D5.1 – Initial DSF Hybrid Simulation Engine

Deliverable ID	**D5.1**
Deliverable Title	**Initial DSF Hybrid Simulation Engine**
Work Package	**WP5**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**22/02/2017**
Status	**final**
Type	**Prototype**
Lead Editor	**FIT**
Main Contributors	**ISMB, FIT**

Published by the Storage4Grid Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731155.

Document History

Version	Date	Author(s)	Description
0.1	2017-12-21	FIT	First TOC draft
0.2	2018-1-12	ISMB	Contribution on preparation
0.3	2018-1-26	ISMB	Contribution on revision
0.4	2018-2-2	FIT	Contribution on revision
1.0	2018-02-22	FIT	Ready for submission

Internal Review History

Review Date	Reviewer	Summary of Comments
2018-02-07 (v0.5)	Marta Sturzeanu (UPB) Mihaela Albu (UPB)	Approved
2016-02-13 (v0.6)	Marco Baldini (EDYNA)	Approved

Table of Contents

Executive Summary

This deliverable presents the first results of Task 5.1 "DSF Hybrid Simulation Engine", in which a framework for supporting professional scenarios and use cases of S4G has to be developed.

The deliverable presents the structure of the DSF-Simulation Engine framework designed to run complex power flow simulations in the grid of Bolzano and Fur. The framework consists of an API used for interfacing with external applications defining scenarios and for entering datasets modelling the grid. The framework coordinates the usage of different simulation engines if required, the simulation time and datasets necessary for each simulation and it retrieves results after the simulation is finished.

This is the first draft of the DSF-Simulation Engine Framework that will be developed in the S4G-Project. The framework will be tested and expanded in future works related to WP5. In general, the framework is built in a way that can be applied for testing and evaluating different scenarios developed in WP2 of the S4G-Project through its API.

1 Introduction

The deliverable 5.1 Initial DSF Hybrid Simulation Engine documents the first results of T5.1 whose main goal is to provide a tool able to analyse different scenarios featuring combinations of loads, energy storage systems and renewables in an existing grid.
Chapter 2 presents the structure of the first DSF-SE describing its components and presenting examples of its use. Chapter 3 describes the installation procedure of the DSF-SE and of the OpenDSS power flow solver. Chapter 4 shows software dependencies and requirements. Chapter 5 shows a preliminary API reference. Finally, chapter 6 discusses conclusions and next steps.

1.1 Scope

The goal of WP5 is the development of the S4G Decision Support Framework to support analysis, planning, forecast and optimization of storage system behaviour. Core of this development is the DSF-Simulation Engine, which allows the power flow analysis of the grid in different conditions. Therefore, a first version of the DSF-SE has been developed by Task T5.1 – "DSF Simulation Engine" and is described in this deliverable.
A further update of this prototype has to take into consideration developments made in T5.2 (DSF Interoperability with 3rd party and DSO systems) and T5.3 (DSF Hybrid Simulation Support). It is to be released as D5.2 Final DSF Hybrid Simulation Engine in M27.

1.2 Related documents

ID	Title	Reference	Version	Date
D2.1	Initial Storage Scenarios and Use Cases		V1.1	2017-06-08
D3.1	Initial S4G Components, Interfaces and Architecture Specification		1.0	2017-08-31
D6.1	Test Site Plans		1.0	2017-08-31
D5.3	Initial DSF Connectors for external systems and services		1.0	2017-09-04
D6.7	Initial Interfaces for Professional and Residential Users		1.0	2017-08-31

2 DSF-SE Prototype Overview

The overall, high-level structure of the DSF-SE prototype is depicted in Figure 1.

