INTERPLAN
INTEgrated opeRation PLAnning tool towards the Pan-European Network

Work Package 6

INTERPLAN model validation and testing

Deliverable 6.2

Documentation of the data model and interfaces used in INTERPLAN

Grant Agreement No:	**773708**
Funding Instrument:	**Research and Innovation Action (RIA)**
Funded under:	**H2020 LCE-05-2017: Tools and technologies for coordination and integration of the European energy system**
Starting date of project:	**01.11.2017**
Project Duration:	**36 months**

Contractual delivery date:	**31.10.2019**
Actual delivery date:	**20.12.2019**
Lead beneficiary:	**5 Fraunhofer Gesellschaft zur Förderung der angewandten Forschung e.V.**
Deliverable Type:	**Report (R)**
Dissemination level:	**Public (PU)**
Revision / Status:	**RELEASED**

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 773708

Document Information

Document Version:	1.0
Revision / Status:	RELEASED
All Authors/Partners	Jan Ringelstein / IEE Tobias Banze / IEE Saber Talari / IEE Ata Khavari / DERlab Roberto Ciavarella / ENEA Marialaura Di Somma / ENEA Anna Wakszyńska / IEn Michał Kosmecki / IEn Michał Bajor / IEn Minas Patsalides / FOSS Christina Papadimitriou / FOSS Venizelos Efthymiou / FOSS Sohail Khan / AIT Sawsan Henein / AIT
Distribution List	INTERPLAN consortium
Keywords:	Showcase, Scenario, Simulation, Verification

Document History

Revision	Content / Changes	Resp. Partner	Date
0.1	Structure creation	IEE	16.01.2019
0.5	Refinement of the section 3.1	FOSS	28.11.2019
0.6	Wrapped up all sections, removed editor's comments	IEE	29.11.2019
0.7	Added executive summary and conclusion. Prepared version for internal review.	IEE	6.12.2019
0.8	Minor corrections	FOSS	07.12.2019
0.9	Corrections to section 3.6	AIT	06.12.2019
1.0	Chapter 2 added, cleaned version after reviews by IEn and AIT	IEE	18.12.2019

Document Approval

Final Approval	Name	Resp. Partner	Date
Review WP Level	Anna Wakszyńska	IEn	11.12.2019
Review Management Level	Helfried Brunner Marialaura di Somma Giorgio Graditi	AIT ENEA	17.12.2019 20.12.2019

Disclaimer

This document contains material which is copyrighted by certain INTERPLAN consortium parties and may not be reproduced or copied without permission. The information contained in this document is the proprietary confidential information of certain INTERPLAN consortium parties and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information in this document may require a licence from the proprietor of that information.

Neither the INTERPLAN consortium as a whole, nor any single party within the INTERPLAN consortium warrant that the information contained in this document is capable of use, nor that the use of such information is free from risk. Neither the INTERPLAN consortium as a whole, nor any single party within the INTERPLAN consortium accepts any liability for loss or damage suffered by any person using the information.

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

Copyright Notice

Table of contents

Abbreviations

AMPL	*A Mathematical Programming Language*
API	*Advanced Programming Interface*
CSV	*Comma Separated Values*
DER	*Distributed Energy Resource*
DG	*Distributed generation*
DR	*Demand response*
DRES	*Distributed renewable energy sources*
DPL	*DIgSILENT Programming Language*
DSL	*DIgSILENT Simulation Language*
DSO	*Distribution System Operator*
EAC	*Energy Authority of Cyprus*
EHV	*Extra High Voltage*
EU	*European Union*
EV	*Electric vehicle*
fFRC	*Fast Frequency Restoration Control*
GUI	*Graphic User Interface*
HV	*High voltage*
IP	*Internet Protocol*
IPOPT	*Interior Point Optimizer*
JSON	*Java Script Object Notation*
KPI	*Key Performance Indicator*
LV	*Low voltage*
MCP	*Master Control Program*
MQTT	*Message Queue Telemetry Transport*
MV	*Medium voltage*
OLTC	*On-Load Tap Changer*
OPF	*Optimal Power Flow*
PFD	*PowerFactory Datafile*
PV	*Photovoltaic*
RES	*Renewable energy sources*
RoCoF	*Rate of Change of Frequency*
SQL	*Structured Query Language*
TSO	*Transmission System Operator*
UC	*Use case*
WP	*Work Package*
XML	*eXtended Markup Language*

Executive Summary

The INTERPLAN project looks at the potential operation challenges which TSOs and DSOs are called to address in the 2030+ power system within the pan-European Network. The project focus is on TSO and DSO grid operation and operation planning, as well as TSO-DSO interfaces to support the EU in reaching the expected low-carbon targets, while maintaining network security and reliability. To this end, INTERPLAN has defined an INTEgrated opeRation PLAnning tool which is a three stage methodology consisting of a set of tools (grid equivalents, control functions) for grid operation planning. Also, INTERPLAN has defined five showcases. Each showcase combines one or two use cases, grid model, scenario, KPIs, simulation environment, simulation type, time-series data, grid operation planning criteria, and control functions. Control functions enable for grid operation planning according to the showcase's criteria. For each of the five showcases, there is a base showcase without control functions and planning criteria which serves as a reference case. The showcases are demonstrated and validated by means of simulation, and assessed by comparison of KPIs of base showcases and showcases.

The document at hand presents a description of the interfaces and means used for data transmission for implementing the showcase control functions and performing the simulations. Also it presents a set of software tools with according interfaces which can be used to prepare and execute a co-simulation for INTERPLAN. The co-simulation is planned to be used to test the application of the INTERPLAN tool.

The project relies on predefined or proprietary data formats, data models and interfaces. The main types of interfaces described herein relate to interfaces between software systems that are used to develop, test, or simulate control functions. Most of these functions need two types of data as inputs: grid models and time series. Such data is thus very central to the project and is provided in different formats. For the grid models, there are generally two options:

1. The internal data format of the grid calculation software PowerFactory by the manufacturer DIgSILENT and,
2. The JSON format supported by pandapower 2.0, which is a grid calculation software previously developed by Fraunhofer IEE and University of Kassel.

For conversion of the pandapower JSON format to the PowerFactory format, an automatic converter is used which was further developed for project INTERPLAN.

For time series, no specific data model or format is used as it is convenient enough to store them either in CSV or Excel files. Those files can be read or written by Python scripts, which are broadly used by developers in the project in order to implement control functions, automatize simulation processes, and convert, record and process data. Many of the projects showcases also use a Python/PowerFactory advanced programming interface which is provided by DIgSILENT in order to exchange data between Python scripts and grid model. Also, time series data can be stored directly in the PowerFactory internal data format.

Further interfaces and proprietary data formats are used to connect control functions to additional software, most prominently the mathematic programming language AMPL. Also, many project showcases [1] include two use cases [2] and hence at least two separated control functions which are executed subsequently. This again creates the need to transmit information between them, which

particularly relates to setpoints for grid assets calculated by the control function first executed and used by the second control function as an input. Those data formats are proprietary and have been developed by showcase teams, and mainly rely on Excel tables, CSV files or dat files for information exchange. The same is the case for results storage.

