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.

Several open source Modelica libraries exist for the modelling of electrical power systems, each with a different design philosophy. The scope of the library presented in this paper covers electro-mechanical, phasor-based models of power generation and transmission systems, possibly up to the scale of full pan-European grid models.

The target audience of the library are transmission and distribution system operators, as well as scholars and students in this field. The library is not only meant to be used, more or less as a collection of black-box models, to build system models, but rather to be easily accessible at the source code level by domain experts, which normally do not have an extensive Modelica training, so that they can understand what's inside the models and may contribute new ones avoiding too steep a learning curve.

For this purpose, 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.

For example, existing base classes in the library define basic concepts such as 3-phase balanced AC connectors, 3-phase balanced AC ports, components with one or two AC ports, and one-port components with Park transformation to dq-axis per-unit variables, so that one is spared the tedious task of defining such basic quantities and to do so in a way that Modelica tools can exploit to generate efficient code, e.g. by setting the appropriate `start` and `nominal` attributes. One can then write the equations of a synchronous machine model as shown in Fig. 1, which are immediately recognizable by a domain expert. On the other hand, excessive use of abstraction, multiple inheritance, and replaceable classes should rather be avoided, because they can make the code unreadable to people without a deep experience in advanced Modelica code writing.

The design of the library is fully declarative, exploiting features such as nominal attribute for sound numerical scaling of physical variables, Complex numbers, and a-causal modelling.

The paper discusses the design of the library thoroughly, as well as the implementation of a few basic components, such as transmission lines, transformers, loads, and synchronous generators.

The components in the library have been successfully verified against analytical solutions in simple test cases, and validated against the corresponding models of the iPSL Modelica library in a number of test cases.

The library has also been successfully demonstrated to work in scalable test cases up to 4000 nodes and about one million equations using the OpenModelica tool, although the results show that above about 500 nodes a breakthrough in Modelica compiler technology is required, to avoid the performance penalty brought by the very large size of the simulation executable code, which turns out to be above 100 MBytes.

The library is planned to be released as open source during the year 2019.

```
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 1. Equation section of the synchronous machine model