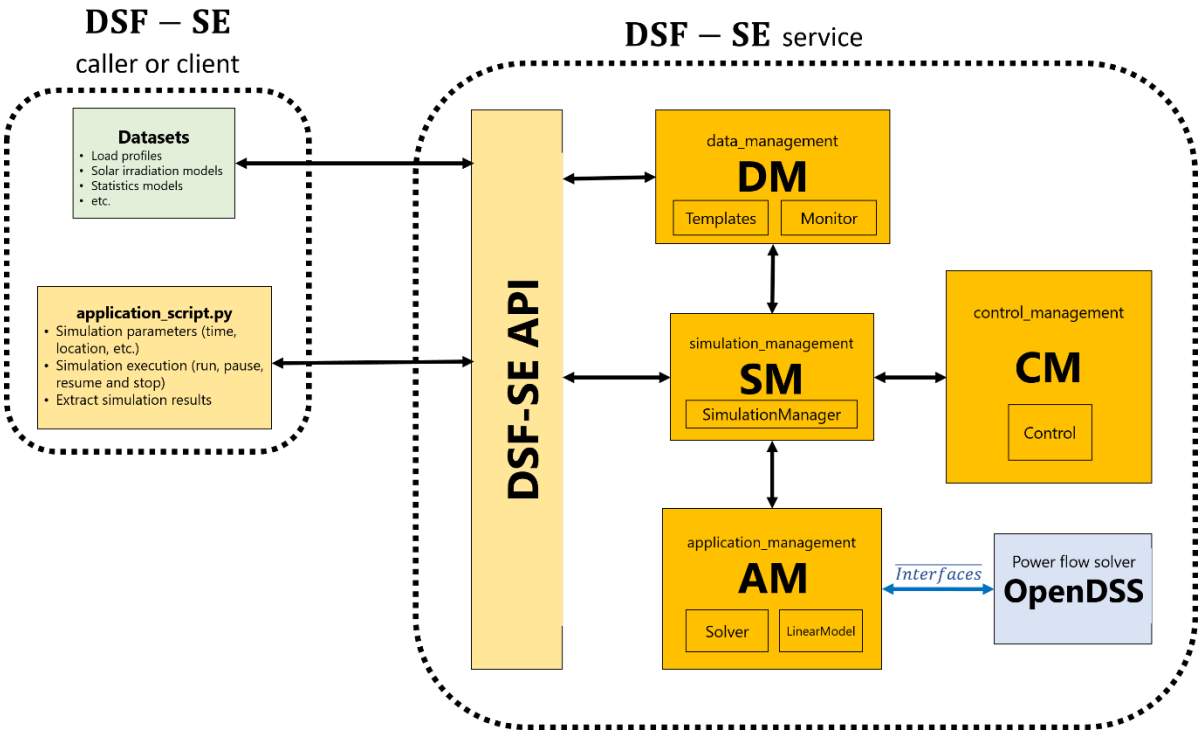

Figure 1 – The prototype structure

This co-simulation framework basically puts into service a scripting interface for the user and makes use of OpenDSS as its power-flow solver engine. The figure above represents different components in DSF-SE v.1.0, which are written in python.
The modular architecture of this framework allows creating several simulation instances, running them and evaluating the results.
The sub-components of the prototype are shortly summarized in the following:

2.1 Simulation Management (SM)

The *simulation_management (SM)* sub-component contains one class named *SimulationManager* that is the master core of the co-simulation framework. This sub-component is devoted to interface between DSF-SE and DSF-SE service caller (a simple script running on Windows in the first version) via its DSF-SE API.
The *SM* is in charge of:

1. coordination of other sub-components (modules). The SM synchronizes in this way the request and reply from other sub-components. When time is advancing and *Control module (CM)* needs to check the impact of a new configuration for instance, SM calls a power-flow function via the application object from *Solver* class.
2. control of the simulation time flow in different formats such as epoch, date time and day-night steps. For example, 01/01/2017 12:00 PM expressed in date time format is the same as 1483272000.0 in epoch and 720.0 in day-night steps (with step-size of 1 minute). This time updates in every simulation step and is used for advancing simulation run, reading correctly data from external files and publishing the result in a human readable format.
3. update of elements' attributes according to their profiles. Up to now, such elements are load profiles, solar irradiation and also statistical models for various simulation parameters.

2.2 Application Management (AM)

The *application_management sub-component (AM)* creates instances of the *Application* class, one for each different external power system software (at the moment only OpenDSS). In sequence, the AM creates custom objects from its interfaces.

AM includes two classes:

2.2.1 Solver

Solver is intended to incorporate power-flow solving functionality from various power system simulation software. In the current version, OpenDSS COM interface is integrated into the Solver *class* by instantiating the different interfaces that OpenDSS offers, as seen in Figure 2.