Furthermore, a toolset for co-simulation of INTERPLAN solutions has been developed. The co-simulation is based on the OpSim framework previously developed by Fraunhofer IEE and University of Kassel. OpSim provides its own data model and interfaces in form of “Client/Proxy” modules, which were found to be suitable for the project requirements, thus are used unchanged. Named toolset mainly uses Excel and CSV files in order to obtain input data for a simulation, and stores resulting time series in a local database which contains one table per physically measurable unit, e.g. the grid asset’s active power. Communication between OpSim subsimulations relies on message queue telemetry transport (MQTT) and uses the existing OpSim data model. In order to define a simulation setup, an automatically generated XML data file is used with a format defined by OpSim. A first co-simulation of an INTERPLAN control function was successfully executed as a proof-of-concept. This paves the way for validating the INTERPLAN tool by the example of one of the showcases.

1. Introduction

1.1 Purpose and scope of the Document

INTERPLAN has defined an INTEgrated opeRation PLAnning tool which is a three stage methodology consisting of a set of tools (grid equivalents, control functions) for grid operation planning. Also, INTERPLAN has defined five showcases which are a combination of use cases, grid model, scenario, KPIs, simulation environment, simulation type, time-series data, grid operation planning criteria, and control functions. Control functions enable for grid operation planning according to the showcase's criteria. For each of the five showcases defined in INTERPLAN, there is a base showcase without control functions and planning criteria which serves as a reference case. The base showcases and showcases are validated and tested by means of simulation.

This document falls in scope of INTERPLAN project activities around this model validation and testing (work package 6). It presents a description of the interfaces and means used for data transmission for implementing the control functions and performing the simulations. Also it presents a set of software tools with according interfaces which can be used to prepare and execute a co-simulation for INTERPLAN. The co-simulation is used to test application of the INTERPLAN tool.

1.2 Structure of the Document

This document is structured as follows. Chapter 2 briefly introduces the INTERPLAN project. Chapter 3 focuses on data and interfaces related to grid models and time series. It particularly introduces the requested grid data format conversion from pandapower to DIgSILENT PowerFactory formats. Chapter 4 discusses interfaces and data transmission used for implementing the five INTERPLAN showcases [3]. Chapter 5 presents a co-simulation toolset which was prepared for validation of INTERPLAN control functions and tools. Chapter 6 concludes the document.

2. INTERPLAN project

The European Union (EU) energy security policy faces significant challenges as we move towards a pan-European network based on the wide diversity of energy systems among EU members. In such a context, novel solutions are needed to support the future operation, resilience and reliability of the EU electricity system in order to increase the security of supply and also accounting for the increasing contribution of renewable energy sources (RES). The goal of the INTERPLAN project is to provide an INTEgrated opeRation PLAnning tool towards the pan-European Network, with a focus on the TSO-DSO interfaces to support the EU in reaching the expected low-carbon targets, while maintaining the network security and reliability.

INTERPLAN project looks at the potential operation challenges which TSOs and DSOs are called to address in the 2030+ power system. In fact, the ongoing deployment of the pan–European network strongly depends on different potential scenarios related to the RES share in generation and installed capacity, as well as penetration of emerging technologies, such as storage and Demand Response (DR). Although these factors represent the preferential patterns to meet the EU decarbonized energy targets for 2030 and 2050, they bring new challenges for the energy system, which will outline the key operational needs of the European grid operators in the near future.
In such a context, TSOs will need to evolve progressively from a “business as usual approach” to a proactive approach in order to avoid a bottleneck effect in the future European grid, and this could be addressed through a proper system operation planning. As for the distribution networks, they have been traditionally designed and treated to transport electrical energy in one direction, i.e. from the generation units connected to the transmission system to the end-users. However, with the growing share of non-dispatchable distributed generation, customers are increasingly generating electricity themselves, and by becoming “prosumers”, they are shifting from the end point to the center of the power system. As a result, DSOs will need to actively manage and operate a smarter grid through appropriate system control logics, by utilizing the flexibility potential in the grid, with the aim to optimize the distribution network performance. Furthermore, an additional critical issue is the interface between transmission and distribution systems, which is expected to evolve in the near future through a mutual cooperation between TSOs and DSOs, with the aim to address operational challenges as congestion of transmission and distribution lines and at the interface among them, voltage support between TSOs and DSOs, and power balancing concerns. The increasing complexity of the grids requires control and operation planning tools even more advanced and homogenous among European countries.

With these premises, the INTERPLAN idea was born. In such a framework, the projects aims to develop control system logics which suit the complexity of the integrated grid, while managing all relevant flexibility resources as “local active elements” in the best manner. Moreover, by looking at the 2030+ power system, the project also addresses policy and regulation aspects aiming to identify a set of possible amendments to the existing grid codes, reflecting the developments achieved in INTERPLAN through its tool, use cases and showcases. The aim of this analysis is to break down the current barriers to the integration of emerging technologies and to foster TSO-DSO cooperation in managing grid operation challenges.

In detail, a methodology for a proper representation of a “clustered” model of the pan-European network is provided, with the aim to generate grid equivalents as a growing library able to cover a number of relevant system connectivity possibilities occurring in the real grid, by addressing a number of operation planning issues at all network levels (transmission, distribution and TSO-DSO

interfaces). In this perspective, the chosen top-down approach leads to an “integrated” tool, both in terms of voltage levels, moving from high voltage level down to low voltage level and end user, as well as in terms of developing a bridge between static, long-term planning and operational issues considerations, by introducing proper control functions in the operation planning phase. Therefore, in the project, novel control strategies and operation planning approaches are investigated in order to ensure the security of supply and resilience of the interconnected EU electricity power networks, based on a close cooperation between TSOs and DSOs, thereby responding to the crucial needs of the ongoing pan-European network and its operators.

