:10.1145/3158191.3158192.

Francesco Casella, Andrea Bartolini, Simone Pasquini, and Luca Bonuglia. Object-oriented modelling and simulation of large-scale electrical power systems using Modelica: a first feasibility study. In *Proceedings of the 42nd Annual Conference of the IEEE Industrial Electronics Society IECON 2016*, pages 0–6, Firenze, Italy, Oct. 24-27 2016. IEEE, IEEE. ISBN 978-1-5090-3474-1.

Francesco Casella, Alberto Leva, and Andrea Bartolini. Simulation of large grids in OpenModelica: reflections and perspectives. In *Proc. 12th International Modelica Conference*, pages 227–233, Prague, Czech Republic, 2017. doi:10.3384/ecp17132227.

A. Chieh, P. Panciatici, and J.Picard. Power system modeling in Modelica for time-domain simulation. In *Proc. PowerTech*. IEEE, June 2011.

ENTSO-E SG SPD Report. Analysis of CE inter-area oscillations of 1st december 2016. Technical report, a. URL `https://www.entsoe.eu/publications/system-operations-reports/`.

ENTSO-E SG SPD Report. Documentation on controller tests in test grid configurations. Technical report, b. URL `https://www.entsoe.eu/publications/system-operations-reports/`.

ENTSO-E System Protection and Dynamics WG Report. Oscillation event on 3 december 2017. Technical report. URL `https://www.entsoe.eu/publications/system-operations-reports/`.

A. Guironnet, M. Saugier, S. Petitrenaud, F. Xavier, and P.Panciatici. Towards an open-source solution using Modelica for time-domain simulation of power systems. In *Proc. 8th IEEE PES ISGT Europe*, Sarajevo, Bosnia and Herzegovina, Oct 21–25 2018.

IEEE PES-TR1. Dynamic models for turbine-governors in power system studies. Technical report, IEEE-PES, 2013.

IEEE Std 421.5-2016. IEEE recommended practice for excitation system models for power system stability studies. Technical report, IEEE, 2016.

iTesla. iTesla: Innovative Tools for Electrical System Security within Large Areas. URL `http://www.itesla-project.eu`.

Prabha Kundur. *Power System Stability and Control*. McGraw-Hill, 1994.

G. León, M. Halat, M. Sabate, J.-B. Heyberger, F.J. Gomez, and L. Vanfretti. Aspects of power system modeling, initialization and simulation using the Modelica language. In *Proc. 2015 IEEE PowerTech*, Eindhoven, The Netherlands, 29 Jun–2 Jul 2015. IEEE.

Pegase. Pegase: Pan european grid advanced simulation and state estimation. URL `http://www.fp7-pegase.com`.

R.Viruez, S. Machado, L.-M. Zamarre no, G. León, F. Beaude, S. Petitrenaud, and J.-B. Heyberger. A modelica-based tool for power system dynamic simulations. In *Proc. 12th International Modelica Conference*, Prague, Czech Republic, May 15–17 2017.

D. Winkler. Electrical power system modelling in Modelica - comparing open-source library options. In *Proc. 58th SIMS*, Reykjavik, Iceland, Sep 25–27 2017.

R. Yan, N. Al-Masood, T. Kumar Saha, F. Bai, and H. Gu. Anatomy of the 2016 south australia blackout: a catastrophic event in a high renewable network. *IEEE Trans. on Power Systems*, 33(5), Sep 2018.