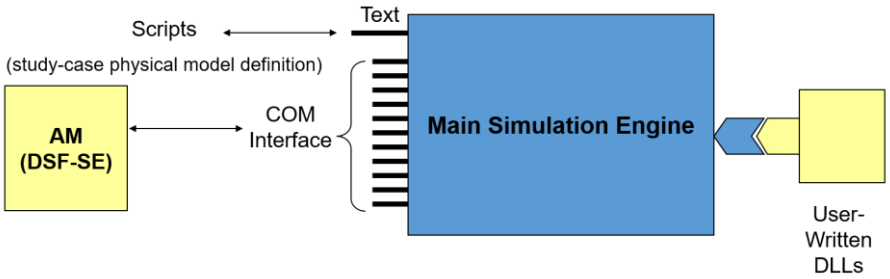

Figure 2 – OpenDSS interfaces with AM sub-component

2.2.2 LinearModel

This class is under development. The rationale behind it is to create an alternative for the cases in which simulation run is very time consuming. In the first version of DEF-SE a simulation of 24 hours with 1-minute resolution lasts around 9 seconds, that made us thinking to build a different option that provide a trade-off between accuracy and solving time.

2.3 Data Management (DM)

The *data_management* (DM) sub-component handles the available/provided dataset for the specific study case. This sub-component contains two classes:

2.3.1 Templates

A class named *Templates* uses existing data such as load profiles and statistical models for the relevant objects. More than 500 load profile templates for load profiles are available up to now from standards (UK market for instance) [1,2,3] or experimental resources (available open source files)[4].

2.3.2 Monitor

Monitor class will be writing and registering the variables of interest as well as results in every time step for every instance. The results are available once the simulation is over or paused.

2.4 Control Management (CM)

The *control_management* (CM) sub-component is where the artificial intelligence of the model will be laid and will emulate the decision making layer of the virtual system.
CM will be hosting several algorithms that become available for different objectives and scenarios, based on the entities' interests.
The following examples explain its capabilities:

1) A residential user owning a PV system might be interested in automating the storage system operation to maximize his self-consumption. This optimization relates to the algorithm implemented by a control class in the CM.
2) GESSCon requires the testing of its generated charging and discharging profiles of ESSs for the next 24h with different internal LESSAg algorithms to optimize diverse energy objectives (e.g. self-consumption maximization, voltage deviation minimization, etc.). Several classes implemented by the CM are in charge of scheduling the generated profiles using varying inputs, objectives, algorithms, and constraints.

2.5 DSF-SE API

The DSF-SE API is a RESTful interface that uses HTTP requests to GET, POST and DELETE data about grid elements and characteristics. The POST of a new group of transformers with their respective parameters constitutes an example of this interface. More information about the DSF-SE API can be found in chapter 5.

2.6 Example of use case simulation

For instance, to run a study-case simulation, the data including model file indication, simulation period and its geographical position are entered manually by the user or automatically by a higher-level system.

The simulation process is shown in flowcharts in Figure 3. A general case is shown on the left side whereas a specific S4G use case is shown on the right as an example.