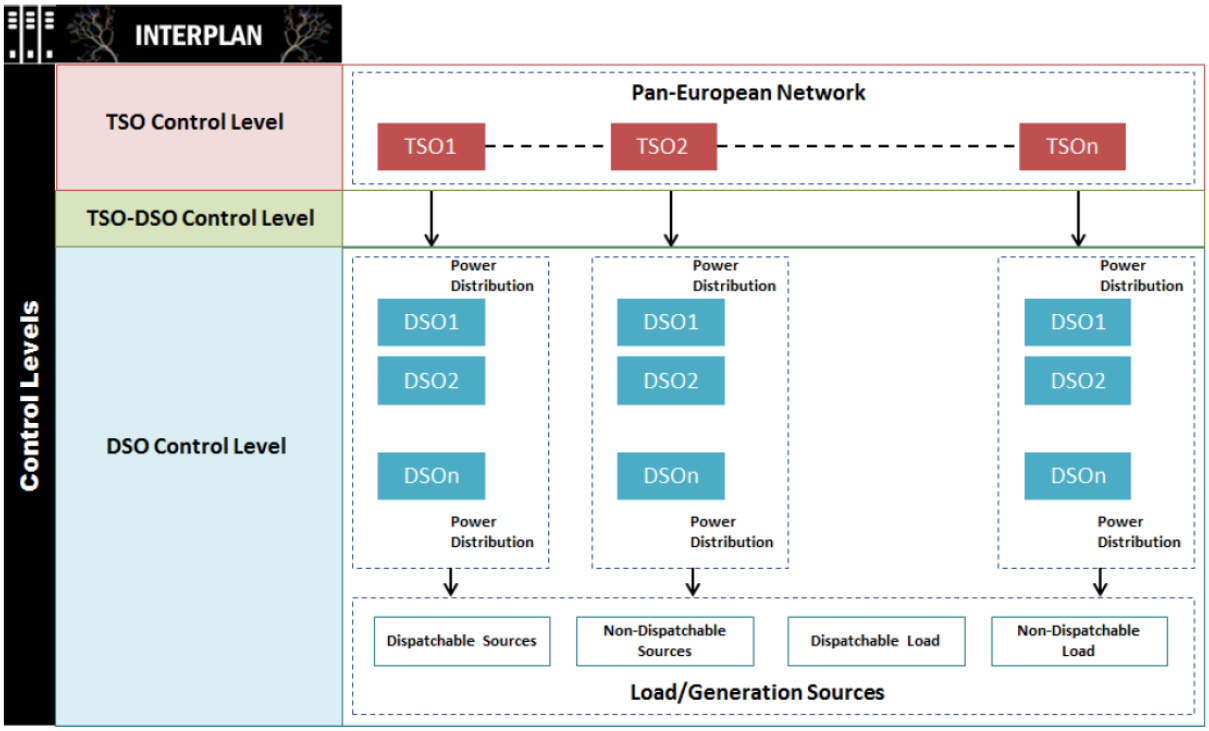

Figure 1: INTERPLAN concept

3. Data models and interfaces for grid models and time series

3.1 Grid model format conversion

A very central type of data in INTERPLAN project are grid models. For the creation of the grid models used for three out of five INTERPLAN showcases, the open source software “pandapower” [4] was used. It is a Python based tool for electrical grid calculation and analysis. This software combines the “pandas” and the “PYPOWER” Python packages and was developed in cooperation between University of Kassel and Fraunhofer IEE. The basic data to build up pandapower INTERPLAN grid models originates from the SimBench project [5]. In this project the Universities of Kassel, Aachen and Dortmund together with Fraunhofer IEE developed benchmark datasets for the analysis, planning and operation of electrical networks. For an easy use of the data from SimBench, pandapower is a suitable choice.

On the other hand for development of showcases and control functions, the favored software for grid processing and simulation is DIgSILENT PowerFactory [6]. PowerFactory defines a grid model file format which is different from the formats supported by pandapower; moreover PowerFactory can not import the pandapower format. Thus, a file format conversion was applied to make pandapower networks available in PowerFactory. For this task a converter which is implemented in the “pro”-version of pandapower was used. This converter iterates through all grid components and translates them into the PowerFactory data structure. After the conversion, a new project is created in the PowerFactory software which represents the pandapower grid model. When publishing this report the converter is still in development and available in a beta version. In order to convert all relevant data from the INTERPLAN networks, some adaptations were made to the converter.

The biggest adaptation was to implement code for the conversion of rotating generators. The already existing converter was modified as follows. First, creation of rotating generator standard types which are needed in PowerFactory to build this type of generator was added. Furthermore, a query was added to the static generator converting part, which set the rated apparent power to zero if the parameter is undefined in the pandapower grid model; such would otherwise result in a failure. Finally, an adaptation of the naming of grid components was applied. Before this change the converter named all components with their pandapower name plus the pandapower array list index in brackets. This additional index at the end of the component names led to problems during the calculation of time series, hence it was removed.

A graphical result example of the conversion is shown in Figure 2. It refers to the eastern Germany network model planned to be used for validation of INTERPLAN showcases 2,3 and 4. The northern part of the same network was prepared beforehand in order to allow development of base showcase simulations.

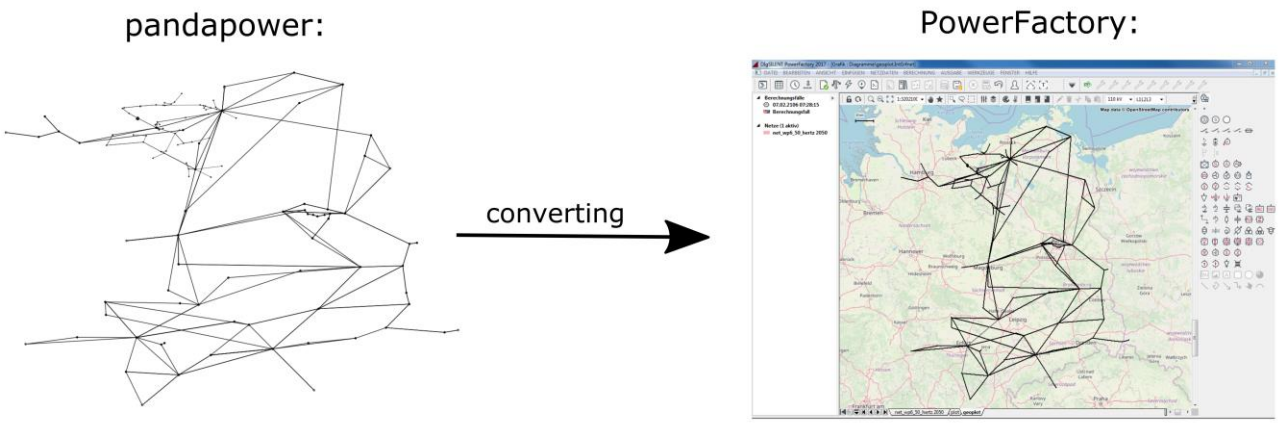

Figure 2: Conversion of a pandapower grid model into PowerFactory

3.2 Time series generation formats

Time series data define the operation of all loads, generators and other assets within a grid model. They define the active power in-feed or consumption of the assets and represent another central data type needed for simulations and development of control functions. To create time series, input data in form of daily profiles based on historical or forecasted data is needed. A complete set of data for one year would contain profiles for 365 days for photovoltaic (PV) and wind (onshore and offshore) generation, as well as information about energy consumption (loads). Data for each type should be stored in a separate *.txt file and the file structure should be as shown in Figure 3: Each row indicates one day in a year and each column refers to an active power value (in pu) for a given time step.

#DAY/HOUR	00:00	00:15	00:30	00:45	01:00	01:15
2018-01-01	0.17	0.19	0.18	0.18	0.15	0.12
2018-01-02	0.23	0.24	0.24	0.23	0.23	0.21
2018-01-03	0.13	0.14	0.16	0.16	0.19	0.19
2018-01-04	0.04	0.04	0.04	0.04	0.03	0.03

Figure 3: Example section of daily profiles containing active power values in p.u.

