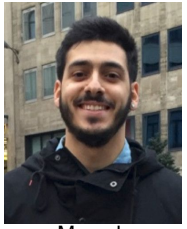
Rensselaer

why not change the world?®

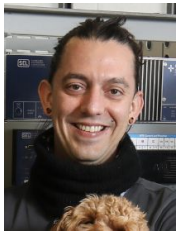
ALSET lab

Sergio Dorado-Rojas S'19

Manuel Navarro F'18



Marcelo de Castro F'18



Prof. Luigi Vanfretti

Time-domain Simulation Performance Benchmark between Modelica and Dymola

Sergio A. Dorado-Rojas, Manuel Navarro Catalán, Marcelo de Castro Fernandes, Luigi Vanfretti

Overview

- Background
- Methodology
  - OpenIPSL Library
  - IEEE 14 Bus Power System with OpenIPSL
- Simulation Automation
- Simulation Results
- Performance Results
- Performance Scores
- Future Work

Overview

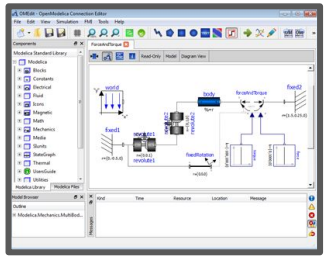
- Background
- Methodology
  - OpenIPSL Library
  - IEEE 14 Bus Power System with OpenIPSL
- Simulation Automation
- Simulation Results
- Performance Results
- Performance Scores
- Future Work

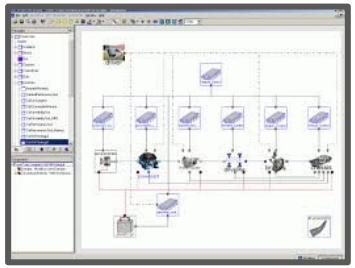
Background

Computer-based studies are common in power systems.
Complexity of models increases with the on-going penetration of renewable energies.

Modelica represents an accurate, equation-based, multi-domain modeling and simulation alternative.

- The impact of the Modelica language has grown significantly during the last years.
- Development of a vast amount of libraries from users coming from a very wide spectrum.





Free tools such as *OpenModelica* are fundamental for learning the language at no cost.

Commercial tools such as *Dymola*, SystemModeler or SimulationX provide advanced functionalities that satisfy particular requirements from the industry.

It is not clear how open-source tools measure up to tools with a price tag

This work addresses this question by comparing the time-domain simulation performance of Dymola and OpenModelica subjected to different solver settings

Overview

- Background
- Methodology
  - OpenIPSL Library
  - IEEE 14 Bus Power System with OpenIPSL
- Simulation Automation
- Simulation Results
- Performance Results
- Performance Scores
- Future Work

Methodology

The proposed benchmark was developed as follows:

- Modification of the IEEE14 bus application example of the open-source library OpenIPSL to configure three different simulation discrete event scenarios
- Automation of time-domain simulations with Python using the corresponding Dymola/OpenModelica interfaces.
- Quantify the results of the simulation execution for the different tools in each scenario through a single performance metric.

OpenIPSL Library

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 (mainly PSS/E)

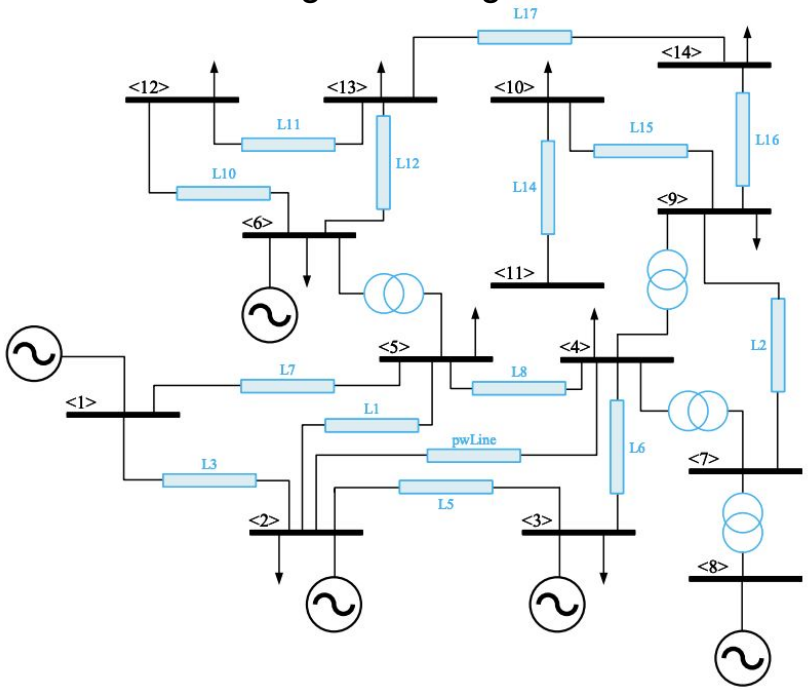
OpenIPSL enables:

