Development and Continuous Integration of the *OpenIPSL* Modelica Library for Power Systems Simulation

Maxime Baudette, Tin Rabuzin and *Luigi Vanfretti*

Assoc. Prof. Luigi Vanfretti - luigiv@kth.se

https://www.kth.se/profile/luigiv/

This work was supported in part by:

*OpenModelica* Annual Workshop

February 6th, 2017
Linköping University, Sweden
Outline

- **OpenIPSL**
  - Modelica and Power Systems
  - OpenIPSL
  - Project Documentation
  - Latest Developments
- **Continuous Integration**
  - A Collaborative Workflow
  - Toward Continuous Integration
  - Continuous Integration Service
  - Extensions
  - Model Validation
  - GitHub Integration
Previous and Related Efforts

- Modelica for power systems was first attempted in the early 2000’s (Wiesmann & Bachmann, Modelica 2000) - “electro-magnetic transient (EMT) modeling” approach.
  - SPOT (Weissman, EPL-Modelon) and its close relative PowerSystems (Franke, 2014); supports multiple modeling approaches –i.e. 3phase, steady-state, “transient stability”, etc.

- Electro-mechanical modeling or “transient stability” modeling:
  - Involves electro-mechanical dynamics, and neglects (very) fast transients
  - For system-wide analysis, easier to simulate/analyze - domain specific tools approach

- ObjectStab (Larsson, 2002; Winkler, 2015) adopts transient modeling.

- The PEGASE EU project (2011) developed a small library of components in Scilab, which where ported to proper Modelica in the FP7 iTesla project (2012-2016).

- The iPSL - iTesla Power Systems Library (Vanfretti et al, Modelica 2014, SoftwareX 2016), was released during 2015. Most models validated against typical power system tools.

**OpenIPSL takes iPSL as a starting point and moves it forward (this presentation).**

- F. Casella (OpenModelica 2016, Modelica 2017) presents the challenges of dealing with large power networks using Modelica, and a dedicated library to investigate them using OM.
Why another library for power systems?

- Why not use one of the existing Modelica projects?
  - *There is no technical argument*: in principle, either SPOT, PowerSystems, or ObjecStab could have been used instead of creating a new library (iPSL or OpenIPSL)

Social Aspects (Vanfretti et al, Modelica 2014):

- Resistance to change: irrational and dysfunctional reaction of users
  - Users of conventional power system tools are skeptical about any other tools different to the one they use (or develop), and have concerns about new technologies (lack of knowledge)
- Change agents contribute (+/-) to address resistance through actions and interactions:
  - Did not impose the use of a software tool, instead:
  - Propose a common math. “description”: use of Modelica for unambiguous model exchange.
- Decrease avoidance forces:
  - SW-to-SW validation give quantitatively an similar answer than domain specific tools.

A never-ending effort:

- Our (my) goal has been to bridge the gap between the Modelica and power systems community by
  - Addressing resistance to change (see above)
  - Interacting with both communities – different levels of success…
The **OpenIPSL** Project

- **KTH SmarTS Lab** (my research team) actively participated in the group or partners developing **iPSL** until the end of the **iTesa** project (March 2016)
- **iPSL** is a nice prototype, *but we identified the following issues:*
  - **Development:** Need for compatibility with OpenModelica, (better) use of object orientation and proper use of the Modelica language features.
  - **Maintenance:** poor harmonization, lack of code factorization, etc.
  - **Human issues:** The development workflow was complex, because of
    - Different parties with disparate objectives, levels of knowledge, philosophy, etc.

New research requirements and the experiences from previous effort indicated:
- *a clear need for a different development approach* –
  - one that should address a complex development & maintenance workflow!

- **OpenIPSL** *started as a fork* of **iPSL**
- **OpenIPSL** is hosted on GitHub at [https://github.com/SmarTS-Lab/OpenIPSL](https://github.com/SmarTS-Lab/OpenIPSL)
- **OpenIPSL** is actively developed by SmarTS Lab members and friends, as a research and education oriented library for power systems
  - *it is ok to try things out*!

**Fork:** copy of a project going *in a different development direction*
OpenIPSL is an open-source Modelica library for power systems

- It contains a set of **power system components** for **phasor time domain** modeling and simulation.
- Models have been **validated** against a number of reference tools.

**OpenIPSL enables:**

- **Unambiguous** model exchange
- **Formal mathematical description** of models
- **Separation of models** from IDEs and solvers
- Use of **object-oriented** paradigms
The intention is to have comprehensive documentation in the repositories:

- Documentation of the code changes
  → Explicit messages in *commits* and *pull-requests*

- Documentation of the project
  - Presentation
  - User guide
  - Dev. guidelines & How to contribute

→ The documentation is written in *reStructuredText* (reST) hosted on http://openipsl.readthedocs.io/

*Note*: Model documentation is not included, users are referred to the proprietary documentations.
The **OpenIPSL** Project

Latest Developments/Contributions

Some of the latest development in the library:

- **100% Compatibility with OM (100% Check, 100% Simulation for components)** through efforts in Continuous Integration adoption
- Change in the models to include inheritance (code factorizing)
- Fixing and validating network models (thanks to CI)
- Component for interfacing OpenIPSL with 3 phase models (aka MonoTri)
  - For distribution grid (unbalanced) simulations
  - Starting point for mixed transmission and distribution network simulations

**ENTSO-E IOP:**

- Proof of concept and test model
- Excitation system and small network model

**OpenCPS Models**

- Small power network models for analysis of continuous and hybrid systems (sampling and discretized AVR model)
- Use case examples being developed will be added soon.
New research requirements and the experiences from previous effort indicated a clear need for a different development approach - one that should address a complex development and maintenance workflow!

