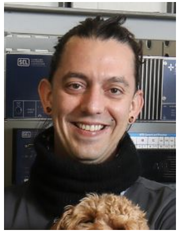
Rensselaer
why not change the world?®
ALSET lab


Marcelo de Castro


Manuel Navarro


Sergio Dorado-Rojas


Luigi Vanfretti


Maxime Baudette

# OpenIPSL - A Modelica Library for Power System Stability Analysis

Marcelo de Castro, Manuel Navarro, Sergio Dorado, Luigi Vanfretti, Maxime Baudette

# Presentation Overview

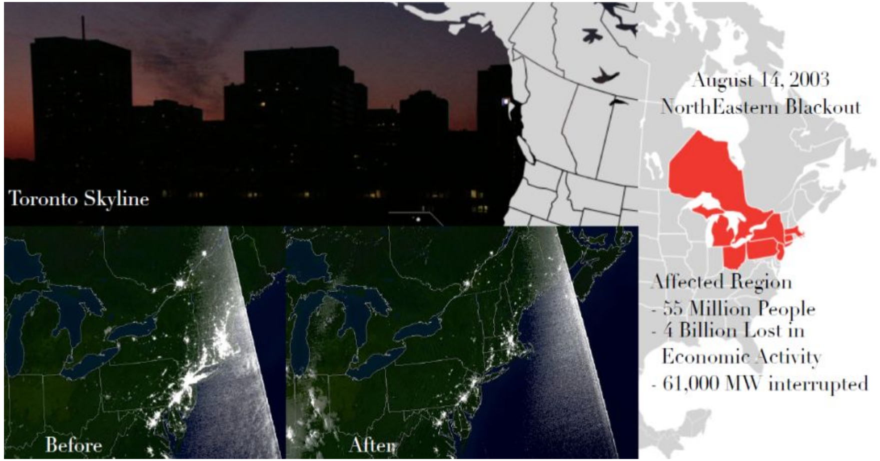
- Models and Simulation in Power Systems
  - The need for models and the different time-scales
- Modelica and Power Systems
- OpenIPSL Library
  - Key features and how it works
- OpenIPSL Applications
  - Simulation of hybrid models (three-phase and single phase)
  - Multi-domain Simulation
  - Training Data Generation for ML-based Application
  - Extremum Seeking Control
- Ongoing Development
  - Continuous Integration and Model Verification
  - Initializing OpenIPSL Models with Python
- Where to find OpenIPSL

# Presentation Overview

- Models and Simulation in Power Systems
  - The need for models and the different time-scales
- Modelica and Power Systems
- OpenIPSL Library
  - Key features and how it works
- OpenIPSL Applications
  - Simulation of hybrid models (three-phase and single phase)
  - Multi-domain Simulation
  - Training Data Generation for ML-based Application
  - Extremum Seeking Control
- Ongoing Development
  - Continuous Integration and Model Verification
  - Initializing OpenIPSL Models with Python
- Where to find OpenIPSL

# The Need for Models in Power Systems

- Models have always been created for power systems.
  - It is a way to study each device and how many devices behave when interconnected.
- Networks increased in size and in complexity of devices.
  - Tools were adapted and enhanced!
- Simulation tools should provide means of anticipate any failures and tips to improve the current power system.
  - Failure to anticipate events may result in huge costs!
- Many examples of such events can be found in the last 20 years:
  - WECC 1996 Break-up, European Blackout (4-Nov.-2006), London (28-Aug-2003), Italy (28-Sep.-2003), Denmark/Sweden (23-Sep.-2003)




**Failure!**

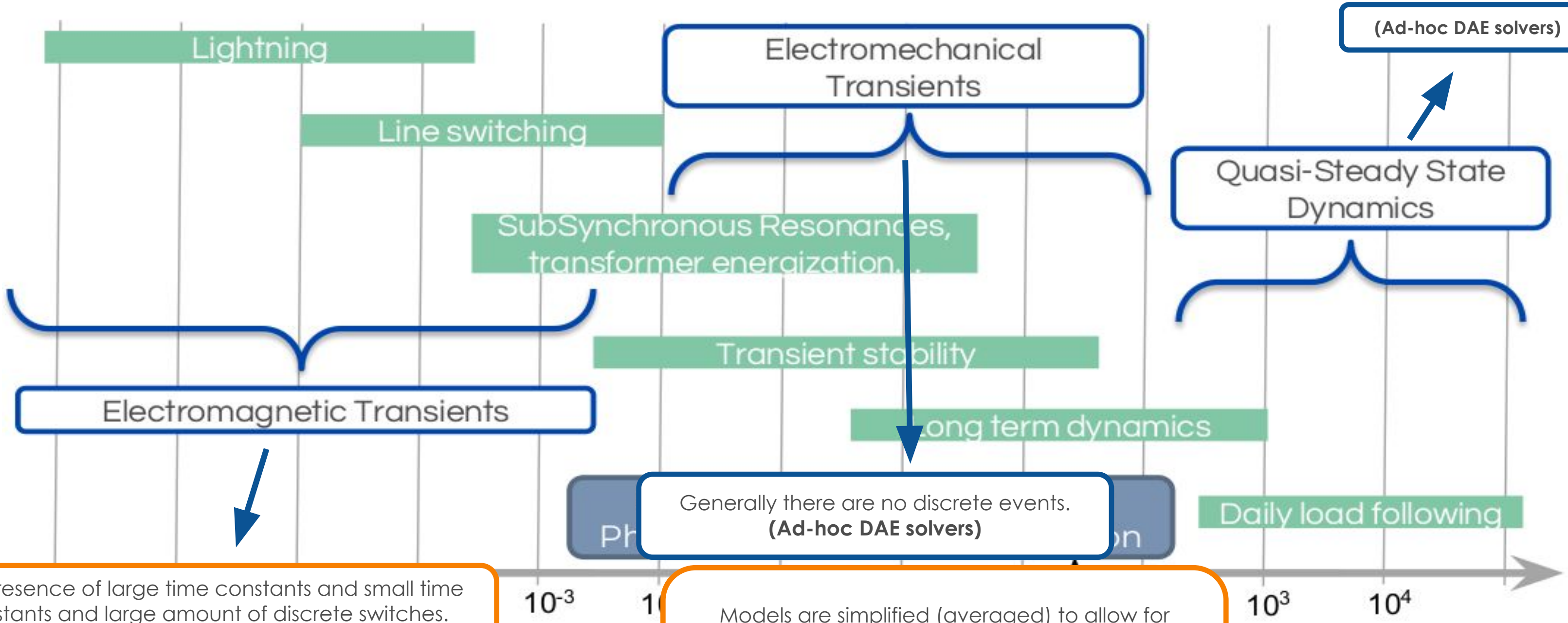
Existing modeling and simulation (and associated) tools were unable to predict this (and other) events.

# Multiple Time-scales for Power System Models

ALSET lab

The models are simplified further by neglecting most dynamics (replacing most differential equations by algebraic equations).