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

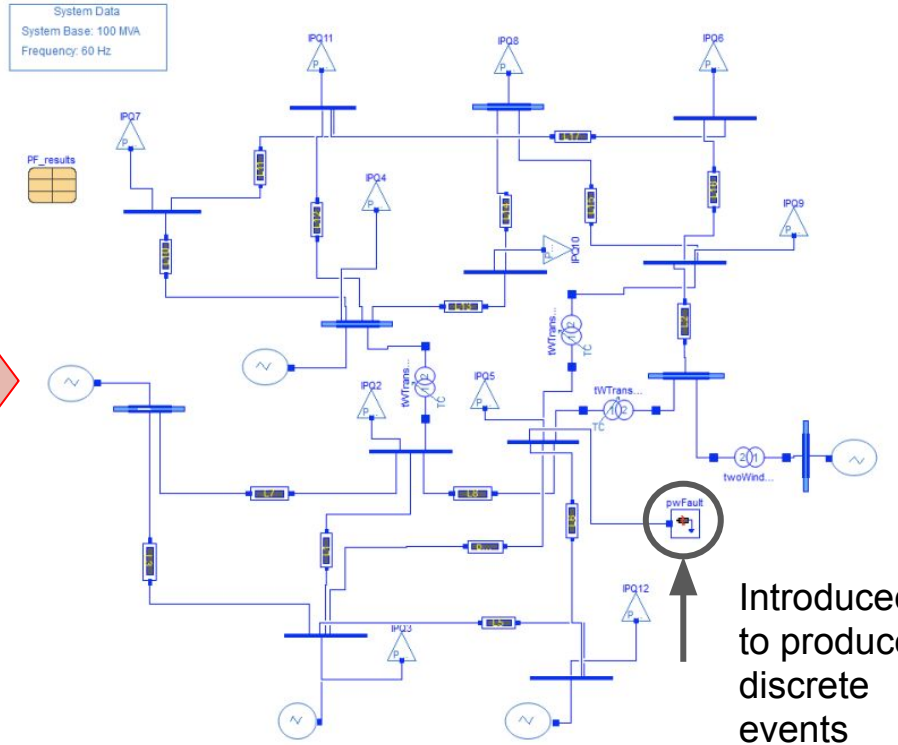
https://github.com/OpenIPSL/OpenIPSL

IEEE 14 Bus System

Single-line diagram



OpenIPSL Implementation



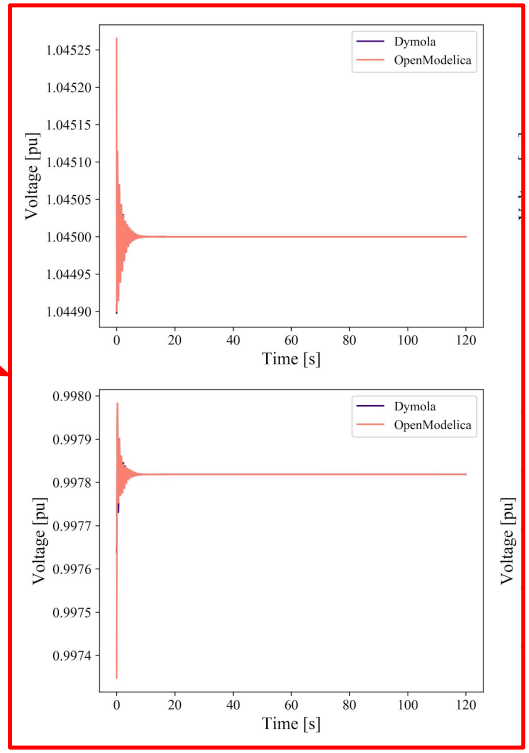
Introduced to produce discrete events

Description of Testing Scenarios

The IEEE 14 Bus model will be tested for:

- **Different Modelica Software**
  - Dymola
  - OpenModelica
- **Different Solvers**
  - `dassl`
  - `euler`
  - `Runge Kutta`
- **Different Scenarios**
  - **Model Initialization**
  - **Line Opening** (Between buses 2 and 4)
    t = 60 s and re-close at t = 61.5 s
  - **Bus Three-phase-to-Ground Faults**
    One happening at t = 20 s and removed at t = 21.2 s
    The other at t = 80 s and removed at t = 81.2 s