The other input data necessary is information on grid model elements, particularly name and maximum active power; additionally minimum active power in case of synchronous generators. This data must be available in the grid model and can be obtained through a dedicated Python script. The grid model containing this data needs to be imported into PowerFactory for that. Figure 4 shows an example for such data.

```
EHV Sgen 70.ElmGenstat   86.59201049804688 :
EHV Sgen 112.ElmGenstat   27.06000328063965 :
EHV Sgen 127.ElmGenstat   22.632009506225586 :
EHV Sgen 128.ElmGenstat   22.632009506225586 :
EHV Sgen 133.ElmGenstat   52.47999572753906 :
```

Figure 4: Example section of data read from grid model (here onshore wind turbines)

When all necessary input is ready, time series can be produced through a Python script for each load and generator in a given grid model. Resulting time series data are kept in single CSV file,

which can be then imported into the PowerFactory project. Figure 5 gives an example section from a resulting CSV file. It contains the active power in MW for each generator and timestep.

```
EHV Sgen 70;40.75020014038086;41.97980668945313;41.11388658447266;42.1
EHV Sgen 112;12.707377540588379;12.658669534683227;13.335169616699218;
EHV Sgen 127;10.451461989974975;10.057665024566651;11.096474260902404;
EHV Sgen 128;10.469567597579957;9.96034738368988;9.804186518096923;10.
EHV Sgen 133;25.47378992614746;24.9227499710083;26.397437850952148;25.
```

Figure 5: Example section of output file

The CSV file such obtained can be read into PowerFactory as many times as needed (e.g. for different projects using the same grid model), where the time series are used for automatic simulations. Also, they can be used for the co-simulation as described in chapter 5.

4. Data formats and interfaces for tools and control functions

4.1 Overview of use cases and showcases

For further reference, Table 1 lists all INTERPLAN use cases and Table 2 lists the showcases, which may include one or two use cases [3]. The following sections detail the data formats and interfaces for each of the five showcases.

Table 1: INTERPLAN use cases

Use Case No.	Use Case Name	Used in Showcases
1	Coordinated grid voltage / reactive power control	2, 4
2	Grid congestion management	2
3	Frequency tertiary control based on optimal power flow	3, 4
4	Fast frequency restoration control	1
5	Power balancing at DSO level	3
6	Inertia management	1
7	Energy interruption management	5

Table 2: INTERPLAN showcases

Show Case No.	Show Case Name	Included Use Cases
1	Low inertia systems	4, 6
2	Effective DER operation planning through active and reactive power control	1, 2
3	TSO-DSO power flow optimization	3, 5
4	Active and reactive power flow optimization at transmission and distribution networks	1, 3
5	Optimal energy interruption management	7

4.2 Showcase 1: Low inertia systems

Showcase 1 combines fast frequency restoration control (UC4) with inertia management (UC6) in order to maintain frequency stability in low inertia systems. Input data for this showcase is a grid model in PowerFactory PFD format, with time series of one day up to one year incorporated into it. In the grid model, all network elements are modelled including transformers, transmission and distribution lines, generators, distributed resources and storages, depending on the study case being investigated. The dynamic behaviours of the different power production and consumption technologies are incorporated into the elements via DSL/Python. In addition, the control dynamics are also implemented in PowerFactory compatible format (DSL/Python scripts) based on the defined control diagrams, flows, circuits, and transfer functions, and inserted into the grid model. Current regulations and future requirements are considered (as for example local voltage regulation) by adding the required functionality on network elements to achieve a realistic behaviour of the power grid and produce results of high quality and significance. The grid model includes five control areas on which primary and secondary controllers are attachable. The ability to attach controllers on each area can enable the discovery of advanced and intelligent controllers able to maintain the power parameters into desirable range and ensure the safe and stable operation of the power grid. One of

the five areas models conventional power generation. Another area models the wind power generation, while the three remaining areas model distribution grids with different types of distributed power resources and storage.

The output data is stored in CSV files. One or more files are produced for each simulation scenario in order to enable the analysis of data with external software tools. In each file, results for the main power quantities are stored. Amongst the important parameters monitored and captured are the system frequency, the voltages of important buses of the power grid, and the rate of change of frequency (ROCOF). The parameter transients are evaluated for further analysis. From the produced results, plots for system frequency and frequency nadir are created in order to assess system stability as well as frequency restoration control efficacy. Moreover information on the KPIs “Level of distributed generation (DG) / distributed renewable energy sources (DRES) utilization for ancillary services” and “Share of RES” will be accessible.

The following sections detail the data transfer for both use cases involved.

4.2.1 Inertia management (UC 6) control function data

The inertia control function uses internal information from the PowerFactory grid model, such as inertia constants, active powers, and nominal powers. As depicted in Figure 6, the control function is comprised of several phases. In a first phase (steps 1 and 2 in the figure), the system’s kinetic energy and its need for additional inertial support is calculated by a Python script. Also, a choice of power system objects for inertial support is made.
In the following phase (step 3 in the figure), PowerFactory is used for executing a dynamic simulation and exporting the results to CSV files. Afterwards, a results post-processing is applied by a Python script. The post-processing result is a set of plots.

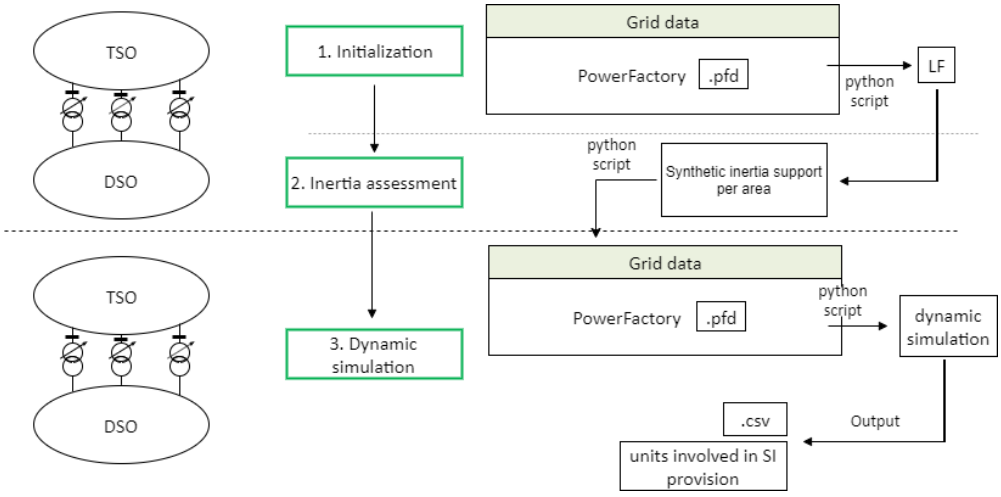

Figure 6: UC6 flowchart and information exchanged

4.2.1 Fast frequency restoration control (UC 4) control function data

The steps for implementation of the fast frequency restoration control function are shown in Figure 7.

With the first step implemented by a Python script, a quasi dynamic simulation is performed in order to calculate the total power-frequency characteristic per each control area of the power grid under analysis. Thus, based on the assets flexibility information, the frequency droop per each device is calculated. The Python code uses a specific Python module to interact with PowerFactory simulation environment and to obtain the information about the assets active power flexibility. For this, the grid model in PowerFactory format is used. The calculated frequency droop per each device is used as input setting for the assets' dynamic models in the PowerFactory environment. Moreover, an instability event is configured in the Python code and used for the dynamic simulation in the second step.