**(Ad-hoc DAE solvers)**




Generally there are no discrete events.
**(Ad-hoc DAE solvers)**

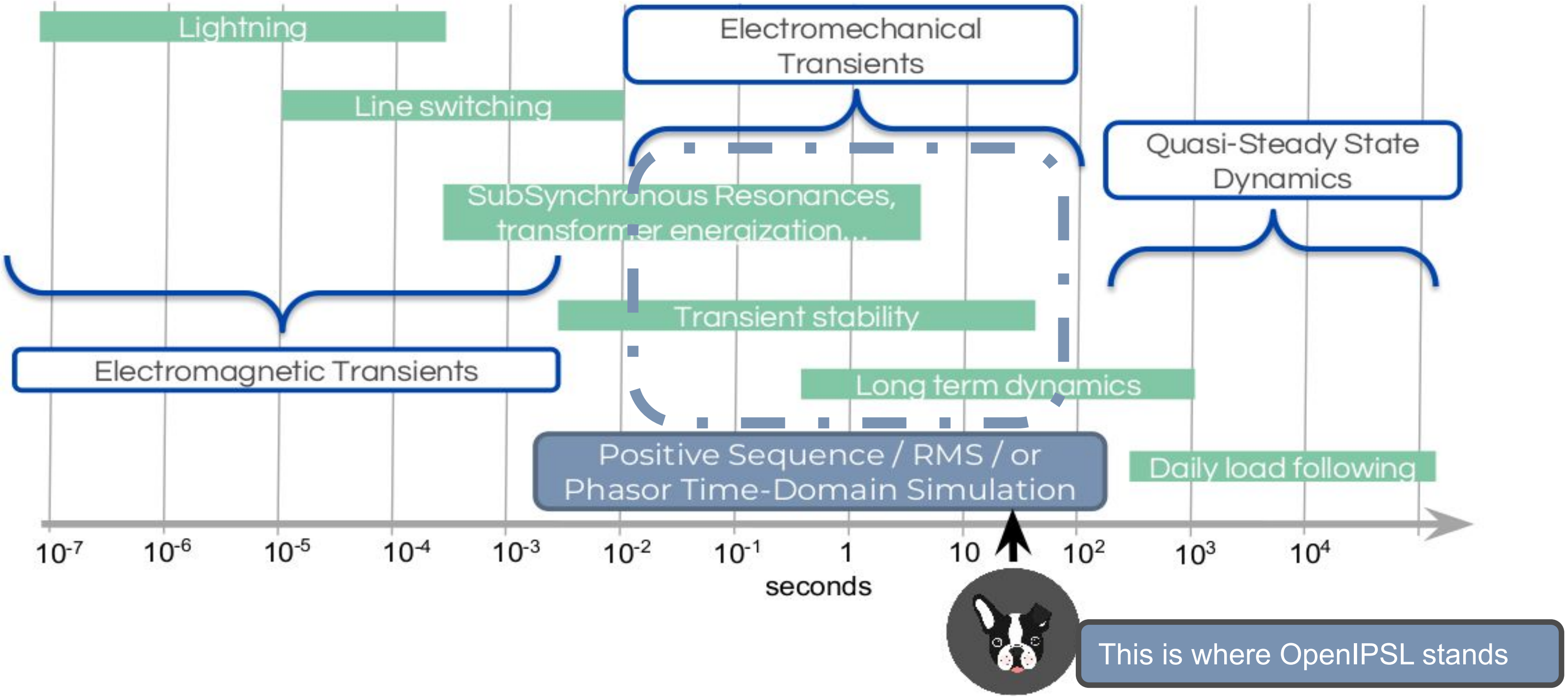
The presence of large time constants and small time constants and large amount of discrete switches.
**Difficult to simulate very large networks (in the past?)**

This is usually deal with by discretizing the model and to solve it using **discrete solvers.**

Models are simplified (averaged) to allow for simulation of **very** large networks.
Ad-hoc solvers have been developed to reduce simulation time, but usually the “model” is “interlaced” with the solver (inline integration)

# Multiple Time-scales for Power System Models

ALSET lab

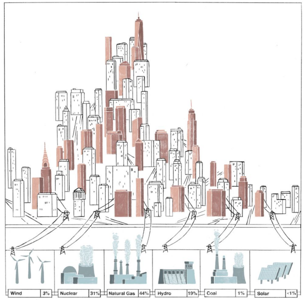


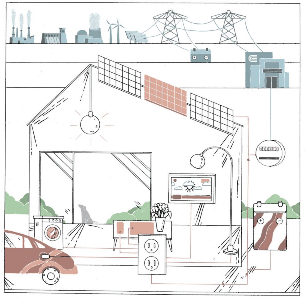
# Presentation Overview

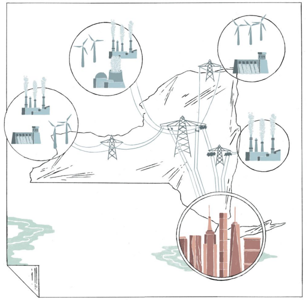
- Models and Simulation in Power Systems
  - The need for models and the different time-scales
- Modelica and Power Systems
- OpenIPSL Library
  - Key features and how it works
- OpenIPSL Applications
  - Simulation of hybrid models (three-phase and single phase)
  - Multi-domain Simulation
  - Training Data Generation for ML-based Application
  - Extremum Seeking Control
- Ongoing Development
  - Continuous Integration and Model Verification
  - Initializing OpenIPSL Models with Python
- Where to find OpenIPSL

# Modelica and Power Systems

- There are many previous and related efforts to create Modelica tools to study power systems.
- Studies have published the challenges of dealing with large power networks using Modelica.
    - Issue that might be circumvented in near future.
- Available libraries:
    - SPOT and PowerSystems.
    - ObjectStab.
    - iPSL (iTesla Power System Library).
        - OpenIPSL takes iPSL as a starting point and moves it forward!





***(1) Strategy*** do not impose the use of a specific simulation environment (software tool), instead,

***(2) Propose***
a **common human and computer-readable mathematical "description":** use of Modelica for unambiguous model exchange.

***(3) Decrease of avoidance forces***

- SW-to-SW validation gives quantitatively an similar answer than domain specific tools.
- Accuracy (w.r.t. to *de facto* tools) more important than performance

***A never-ending effort!***

- The library has served to bridge the gap between the Modelica and power systems community by:
- Addressing resistance to change
- Interacting with both communities

# Presentation Overview

- Models and Simulation in Power Systems
  - The need for models and the different time-scales
- Modelica and Power Systems
- OpenIPSL Library
  - Key features and how it works
- OpenIPSL Applications
  - Simulation of hybrid models (three-phase and single phase)
  - Multi-domain Simulation
  - Training Data Generation for ML-based Application
  - Extremum Seeking Control
- Ongoing Development
  - Continuous Integration and Model Verification
  - Initializing OpenIPSL Models with Python
- Where to find OpenIPSL