Example:
`dassl`
Initialization
Scenario



Overview

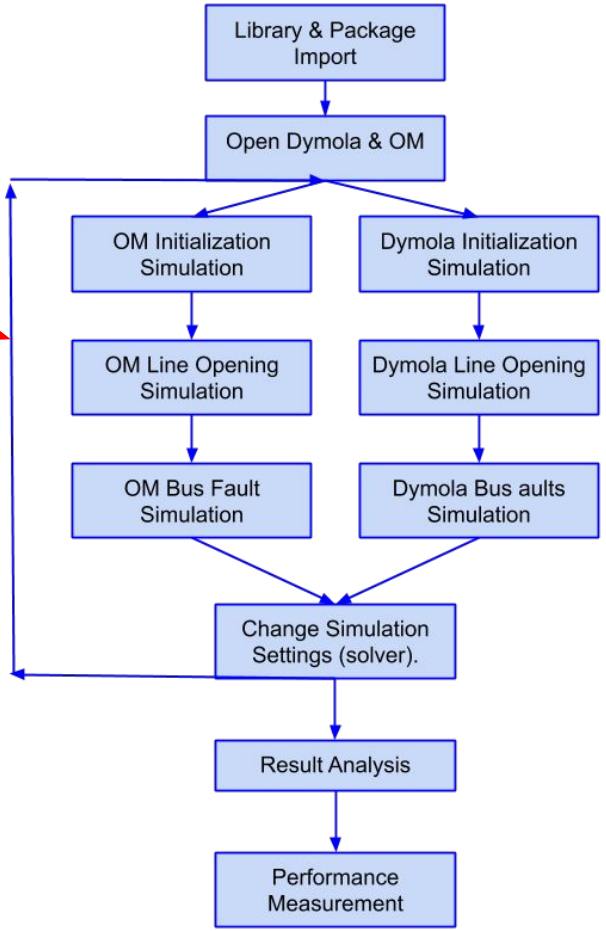
- Background
- Methodology
  - OpenIPSL Library
  - IEEE 14 Bus Power System with OpenIPSL
- Simulation Automation
- Simulation Results
- Performance Results
- Performance Scores
- Future Work

Equipment Specifications

Item	Characteristic
Operating System	Ubuntu Server 18.04 LTS
RAM	128 GB
Processor	Intel ® Xeon ® CPU E-1650 v4 12 Cores @ 3.60 GHz 15 MB Cache
Storage	1 TB
Graphics	4 x NVIDIA GTX 1080 Ti (CUDA Capable) 11 GB GDDR5X (each)
Dymola Distribution	Dymola 2020x
OM Distribution	1.14.0
Python Release	3.6.8
Dymola & OM Compiler	MinGW CC

Script Workflow

Iterate until all of the solvers & scenarios in both OM and Dymola have been simulated.



Simulation Automation Python Code Sample

```
for solver in solvers:
    # Printing info about current solver
    print("========================")
    print("CURRENT EXPERIMENT - SOLVER: {} - {}".format(experiment, solver))
    print("========================\n")
```

Single loop to perform all simulations

```
for n_tool, tool in enumerate(solvers[solver]):

    if n_tool == 0:
        # Current tool: Dymola
        tool_name = "Dymola"

        model_info = {'root_path': os.getcwd(),
        'library_path': "/OpenIPSL-1.5.0/OpenIPSL/package.mo",
        'model_path': "/OpenIPSL-1.5.0/OpenIPSL/IEEE14/package.mo", #Path to the package.mo
        'model_name': "OpenIPSL.IEEE14.IEEE_14_Buses_{}".format(n_exp + 1),
        'output_path' : os.path.join(os.getcwd(), "WorkingDir/Dymola/{}".format(solvers[solver][0]))}
```

Changing the simulation settings for each model automatically

Simulation Automation Python Code Sample

```
# Current solver
solver_name = solvers[solver][0]

# Creating a pool of processes
p = mp.Pool()

# List of running processes
process = []

# Measurement of execution time per simulation
t = time.time()
apfun = p.apply_async(dymola_simulation,
          args = (model_info, path_dymola, solvers[solver][0], True, ))
process.append(apfun)
p.close()
```

Creating a process to run the simulation using given settings

```
# Registering data until all processes are completed
while not process[0].ready():
    measure_performance(metrics[solver][tool_name], True, 0.2)
    #print("Simulation is running (and performance is being measured...)")
```

Measurement of performance in a parallel process

Complete code available in the GitHub repository

Overview

- Background
- Methodology
  - OpenIPSL Library
  - IEEE 14 Bus Power System with OpenIPSL
- Simulation Automation
- Simulation Results
- Performance Results
- Performance Scores
- Future Work