With the second step relying on PowerFactory, a dynamic simulation of the aforementioned instability event is performed. This is done in order to locate the instability event in the grid and verify the effectiveness of the devices frequency droop calculated at step one. The results are stored for further analysis. Also, information on power system objects used for inertial response are noted per simulation case.

After step 2, post processing of the results is applied.

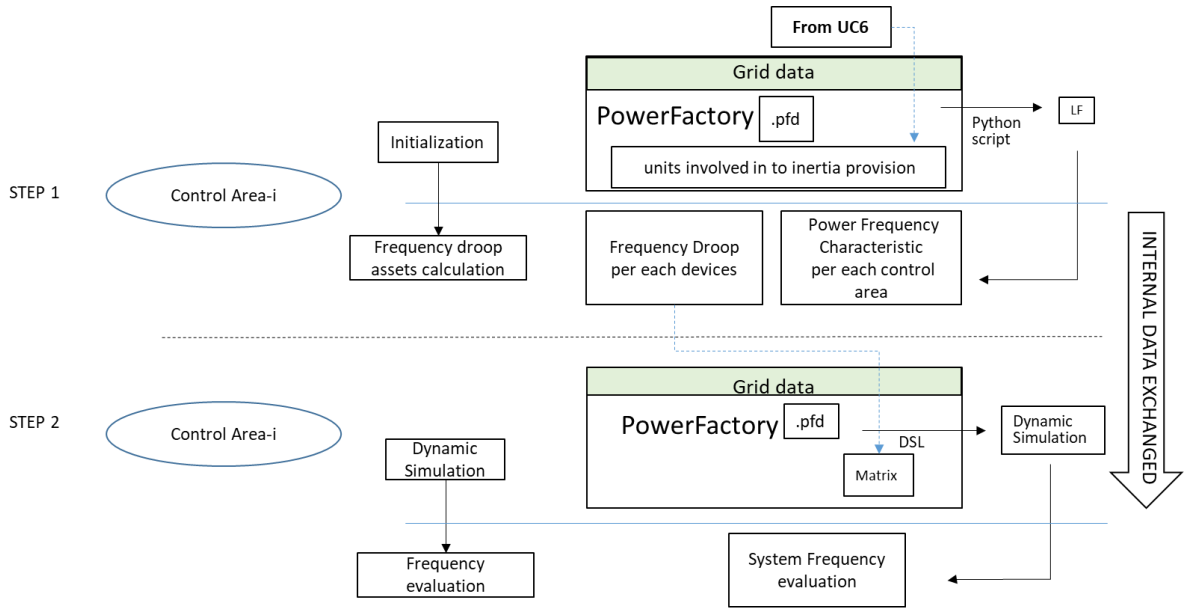

Figure 7: UC4 flowchart, information exchanged and interaction with UC6

4.3 Showcase 2: Effective DER operation planning through active and reactive power control

Showcase 2 combines grid congestion management (UC2) with coordinated grid voltage / reactive power control (UC1) in order to implement mitigate grid congestion while ensuring voltage stability. The use cases are executed subsequently and use different control functions. Due to this, both internal and external information exchange is necessary for each individual use case. The external information exchange especially relates to transmitting UC2 control function results to UC1.

4.3.1 Grid congestion management (UC2) control function data

The UC2 controller is developed in Python code and uses a specific Python module to interact with a grid model implemented by the PowerFactory simulation environment. The grid model is thus provided in the PowerFactory format as an input data to the UC2 control function. It can also be used to test named control function.
Figure 8 shows internal information of the UC2 control function, as related to the data exchange with the PowerFactory grid model.

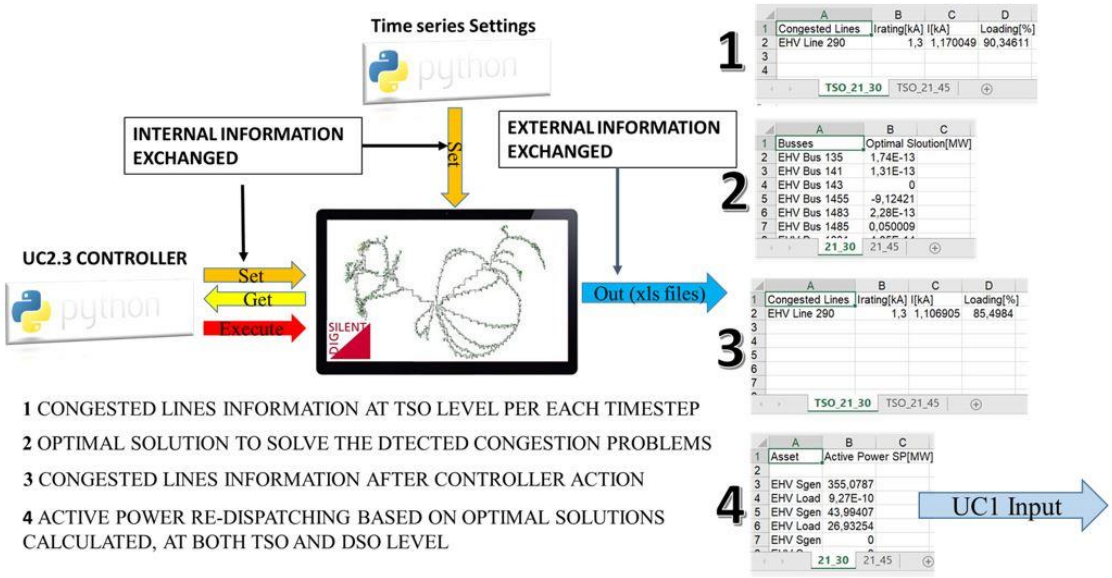

Figure 8: UC2 internal and external exchanged information

This data exchange is enabled through an interface provided by PowerFactory. Through this the Python script for UC2 can obtain the necessary information to solve the possible congestion problems detected in the grid directly from the grid model.
Through the “Set” function, it is possible to set or modify all the electrical parameters in the grid, such as the assets’ flexibility or the time series value for a specific time step. With the “Get” function it is possible fetch the parameter information from the grid model. Finally, the “Execute” function allows performing all the calculation functions available within PowerFactory environment (Load Flow, Sensitivity Analysis, etc). The intermediate results of the UC1 execution depicted in Figure 9 are as follows:

1. A list of congested lines with their loading, fetched from PowerFactory by the “Get” function
2. An optimal attribution of active power values to the buses in the network which aims at eliminating the line overload (in this case a line is considered overloaded if the loading exceeds 90%)
3. A list of changed loadings of the previously overloaded lines and
4. An attribution of active power setpoints to generators connected to the buses found in step (2), aimed at providing the needed optimal active power.

Another important internal information exchange is related to the time series. The time series data is a key part of the definition of the INTERPLAN scenario to analyse. The time series information is

set in the grid model using a Python script, using the same “Set” function as mentioned above.
The aforementioned outputs 1..4 are created after control function execution and stored into Excel files. They thus also represent information that can be exchanged externally. However, the UC1 control function only uses the Excel file related to the active power re-dispatching (output number four) as input. This file provides the new active power set point per each asset and time step to solve the detected congestion problems, and thus represents the communication interface between grid congestion management (UC2) and coordinated voltage/reactive power UC1 control functions.