# *OpenIPSL* - Key features

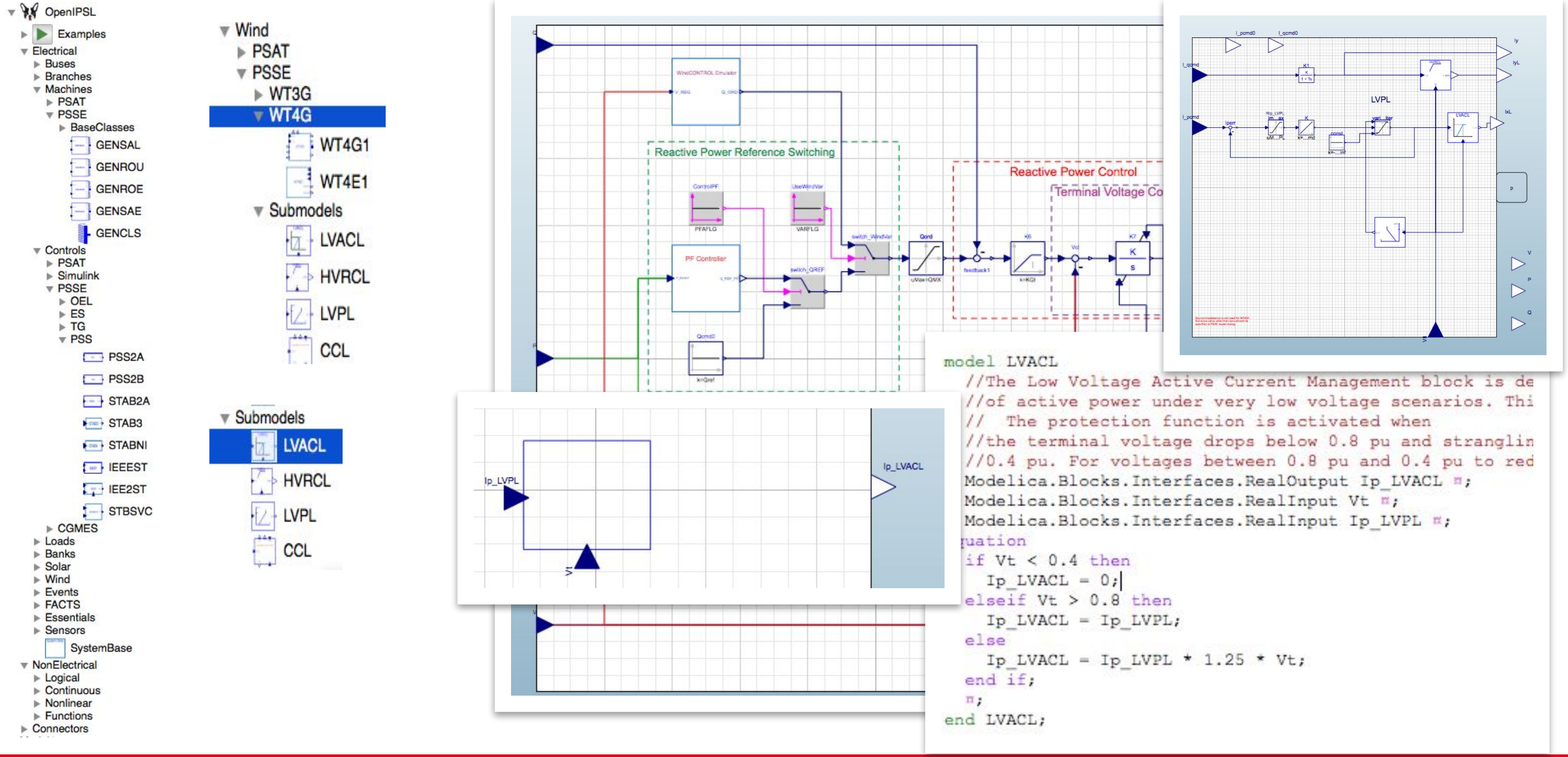
**OpenIPSL** is an open-source Modelica library for power systems that:

- Contains a set of power system components for phasor time domain modeling and simulation of power systems.
- Models have been verified against a number of reference tools (PSS/E, PSAT).

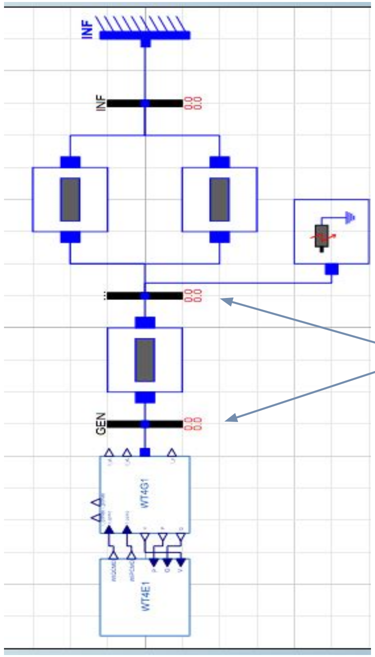
**OpenIPSL** enables:

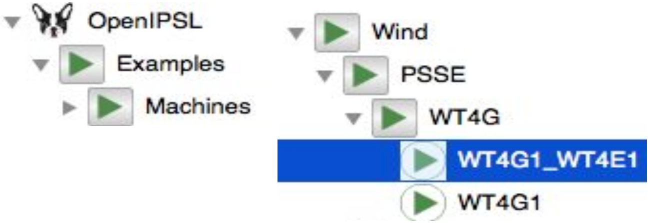
- Unambiguous model exchange.
- Formal mathematical description of models
- Separation of models from tools/IDEs and solvers.
- Use of object-oriented paradigms.

# *OpenIPSL* - Wind Turbine Example



# *OpenIPSL* - Small Network Example




## Class Connections

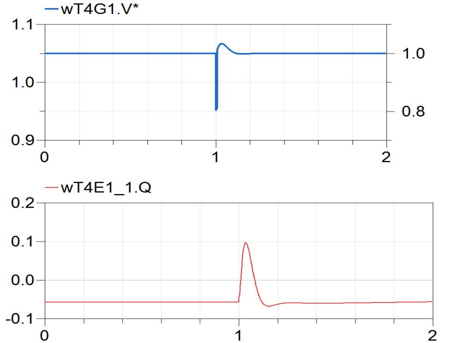
```
equation
  connect(wT4G1.p, GEN.p) ;
  connect(GEN.p, pwLine2.p) ;
  connect(pwLine2.n, BUS1.p) ;
  connect(BUS1.p, pwLine.p) ;
  connect(pwLine1.p, pwLine.p) ;
  connect(pwFault.p, BUS1.p) ;
  connect(pwLine.n, INF.p) ;
  connect(pwLine1.n, INF.p) ;
  connect(INF.p, gENCLS2_1.p) ;
  connect(wT4E1_1.WIQCMD, wT4G1.I_qcmd)
  connect(wT4E1_1.WIPCMD, wT4G1.I_pcmd)
  connect(wT4G1.P, wT4E1_1.P) ;
  connect(wT4G1.V, wT4E1_1.V) ;
  connect(wT4G1.Q, wT4E1_1.Q) ;

end WT4G1_WT4E1;
```

## Resulting Parameter Declaration

```
model WT4G1_WT4E1
  extends Modelica.Icons.Example;
  constant Real pi=Modelica.Constants.pi;
  parameter Real V1=1.00000;
  parameter Real A1=-1.570655e-005;
```

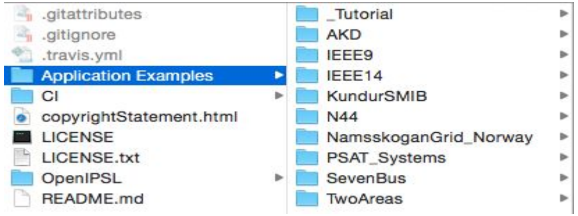
## Resulting Class Instantiation

```
OpenIPSL.Electrical.Branches.PwLine pwLine2(
  G=0,
  B=0,
  R=2.50000E-3,
  X=2.50000E-3) ;
OpenIPSL.Electrical.Wind.PSSE.WT4G.WT4G1 wT4G1(
  V_0=V3,
  angle_0=A3,
  M_b=100,
  P_0=P3,
  Q_0=Q3,
  T_IQCmd=0.02,
  T_IPCmd=0.02,
  V_LVPL1=0.4,
  V_LVPL2=0.9,
  G_LVPL=1.11,
  V_HVRCR=1.2,
  CUR_HVRCR=2,
  RIp_LVPL=2,
  T_LVPL=0.02) ;
```

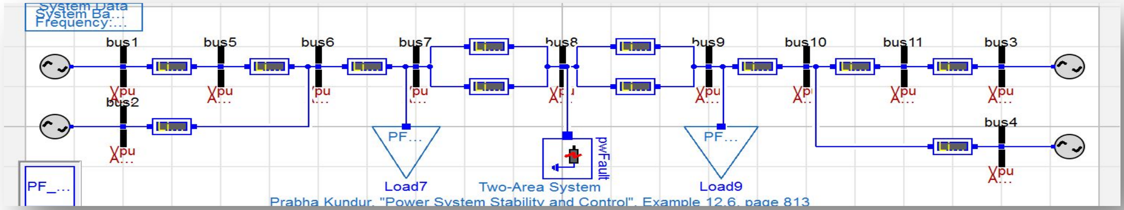


# *OpenIPSL* - Application examples

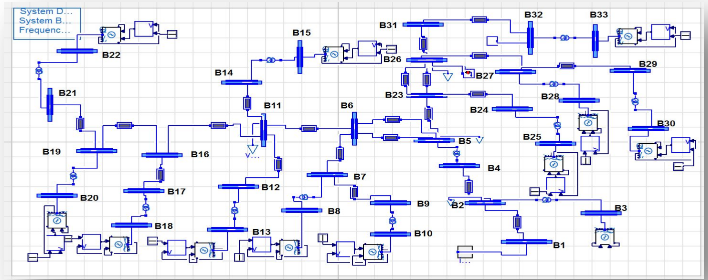
## Many Application Examples Developed!!!