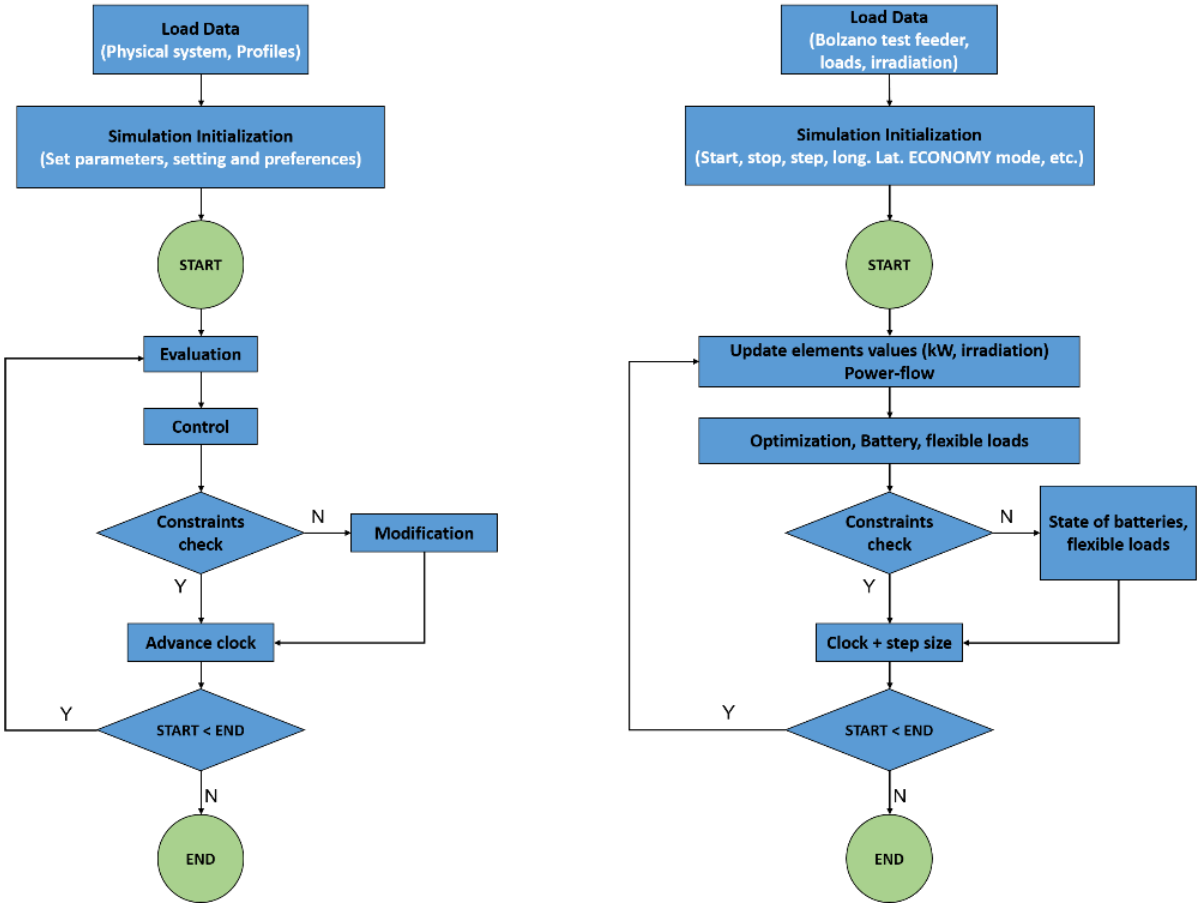

Figure 3 – Simulation process

```
project = 'main.dss'
SIMULATION_START = datetime(2017,6,18, 0, 0, 0, tzinfo=tz.utc)
SIMULATION_END =   datetime(2017,6,19, 0, 0, 0, tzinfo=tz.utc)
STEP_SIZE = 60
latitudine, longitudine = 46.497, 11.35   ### Test site location; Bolzano
```

Once the information is entered as shown before, the DSF-SE modules are imported and the simulation parameters are set:

```
import application_management.application as app
import simulation_management.simulation as sim
import data_management.library as lib
import data_management.monitor as mon
import control_management.control as ctrl

dss_app_object = app.Solver()
sim_object = sim.SimulationManager()
sim_object.set_simulation_parameters(SIMULATION_START, SIMULATION_END, STEP_SIZE, latitudine, longitudine)
dss_app_object.activate_use_case(my_dir, project)
lib_object = lib.Templates(dss_app_object, sim_object)
mon_object = mon.Monitor(dss_app_object)
ctrl_object = ctrl.Control('ECONOMY')
```

After the simulation parameters are set, the simulation assigns randomly load profiles to each existing load in the physical model by default, unless the load profiles have been specifically determined. The Bolzano test feeder contains 22 residential/agricultural consumers. These consumers are modelled as loads with constant PQ model, with the exact model of number of phases, connected phases and maximum contracted active power, but no load profiles associated to them, since there is no/enough information about them. However ones such data are available, an approximated (statistical) load profile can be set for each specific load. The ones without that (all of them for now) will be assigned with a randomly selected profiles from the *Template* class. These templates are scaled to per-unit value and during simulation run are multiplied by the maximum power of that consumer according to its contract. For example, **109211_s;** 109211 is the ID code used in EDYNA's technical scheme that refers to the residential costumer with house number 52A, and subscript **s** is indicating the connected phase to the feeder and here stands for phase *S* (ones without that are 3phase loads) . Its contracted maximum power is 6kW, so that load in OpenDSS assumes the values of the profile (7th row, 1st column) in the figure below, times per 6kW in each step.