4.3.2 Coordinated voltage/reactive power control (UC1) control function data

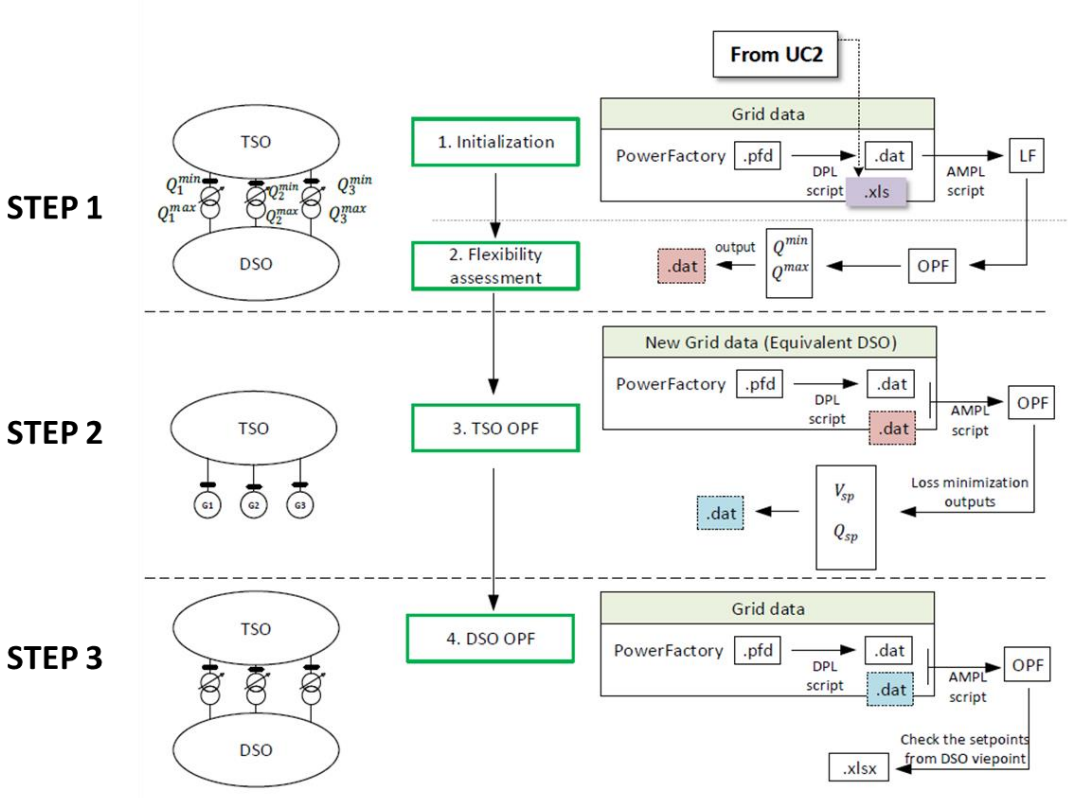

Figure 9: UC1 internal and external exchanged information

UC1 includes three steps: Flexibility assessment, TSO optimal power flow (OPF), and DSO OPF. During these steps, all the required output data from one step to another is internally transmitted to the relevant elements in AMPL. This means that the three steps run subsequently and results of each will be transferred to the next step. AMPL is an algebraic modelling language to describe and solve high-complexity problems for large-scale mathematical computing (i.e., large-scale optimization and scheduling-type problems). AMPL supports dozens of solvers, both open source and commercial software. The optimal power flow problem is coded by different elements including sets, parameters, variables, objective functions and constraints. Parameters are input data and variables are the outputs. After definition of the problem one of the solvers is chosen to solve it. For INTERPLAN, the interior point optimizer (IPOPT) solver is used.
As shown in Figure 9, during the UC1 simulation much of the internally exchanged information is created in .dat format. The .dat file is generated automatically by a DPL (DIgSILENT Programming Language) script in PowerFactory as shown in Figure 9, step 1, grid data frame (.dat file in white

colour). All input grid data (shown in the following tables) required for AMPL are included in this white-coloured .dat file.
The grid data in the .dat file are divided into three categories: Bus data, branch data, and generator data. For each one, there are general data (e.g. the number of buses, branch and generators) and general information such as the number of buses and detail data for each bus at each time step as follows:

Bus data

Bus type	Active power load	Reactive power load	Bus area	Initial Bus voltage magnitude	Initial bus voltage angle	Maximum and minimum us voltage	Nominal bus voltage	Bus GPS longitude	Bus GPS latitude

Branch data

Branch from which bus	Branch To which bus	Branch resistance	Branch inductive reactance	Branch capacitive reactance	Branch capacity	Branch type	Branch rate of power if it includes transformer	Branch tap changer number if it includes transformer

Generator data

Generator bus number	Generator type	Generator operation mode	Initial active power	Initial reactive power	Reactive power boundary	Active power boundary	Initial voltage magnitude and angle

4.4 Showcase 3: TSO-DSO power flow optimization

Showcase 3 combines frequency tertiary control based on optimal power flow (UC 3, Figure 10) with power balancing at DSO level (UC5, Figure 11) for ensuring power balance within the transmission and distribution network, participation of non-synchronous energy resources in the tertiary reserve market, and supporting the TSO in keeping the whole network stable. The two use cases are implemented by separate Python scripts and are executed subsequently for all time steps. UC3 controls the resources at both TSO and DSO levels while UC5 controls only DSO resources, hence the data that have to be exchanged between those two use cases are information on the TSO/DSO interconnection power flow (P,Q) for each time step and the active power set points for controllable assets in DSO level. For this exchange, a CSV formatted file is used.

The overall showcase needs a grid model that can be imported to PowerFactory, as well as time series data with 15 min interval as inputs. The latter can be either already in the model or in CSV format. In case the CSV format is used, the data is imported into the PowerFactory model by a Python script.

4.4.1 Frequency tertiary control (UC3) control function data

UC3 is about provision of tertiary reserves of DER, storages and controllable loads at TSO and DSO levels. As can be seen in Figure 10, the software used for simulation purposes is PowerFactory and

AMPL. Hence, the PFD data format is used for grid models with embedded time series and the .dat format is used for information transmission from PowerFactory to AMPL. AMPL and Python scripts are utilized for algorithm implementation and processing of the results.