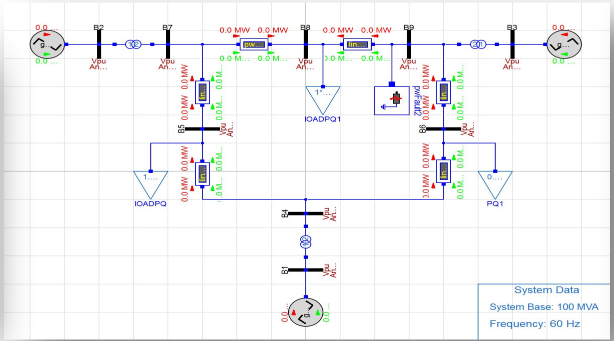

## Klein-Rogers-Kundur 2-Area 4-Machine System



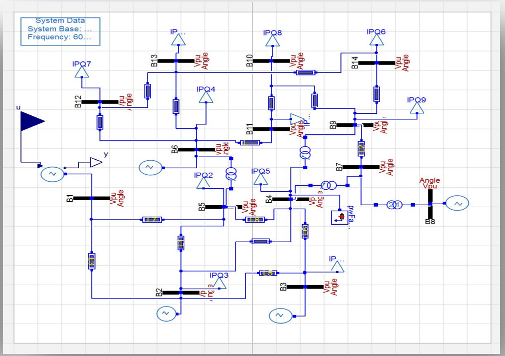

## Namsskogan Distribution Network



## IEEE 9 Bus

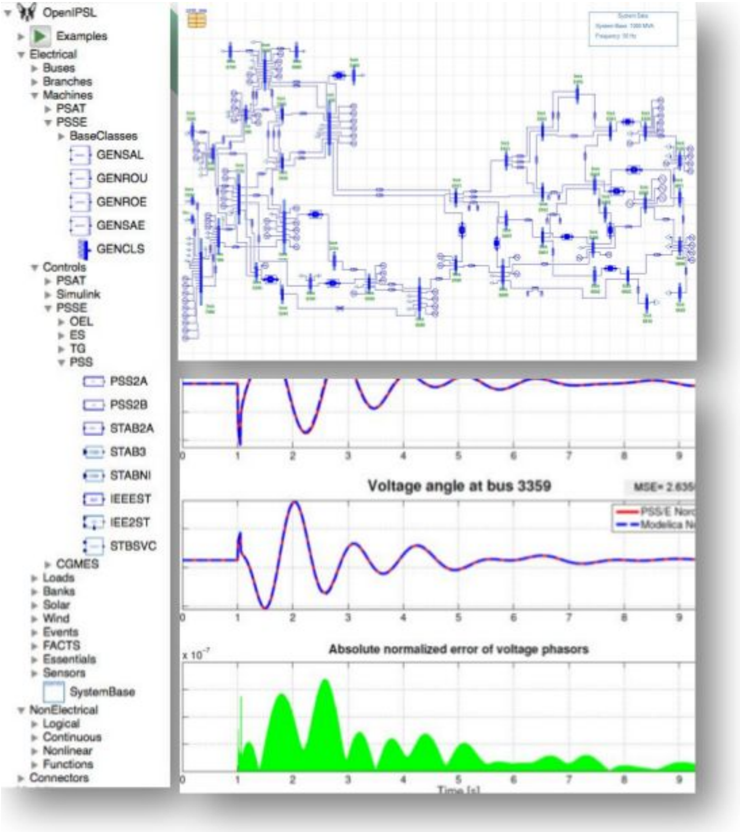



## IEEE 14 Bus



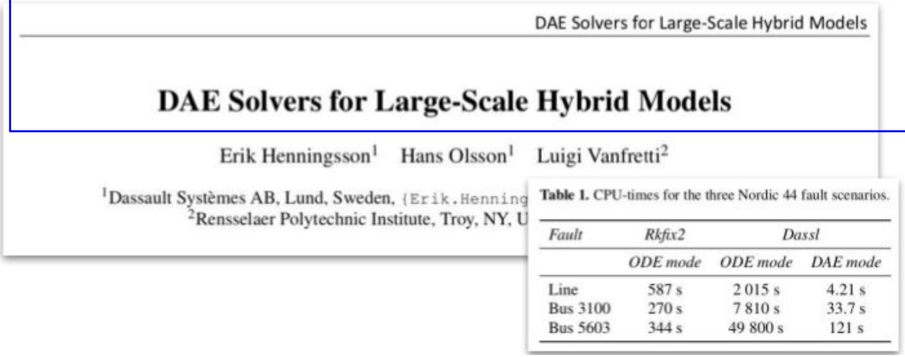
Currently outside the library package but soon to be integrated into the main branch!

# *OpenIPSL* - Large system example




But is it possible to simulate large systems in Modelica using the OpenIPSL? **YES!**

- Simulation depends on the tool, not the model.
- You can simulate the model in many different Modelica tools (facilitates competition between software tools!)
- Dymola 2019FD02 has shown to be competitive to PSS/E as reported in the following paper:



DAE Solvers for Large-Scale Hybrid Models

**DAE Solvers for Large-Scale Hybrid Models**

Erik Henningsson[1] Hans Olsson[1] Luigi Vanfretti[2]

[1]Dassault Systèmes AB, Lund, Sweden, {Erik.Henning
[2]Rensselaer Polytechnic Institute, Troy, NY, U

**Table 1.** CPU-times for the three Nordic 44 fault scenarios.

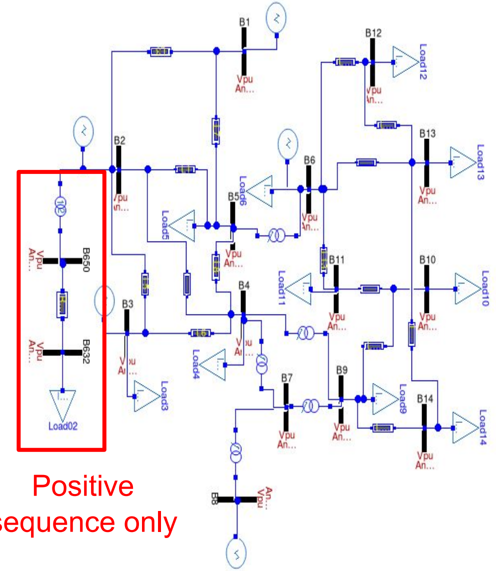
| *Fault* | *Rkfix2* | *Dassl* | |
|---|---|---|---|
| | *ODE mode* | *ODE mode* | *DAE mode* |
| Line | 587 s | 2 015 s | 4.21 s |
| Bus 3100 | 270 s | 7 810 s | 33.7 s |
| Bus 5603 | 344 s | 49 800 s | 121 s |

# Presentation Overview

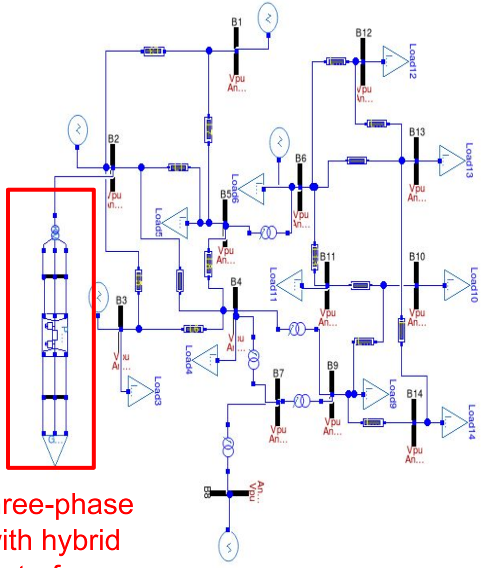
- Models and Simulation in Power Systems
    - The need for models and the different time-scales
- Modelica and Power Systems
- OpenIPSL Library
    - Key features and how it works
- OpenIPSL Applications
    - Simulation of hybrid models (three-phase and single phase)
    - Multi-domain Simulation
    - Training Data Generation for ML-based Application
    - Extremum Seeking Control
- Ongoing Development
    - Continuous Integration and Model Verification
    - Initializing OpenIPSL Models with Python
- Where to find OpenIPSL