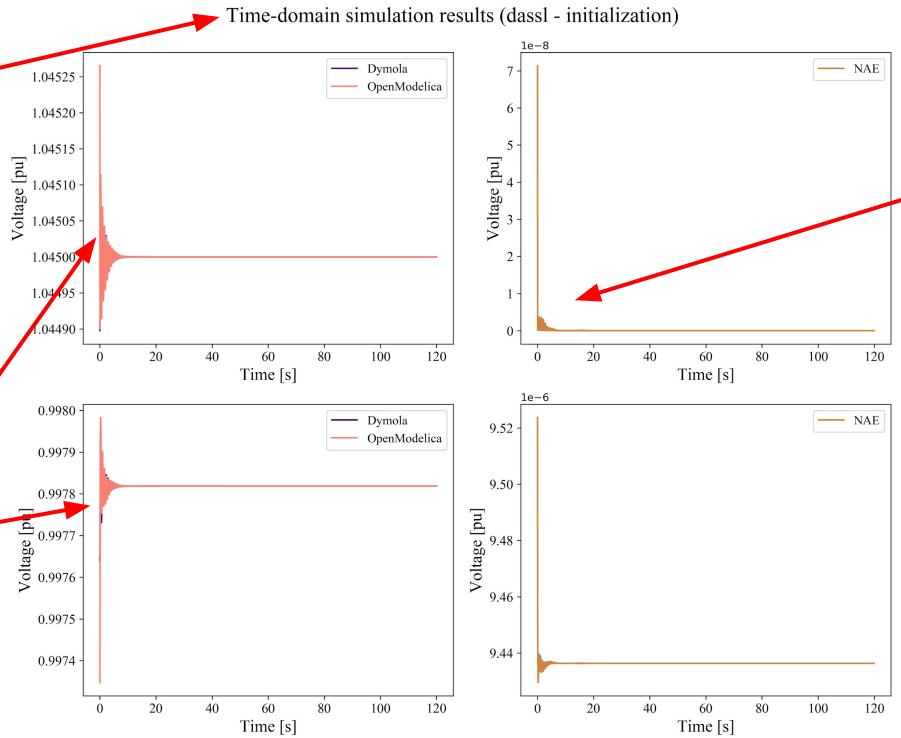
Simulation Results

For each scenario and for every solver, the simulation outputs:

Title, solver and scenario info about the simulation.

Voltage Signal for both OM and Dymola at different buses.

Error between the two signals (OM and Dymola).



Performance Results Tables

		Execution Time (seconds)		
		OpenModelica	Dymola	Result
Scenario 1: Initialization	dassl	7.869 s	0.1664 s	D > OM (47.3 x)
	euler	277.54 s	4420.01 s	OM > D (6.8 x)
	rk	783.01 s	1880.01 s	OM > D (5.6 x)
Scenario 2: Line Opening	dassl	13.4 s	0.3408 s	D > OM (39.3 x)
	euler	310.10 s	1850.01 s	OM > D (6.0 x)
	rk	1086.39 s	4410 s	OM > D (4.1 x)
Scenario 3: Bus Faults	dassl	163.48 s	14.40 s	D > OM (11.3 x)
	euler	378.6 s	1820.01 s	OM > D (4.8 x)
	rk	1344.68 s	4590.01 s	OM > D (3.4 x)

Performance Scores

A single metric was proposed to compute a single performance score of both tools with respect to all solvers and simulation scenarios. The basic score is known as **Normalized Minimum Execution Time (NMT)**

$$\mathrm{NMT}^{[\mathrm{scenario}]} = \frac{\min(\mathrm{ET_D}, \mathrm{ET_{OM}})}{\mathrm{ET_{observed}}}$$

Better performance for variable-step solver

	Dymola			OpenModelica		
	NMT[S1]	NMT[S2]	NMT[S3]	NMT[S1]	NMT[S2]	NMT[S3]
dassl	1	1	1	0.0211	0.0254	0.0880
euler	0.148	0.168	0.208	1	1	1
rk	0.177	0.296	0.293	1	1	1

Better performance for fixed-step solver

Future Work

- Scale up the experiments by selecting a larger power system.
- Perform the experiments enabling special features in Dymola/OpenModelica (e.g., DAE solver and sparse solvers).
- Evaluate the performance using the latest software releases (Dymola 2021 and OM > 1.1.4).
- Evaluate the same scenarios with the Nordic 44 model
- Test this work with different computers/equipment

Rensselaer

why not change the world?®