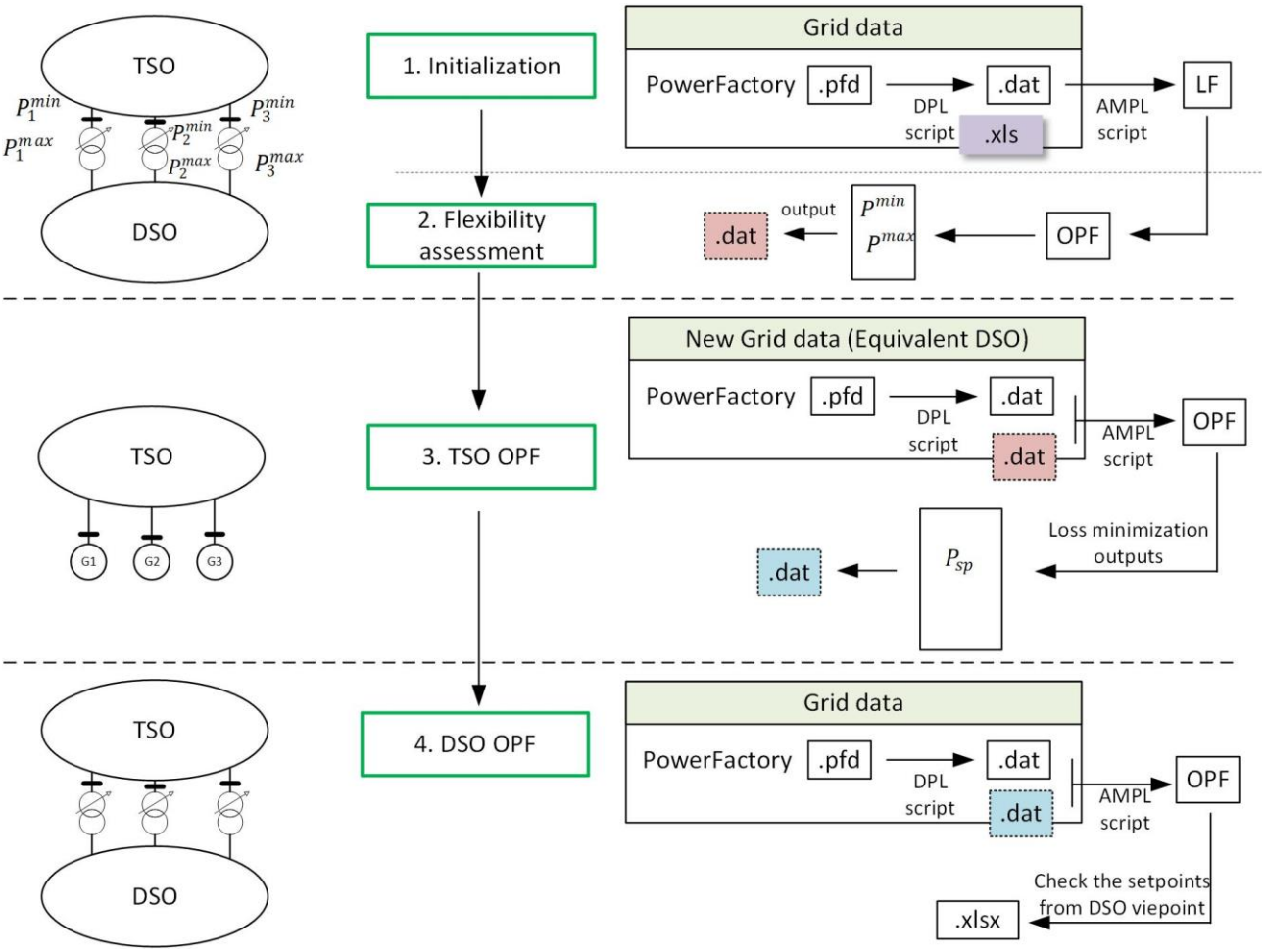

Figure 10: UC3 internal and external exchanged information

The output of the UC3 control function is an Excel file which contains active power setpoints for the TSO and DSO controllable grid assets.

4.4.2 Power balancing at DSO level (UC5) control function data

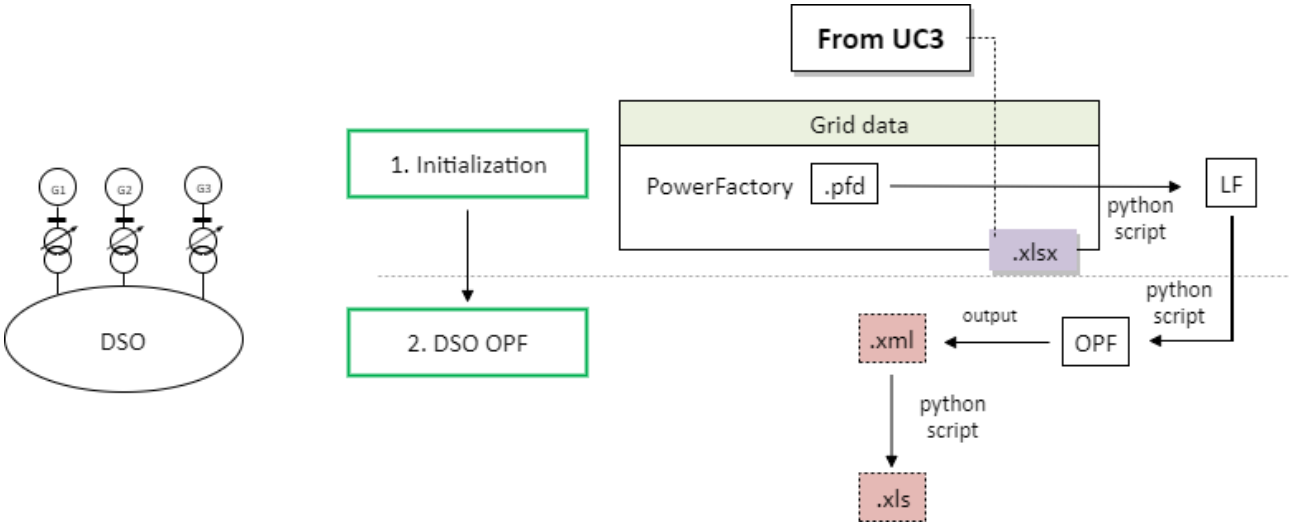

Figure 11: UC5 internal and external exchanged information

The UC5 internal control function steps and data exchange are depicted in Figure 11. The control function uses the output Excel file created by UC3 as an input. In addition, it uses internal grid data on active power of generators and storage, as well as storage state of charge. This information is extracted directly from the PowerFactory grid model using Python.

The output of UC3 execution is a set of Excel files which contain information on transformer loading and losses, line losses, generators' active power, and terminal voltages for each time step. The individual grid assets are identified by their PowerFactory model names.

4.5 Showcase 4: TSO-DSO coordinated active and reactive power flow optimization at all voltage levels

Showcase 4 consists of frequency tertiary control based on optimal power flow (UC3) and coordinated grid voltage / reactive power control (UC1) which are executed subsequently. As input data for both, a PowerFactory grid model is needed. From this, a .dat file is automatically created by an DPL script in PowerFactory which includes all information of different elements of the network: busbars, lines, generation and load attributes. The .dat file contains all data needed an input for AMPL / Python scripts which implement the UC control functions.

The grid data in the .dat file is divided into the same three categories as described in chapter 4.3.2. UC1 and UC3 have both three similar steps of optimisation of active power (UC3) and reactive power (UC1) including flexibility assessment, an OPF at TSO level, and finally an OPF at DSO level. During subsequent execution of these steps, all the required data transfer from one step to the other is handled internally in AMPL/ Python software packages.