# Hybrid Simulation

Positive-Sequence/ Three-Phase Hybrid Interface



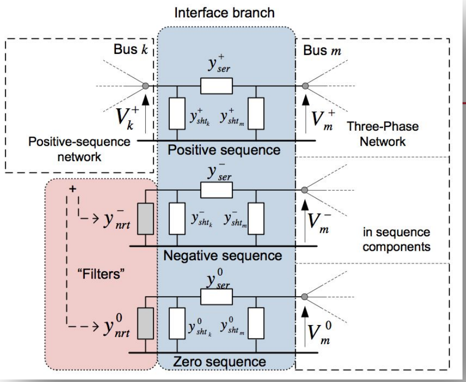
Positive sequence only

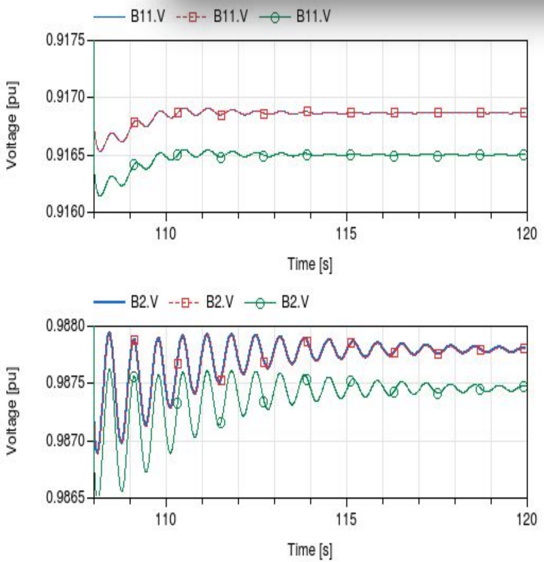


Three-phase with hybrid interface

IEEE 14-bus system model implemented in OpenIPSL

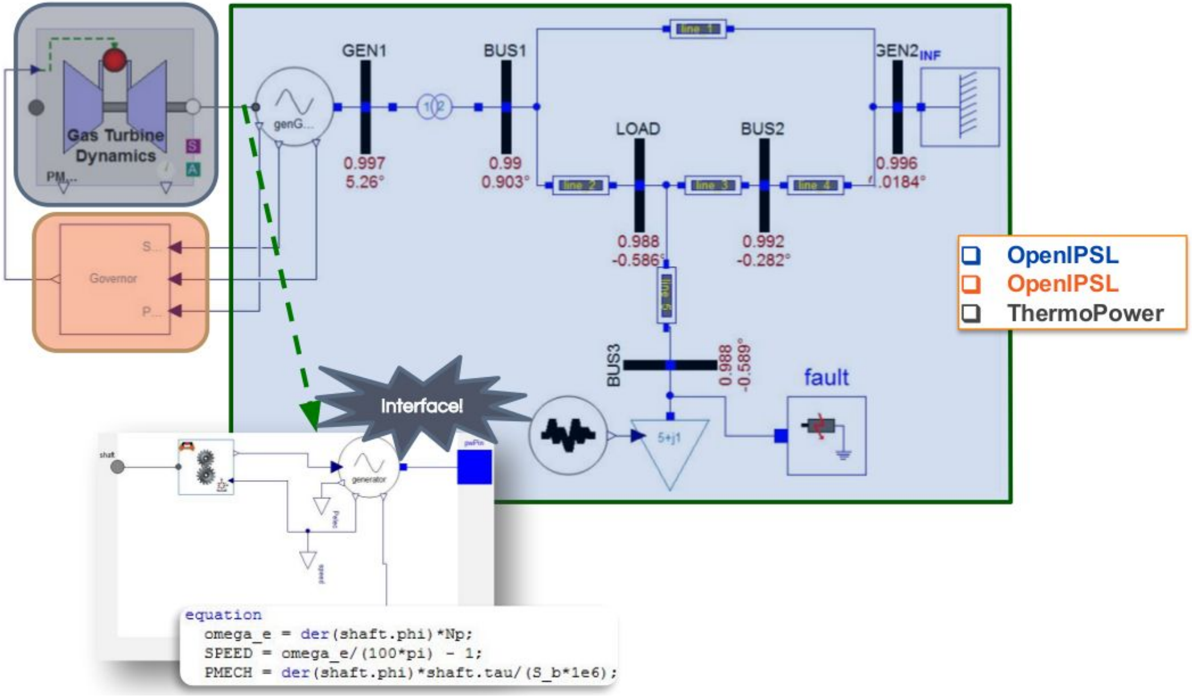
IEEE 14-bus system model using the 3-phase package

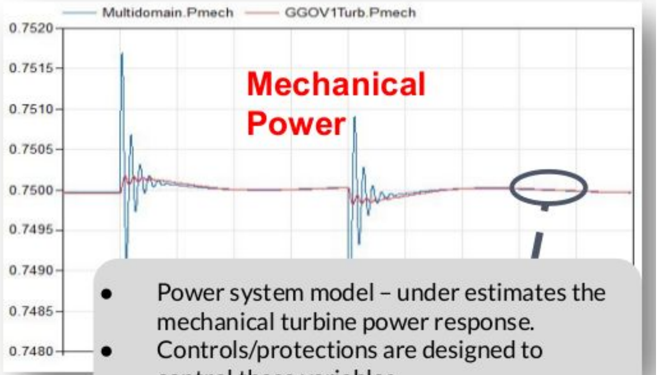


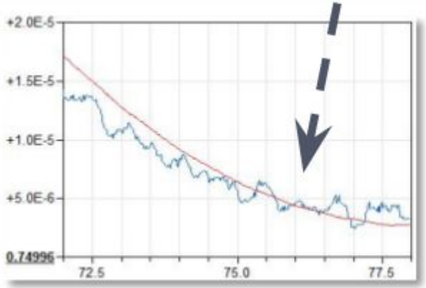
# Multi-domain Simulation

ALSET lab



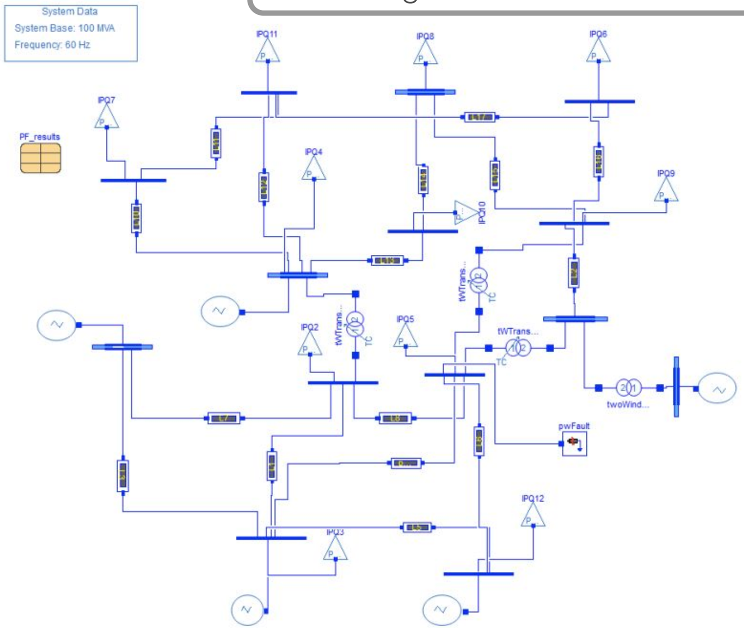



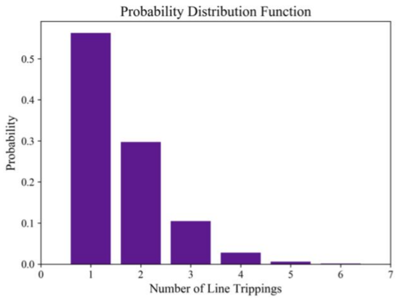

- Power system model – under estimates the mechanical turbine power response.
- Controls/protections are designed to control these variables.