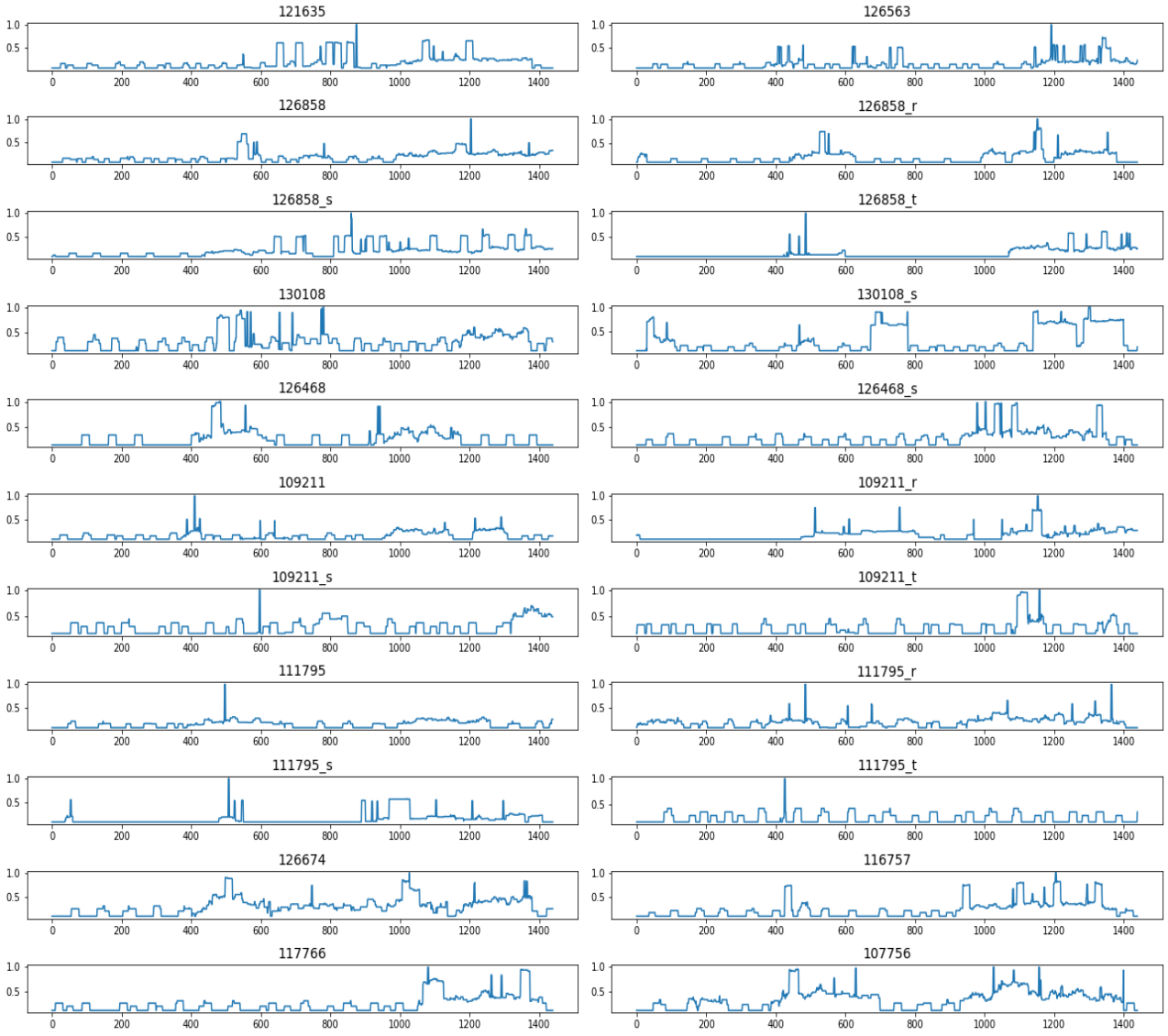

Figure 4 – Randomly selected and assigned load profiles

Irradiation profiles considering no cloud presence according to the simulation time and geographical location of photovoltaics (PV) are also created in *Templates* class and assigned to the existing PVs.