How to master a complex development workflow?

Continuous Integration
A Collaborative Workflow

We adopted the **pull-request** workflow (or GitHub workflow):

- Participants **fork** the repository and work in their repository
- Changes are submitted to the main repository as **pull-requests**
- The pull-requests are **reviewed** by “admin” members of the repository
  - upon **validation** the changes are merged in the code of the repository

- Mistakes can be made by members of our team, **we are still learning!**
- The Git **workflow** adopted allows to **minimize the impact of these errors**.
- Increased library quality!
Toward Continuous Integration

- The previous workflow was used by only few people and resulted in no control over the code quality, even though DVCS was being used.

- The newly adopted workflow turned suitable for the development team, but generated a strong burden for the code review.

This sparked the idea of implementing a **Continuous Integration workflow**:

→ Focus on “lighter”, more frequent pull-requests, containing less code change, all related to a single feature to facilitate the code validation

→ Implement a CI service to automate recurring code validation tests, to liberate “admin” resources.
A CI service was implemented and integrated to the repository. The Modelica support was achieved with the following architecture:

- **Travis** as CI service provider
- **Docker** as the “virtualization” architecture
- **DockerHub** to host a Docker image with Python & OpenModelica

The CI performs automated syntax checks on the library. New changes are submitted as a new pull request to the master branch. The pull request triggers the Travis CI. The tailored Docker image is pulled. The reference traces are pulled from a dedicated server. The latest version of the library containing the changes is pulled from GitHub. The Docker is instantiated to create a replicable environment where the tests are carried out. The pass / fail flag from the tests on Travis is sent to Github.
Go to the OpenIPSL Github repo: [https://github.com/SmarTS-Lab/OpenIPSL](https://github.com/SmarTS-Lab/OpenIPSL), see runTest.py

**OpenIPSL: Open-Instance Power System Library:**

The OpenIPSL or Open-Instance Power System Library is a fork of the iTesla Power System Library, currently developed and maintained by the SmarTS Lab research group, collaborators and friends (contributions are welcome!).
Extension of the CI Service

The first implementation eliminated parts of the ‘rebarbative’ tasks by automating the code checks:

• Avoid error propagation in the library, models “out-of-sync”
• Implementation entirely based on OpenModelica
  → 100% OM Compatibility achieved!

From this successful implementation, an extension was investigated to include model validation into the CI service:

• Model validation tests were carried out “offline” during the model development stages
  → We did it before!
• Automated model validation (aka regression testing), ensures code changes won’t affect existing models
  → Library integrity guaranteed
In the original implementation of the models of the OpenIPSL, a software-to-software validation workflow was designed and carried out “offline”:

- Models are implemented from several reference programs
  - **PSAT**, domain specific tool in MATLAB/Simulink by F. Milano
  - **PSS/E**, domain specific tool from Siemens PTI
- Modelica models were validated using small scale power network
- The traces from the Modelica models were qualitatively and quantitatively assessed: compared to the reference traces

→ Gives confidence to users having a long experience with these reference software …
Model Validation Workflow (SW-to-SW) (2/2)

Reference SW Tool

Power Flow Calculations

Time-domain simulation

Modelica Model

Graphical and Quantitative Assessment

\[ RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2} \]

Model

<table>
<thead>
<tr>
<th>Model</th>
<th>Validation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAC1</td>
<td>Fail</td>
</tr>
<tr>
<td>EXAC2</td>
<td>Fail</td>
</tr>
<tr>
<td>EXST1</td>
<td>Pass</td>
</tr>
</tbody>
</table>

Signal

<table>
<thead>
<tr>
<th>Signal</th>
<th>RMSE</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>2.838e-3</td>
<td>+</td>
</tr>
</tbody>
</table>
Continuous Integration (CI)
Full workflow implementation

Workflow Summary:

• A two-stage process
  – Modelica *syntax* check
  – Model *validation* check
• Fully automated through online CI services

→ Diagnostic help to the developers to *locate the error*
Continuous Integration (CI)  
GitHub Integration

- Syntax Error
- Model Error
- All OK!
- Merging Blocked
- Merging Allowed
Main Take Away(s)

The implementation of Continuous Integration services allows to:

- Systematically check the code syntax
- Systematically check the integrity of the library (through SW-to-SW validation)
- Easier collaboration with more developers
- Easier to diagnostic potential errors
- Better code quality

Other existing Modelica libraries could adopt CI:

- Better compatibility with OM and
- Modelica language version(s).

The OpenIPSL library can be found online: [https://github.com/SmarTS-Lab/OpenIPSL](https://github.com/SmarTS-Lab/OpenIPSL)

Come to the MODPROD Tutorial 3 to learn to use OpenIPSL!
Our work on OpenIPSL has been published in the SoftwareX Journal:

- http://dx.doi.org/10.1016/j.softx.2016.05.001

The OpenIPSL can be found online:

- https://github.com/SmarTS-Lab/OpenIPSL

RaPId, a **system identification** software that uses OpenIPSL can be found at:

- https://github.com/SmarTS-Lab/iTesla_RaPId
- http://dx.doi.org/10.1016/j.softx.2016.07.004

Thanks to all current and former students and developers at SmarTS Lab