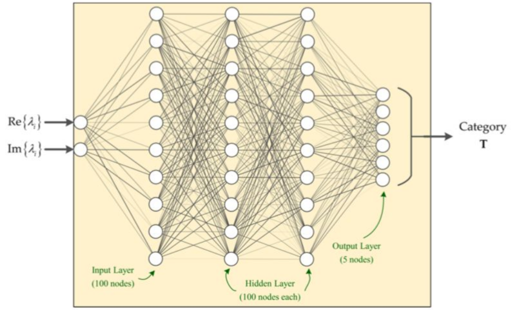


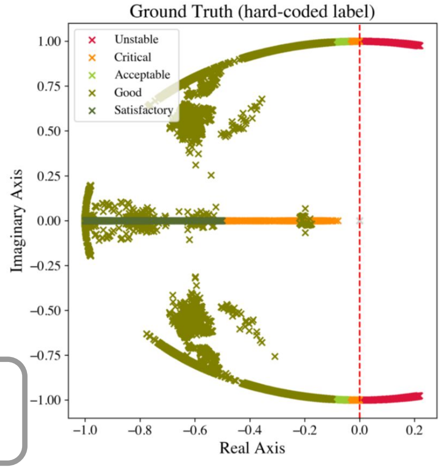
# Training Data Generation for ML-based Application

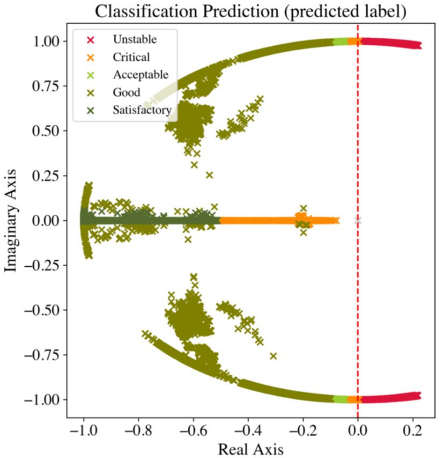
OpenIPSL models are used to generate a large number of scenarios.



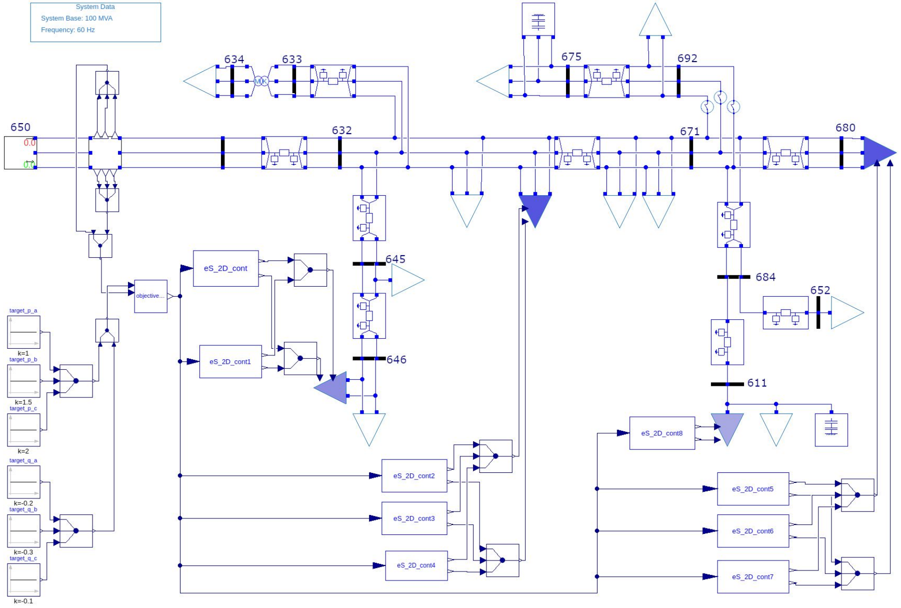




Equation-based models can be easily linearized, helping small-signal analysis

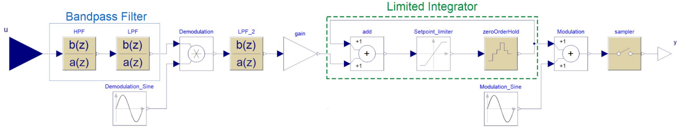
# Extremum Seeking Control

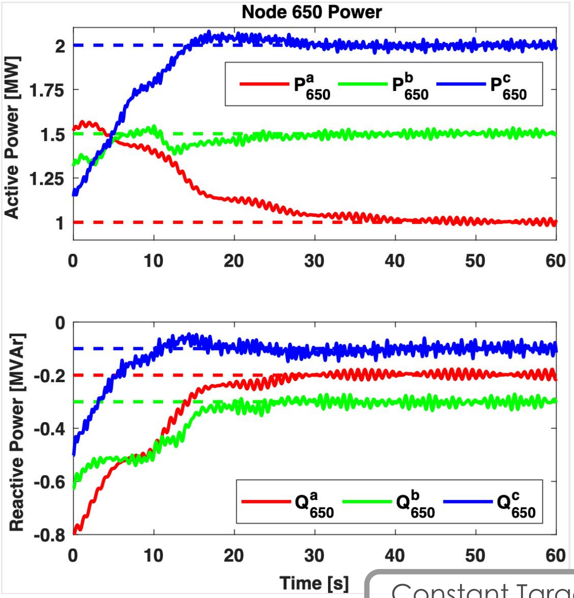
OpenIPSL leveraged to develop and test an Extremum Seeking Modelica Library for DER control




IEEE 13 Node Distribution Feeder (3-phase) with 9 instances of the controller regulating 3-phase P & Q at the feeder head

Extremum Seeking Controller



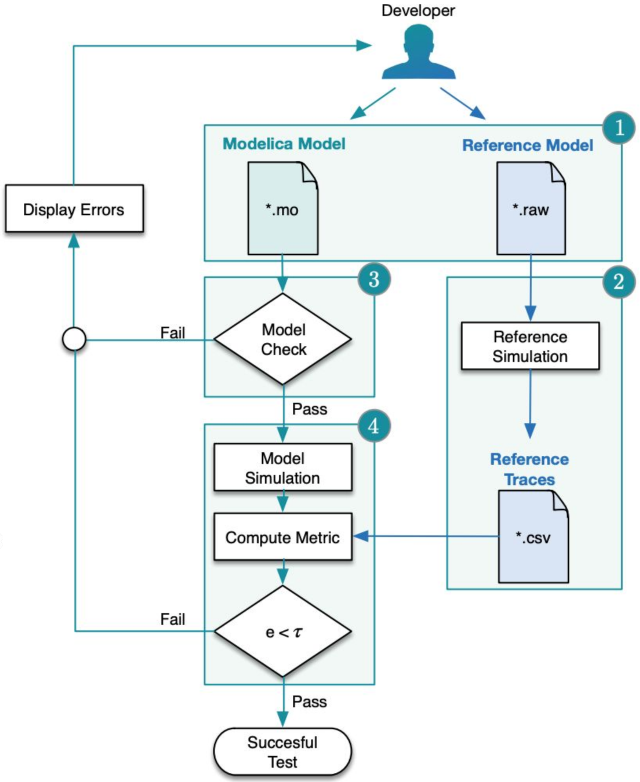

Constant Target Tracking Experiment Results

# Presentation Overview

- Models and Simulation in Power Systems
    - The need for models and the different time-scales
- Modelica and Power Systems
- OpenIPSL Library
    - Key features and how it works
- OpenIPSL Applications
    - Simulation of hybrid models (three-phase and single phase)
    - Multi-domain Simulation
    - Training Data Generation for ML-based Application
    - Extremum Seeking Control
- Ongoing Development
    - Continuous Integration and Model Verification
    - Initializing OpenIPSL Models with Python
- Where to find OpenIPSL