After these elements are set up, the simulation can initialize and run. The results are available to be visualized in the GUI and as csv files.

```
sim_object.initialize_simulation()
start = time()
while sim_object.simulation_status != 'ENDED':
    if sim_object.simulation_status == 'PAUSE': continue
    elif sim_object.simulation_status == 'STOP': break
    sim_object.runSimulation(dss_app_object, lib_object)
    ctrl_object.optimize(sim_object, dss_app_object)
    mon_object.writeResults(sim_object)
print('Simulation lasted ', time() - start, 'seconds')
print('DONE!')
```

Figure 5 depicts the results of a non-controlled (dumb) scenario conducted within the area of test feeder.

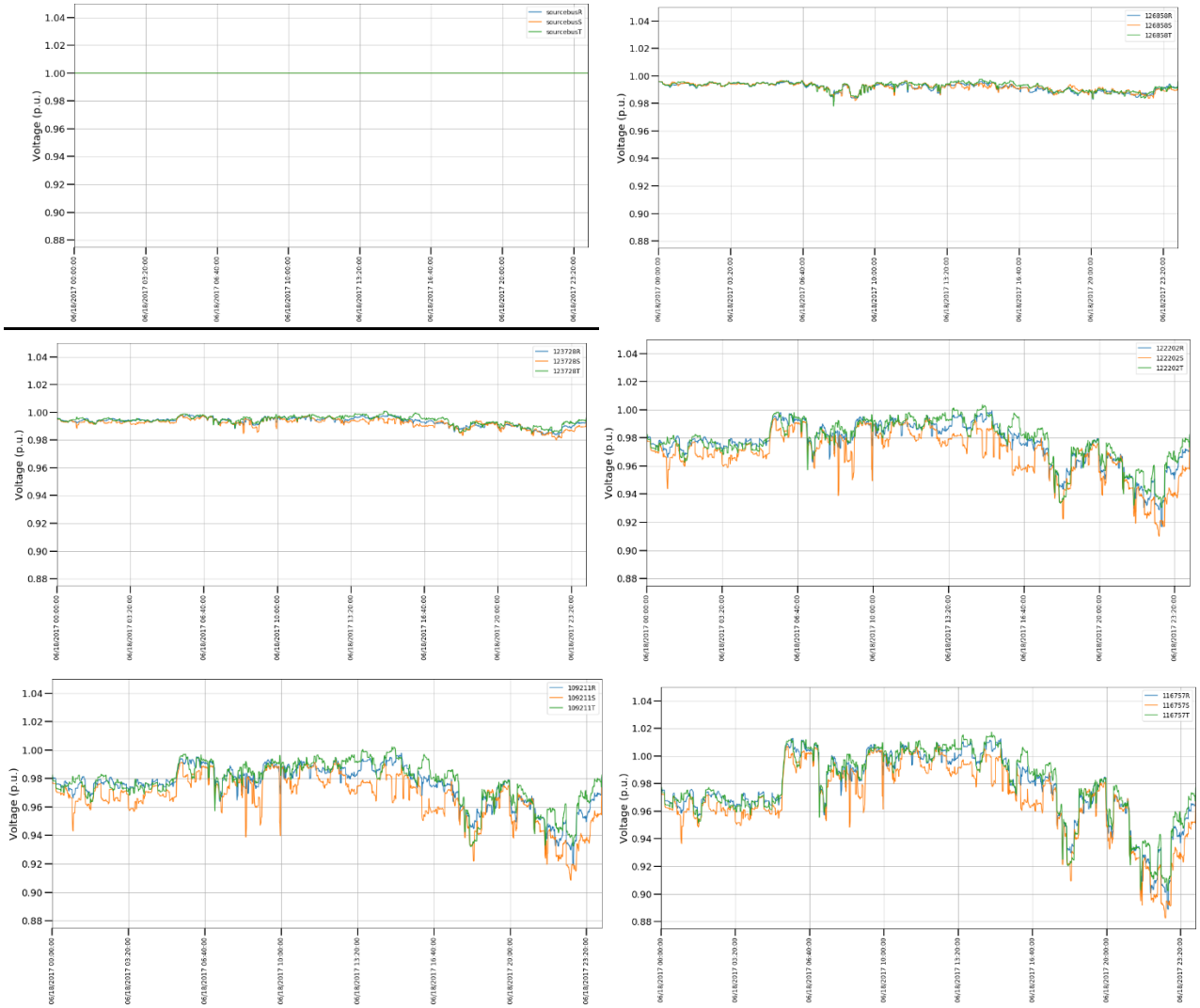

Figure 5 – Results example in term of nodal voltages

As expected, the results show that although the voltage level at substation is not influenced by load variation in downstream feeder, going towards ending point of the feeder voltage level drops, especially in peak hours.