# Continuous Integration / Model Validation

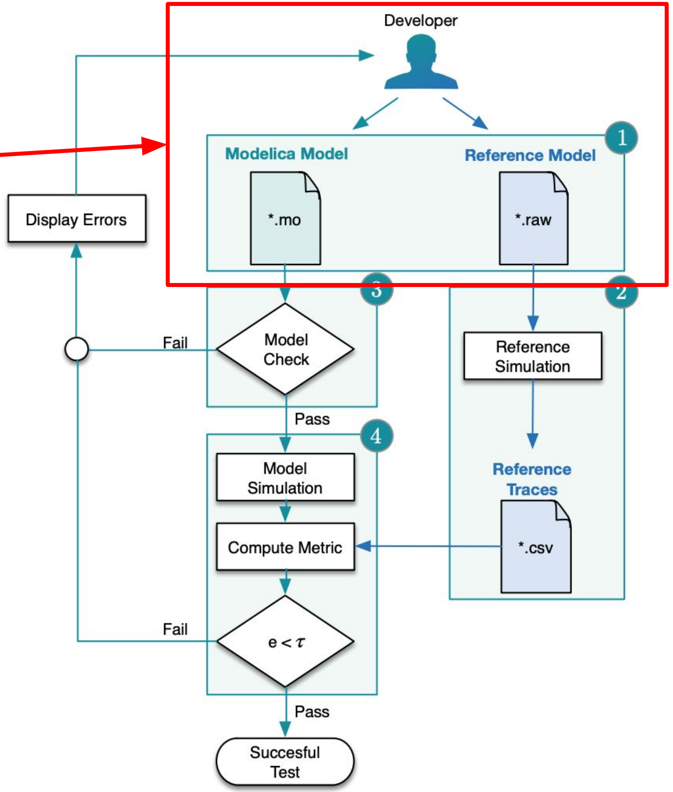
- In order to maintain and further develop the OpenIPSL library we need to create a system to check and store the library.
- **Store:** in order to store, we created a Github repository. It is public for anyone to see/clone/access.
- **Check:** in order to check if the models are the same linear regression will be performed in order to see if they are within tolerance.
- The process we use is…



# Continuous Integration / Model Validation

Step 1

The developer (anyone) creates the OpenIPSL model based on a reference model and uploads it into our GitHub page.



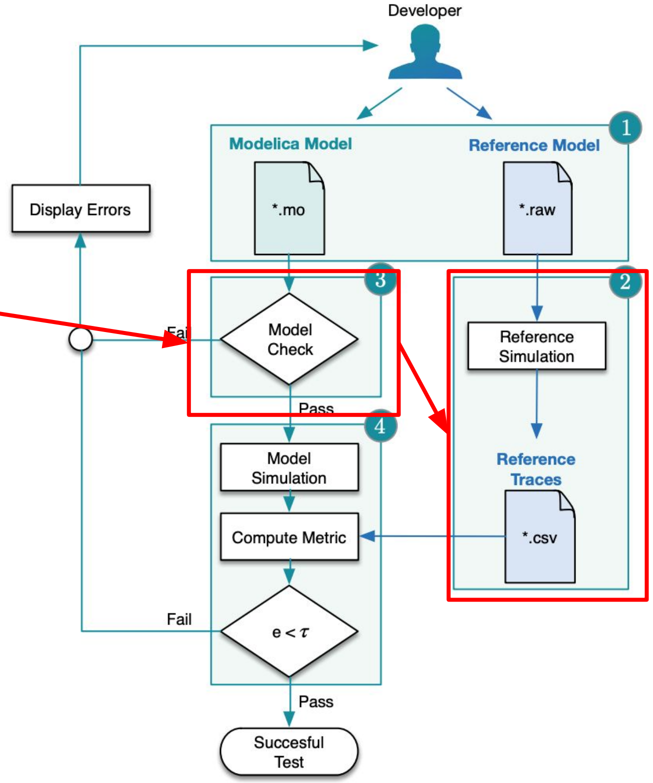

Modelica Model can be developed in any Modelica software:

- Dymola
- OpenModelica
- System Modeler
- etc.…

# Continuous Integration / Model Validation

Step 2

Tests are performed on **both** models for comparison.



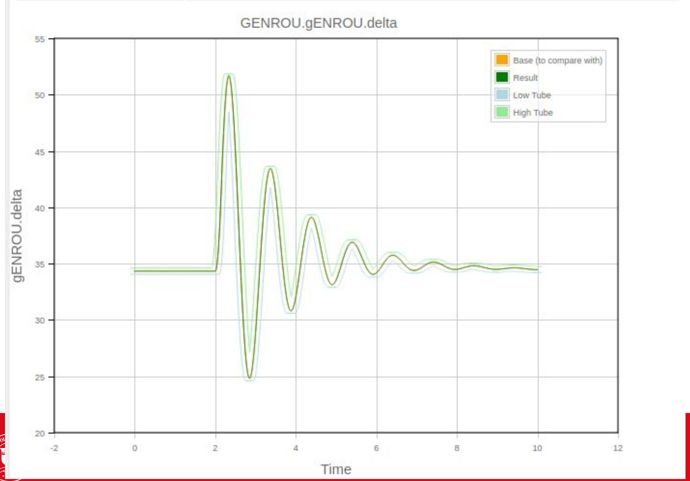

Tests performed:

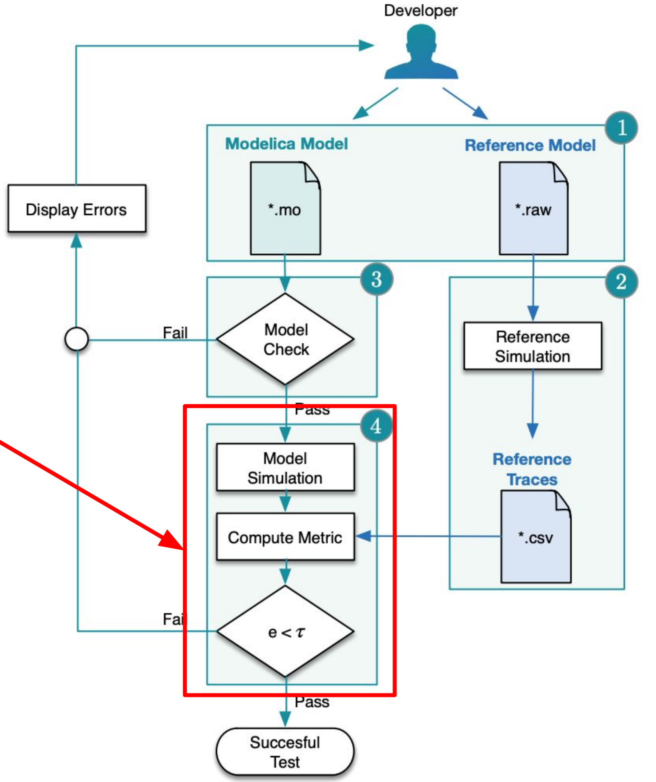
- Faults.
- Load Variation
- Reference steps in exciters.

# Continuous Integration / Model Validation

Step 3

From selected signals, both signals will be compared and linear regression will be performed in order to see if they are the same within tolerance.

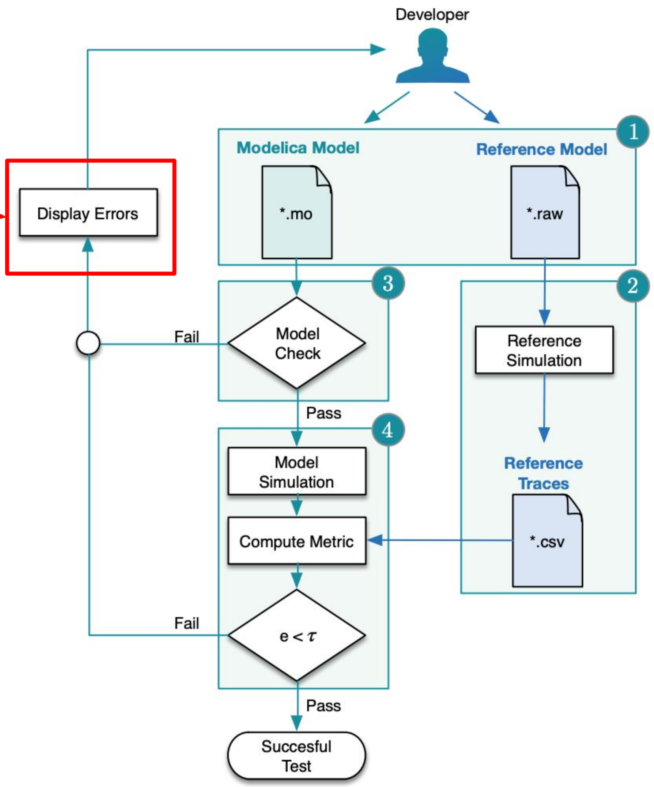


# Continuous Integration / Model Validation

Step 4

The developer (anyone) creates the OpenIPSL model based on a reference model and uploads it into our GitHub page.



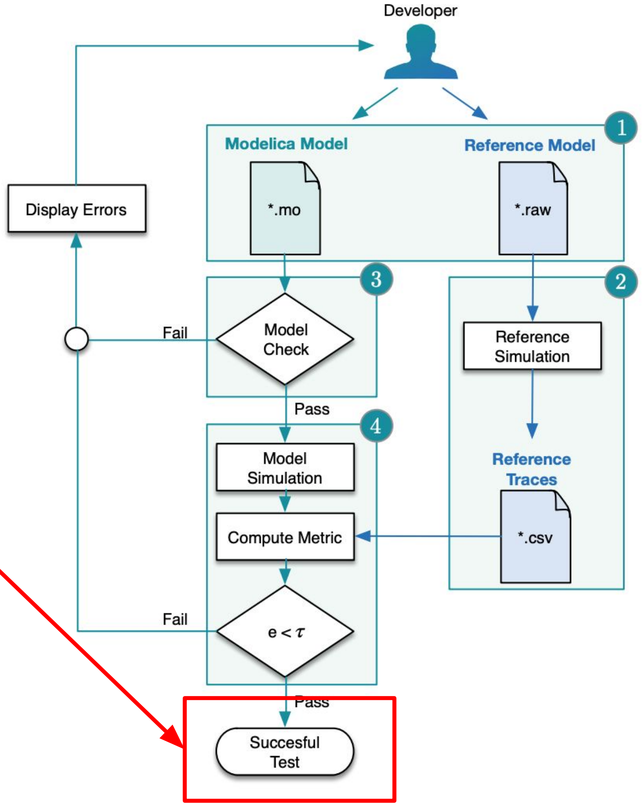

Administrators get notified and the developer is notified in order to fix errors/improve the model.

# Continuous Integration / Model Validation

Step 5

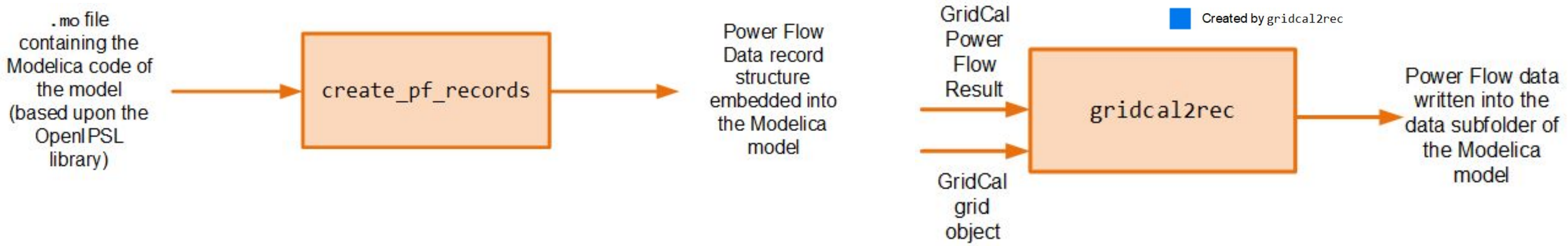
If the signals are within tolerance for **all** the tests then the model can be uploaded to the repository.

This process has been optimized by using Travis Continuous integration tool. Meaning **if and only if** the model's signals are within tolerance the new model will be added to the repository.



# Initializing OpenIPSL Models with Python

- A power flow solution is a cornerstone for any power system analysis.
- Several open-source and commercial alternatives exist to *generate* power flow solutions.
- We intend to use a Python open-source library (GridCal) for power flow computation.
  - It is fully compatible with commercial tools such as PSS/E.
  - Automate it to generate power flow results for initializing dynamic simulations in Modelica.

# Presentation Overview

ALSETlab

- Models and Simulation in Power Systems
  - The need for models and the different time-scales
- Modelica and Power Systems
- OpenIPSL Library
  - Key features and how it works
- OpenIPSL Applications
  - Simulation of hybrid models (three-phase and single phase)
  - Multi-domain Simulation
  - Training Data Generation for ML-based Application
  - Extremum Seeking Control
- Ongoing Development
  - Continuous Integration and Model Verification
  - Initializing OpenIPSL Models with Python
- Where to find OpenIPSL

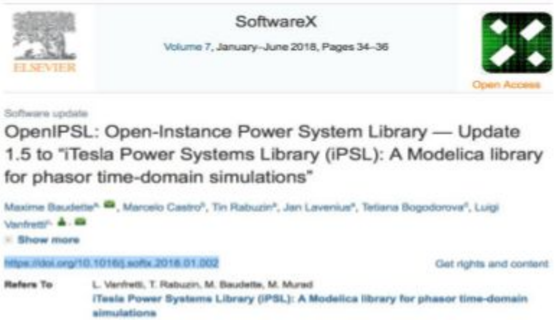
# *OpenIPSL* - Where to find it?

The **OpenIPSL** can be found online

- http://openipsl.org

Our work on **OpenIPSL** has been published in the SoftwareX Journal:

- https://doi.org/10.1016/j.softx.2018.01.002



ELSEVIER

SoftwareX

Volume 7, January–June 2018, Pages 34–36

Open Access

Software update

OpenIPSL: Open-Instance Power System Library — Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations"

Maxime Baudette, Marcelo Castro, Tin Rabuzin, Jan Lavenius, Tetiana Bogodorova, Luigi Vanfretti

Show more

https://doi.org/10.1016/j.softx.2018.01.002

Get rights and content

Refers To: L. Vanfretti, T. Rabuzin, M. Baudette, M. Murad
iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations

# Acknowledgements

This work was funded in part by the New York State Energy Research and Development Authority (NYSERDA) through the Electric Power Transmission and Distribution (EPTD) High Performing Grid Program, in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877, in part by the CURENT Industry Partnership Program, and in part by the Center of Excellence for NEOM Research at King Abdullah University of Science and Technology.

We would also like to thank:

- Dietmar Winkler, University of South-Eastern Norway, and Giuseppe Laera, Rensselaer Polytechnic Institute, for all their contributions to the library development;
- Sergio Dorado-Rojas, Rensselaer Polytechnic Institute, and Maxime Baudette, Lawrence Berkeley National Laboratory, for sharing their applications using library.

Rensselaer

why not change the world?®