3 Deployment instructions

3.1 OpenDSS

OpenDSS is a distribution system software tool which can be run under Windows operating system. A trade-off between reliability, simplicity, being licence-free and having integration capability with other simulation tools and programming languages makes it a good candidate for the power-flow solver of S4G.
An overview of evaluation for different power system simulation tools is available on project's Wiki.
The installation of OpenDSS requires the installation of an executable and optional files into a folder. The COM Interface of OpenDSS can be used through the OpenDSSEngine.DLL, which has to be previously registered into the system. The installer and the optional files can be found at https://sourceforge.net/projects/electricdss/files/OpenDSS/.

3.2 Hybrid Simulation Engine

The DSF-SE framework consists of python scripts that can be started from the console or by another high level software. The installation consists in this case of copying the files to a determined path of the system.

4 Software dependencies and requirements

Table 1 – Software Dependencies

Dependency	License	Role
Apache httpd, version 2.4.26	Apache version 2.0	Used to register and expose web-pages for the GUI, as well as proxying content over SSL channels
OpenDSS Version 7.6.5	Open source license	Power flow simulation engine
Python 3.6.4	GPL-compatible	Used for running the scripts that contain the framework

5 DSF-SE API Reference

The DSF-SE API[1] is a RESTful interface for controlling OpenDSS2 power flow solver and for introducing and fetching resources that this software needs for its operation. The API details these resources and the possible operations of OpenDSS in a human and machine-readable format, which is useful for discovery and integration.

[1] https://fit-bscw.fit.fraunhofer.de/bscw/bscw.cgi/d47992325/DSF-SE_API_V1.yaml
[2] https://sourceforge.net/projects/electricdss/

Besides, the API sets the data model of the resources facilitating its usage even for further development outside S4G-Project.
The resources that are allowed to be introduced or fetched as part of the grid simulation are the following:

- Transformers
- Nodes as points of energy injection and withdraw in the system
- Power lines for interconnecting the nodes
- Load profiles for defining a 24h electricity consumption pattern
- Photovoltaic devices definition
- Generation profiles for photovoltaic devices
- Storage systems definition
- Charging and discharging profiles for energy storage systems for the next 24h calculated in GESSCon module
- Charging profiles for electric vehicles for the next 24h calculated in GEVChCon
- Commands for controlling and getting results from OpenDSS:
  - Start simulation
  - Stop simulation
- Fetching results of simulations
- Introducing new short-time management algorithms for energy storage systems
- Introducing new short-time management algorithms for charging of electric vehicles

6 Conclusions

The present deliverable provides the first version of DSF-SE framework, which empowers DSOs and energy entities to optimize the short/mid-term planning of the grid by taking into account new technologies such as energy storage systems.

After a general study, Python was chosen as development tool for the DSF-SE framework and OpenDSS as the main power system simulation software for running power-flow simulations. The modular architecture of the DSF-SE allows the user to analyze several scenarios and to compare the results for getting an optimized solution. As physical model, model of Bolzano test feeder in OpenDSS is the base of DSF-SE simulation, evaluation and benchmarking in development stage.

In the next step, the DSF-SE framework will be extended for communicating with external hardware components during a simulation process allowing the evaluation of a hybrid simulation environment.

Acronyms

Acronym	Explanation
DSF-SE	Decision Support Framework Simulation Engine
SM	Simulation Management
DM	Data Management
AM	Application Management
CM	Control Management

List of figures

List of tables

References

[1] OpenDSS, distribution system analysis software: https://sourceforge.net/projects/electricdss/
[2] UK energy market standard models: https://rmdservice.com/standard-load-profiles/
[3] DIgSILENT PowerFactory (power system analysis and planning software): https://www.digsilent.de/en/powerfactory.html
[4] Energy data and information from OpenEI resources: https://openei.org/datasets/files/961/pub/