The decision variables of UC1 and UC3 are the results of objective functions for different electric devices in the network. They might have some limitations that should be met by constraints (e.g. operational constraints of a DG).

4.5.1 Frequency tertiary control (UC3) control function data

The detailed exchanged information for UC3 as used in showcase 4 is depicted in Figure 12. The UC3 control function results in an Excel file, which is used by UC1 as an input. The exchanged data consists of the attributes of the grid model as well as active power setpoints for resources and loads.

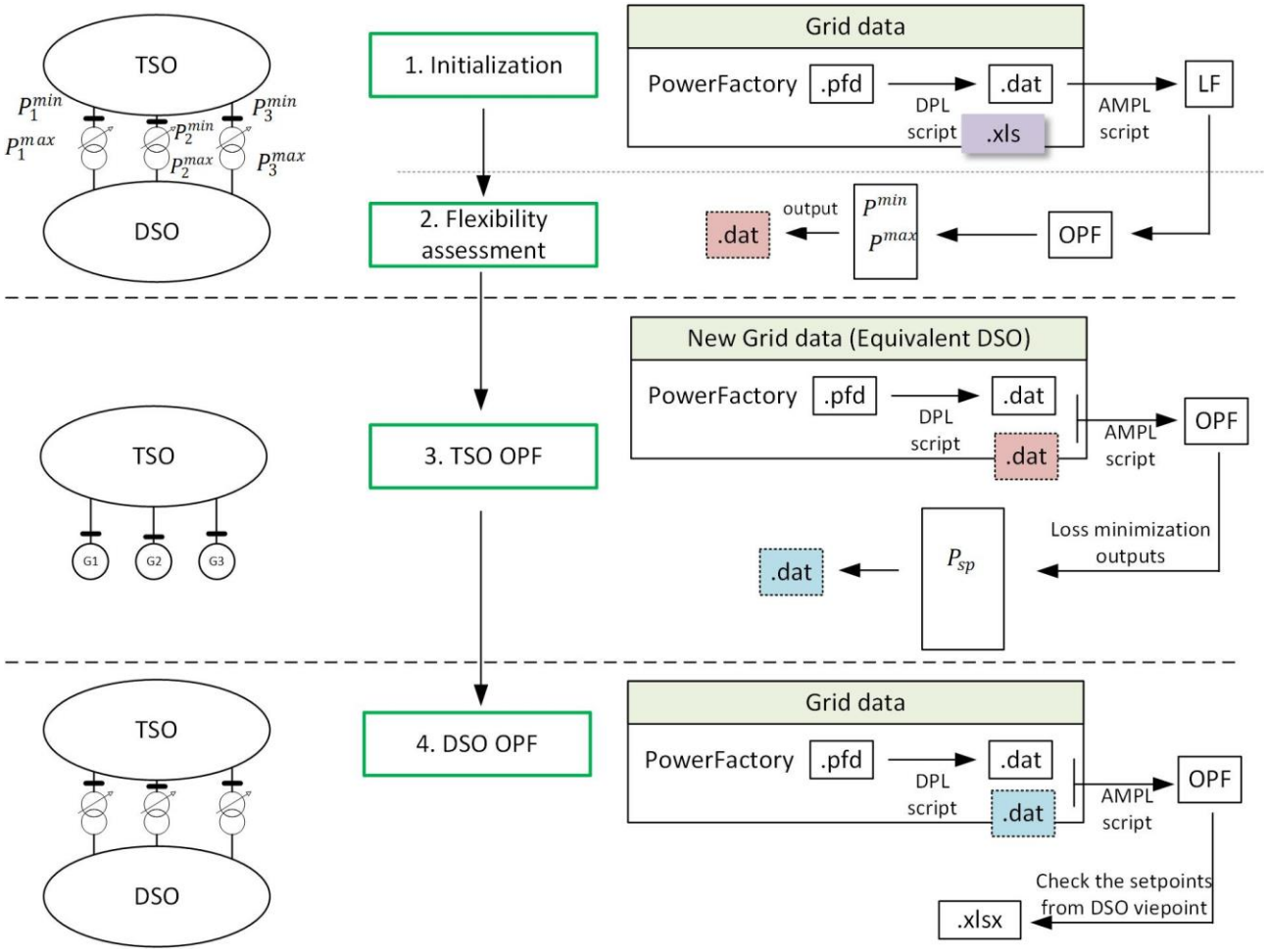

Figure 12: UC3 internal and external exchanged information as used for showcase 4

4.5.1 Coordinated voltage/reactive power control (UC1) control function data

UC1 is here used in the same way as for showcase 2 in section 4.3.2 (Figure 9).
The final output of showcase 4 is stored in Excel format, including the required data for calculation of KPIs.

4.6 Showcase 5: Energy interruption management

Showcase 5 involves energy interruption management (UC7) only. However in this case, the showcase is the combination of sub use cases “optimal generation scheduling and sizing of DER for energy interruption management” (7.1) and “optimal energy interruption management” (7.4). From an external viewpoint sub use case 7.1 can be interpreted as contingency list generator, while 7.4

performs the optimal energy interruption planning against the contingency events. The control functions are implemented through Python code which is connected to the PowerFactory model using the PowerFactory Python API (Advanced Programming Interface).

The PowerFactory network and associated data acts as input to sub use case 7.1. Based on the availability of data, daily or yearly quasi-dynamic simulations are performed. While monitoring the simulation results, a list of credible contingencies is prepared. Each contingency event is activated and the resulting grid state represents the base showcase. As both the sub use cases belong to the same use case, there is seamless transformation of information within the Python source code which is used to implement the showcase.

The information generated by UC7.1 is used for the constraint modelling in UC 7.4. Before the optimization process of sub use-case 7.4, a contingency from the list is activated, and study case date and time are changed to the point in time when the system is most vulnerable to that contingency. Therefore it is assumed that profiles of loads and generators are available for that point in time. It is assumed that after the contingency activation process, the load flow can converge. Afterwards, critical nodes in the post-contingency state of the grid are identified. A sensitivity analysis is performed to prioritize resources (load shedding flexibility and generator re-dispatch) and to tackle critical nodes in the post contingency scenario.

The prioritization is performed by adjusting the costs associated with load shedding and generator re-dispatch. Afterwards, sub use case 7.4 uses “minimization of load shedding” from the optimal power flow module of PowerFactory.

The input to the showcase is the PowerFactory network and associated profiles. The output of UC7.1 is a CSV file containing a list of contingencies with their activation time. This CSV file acts as input to UC7.4. The output of the showcase is a CSV file containing reliability KPI values for each contingency with and without using the interruption management approach proposed by the showcase.

5. Co-simulation data models and interfaces

5.1 Co-simulation toolset for INTERPLAN

Based on the existing co-simulation platform OpSim developed by Fraunhofer IEE and University of Kassel in the German research projects OpSim and OpSimEval, INTERPLAN designed a toolset intended for validation of control functions and tools developed within the project. The toolset consists of several pieces of software, which were mostly implemented in Python. Figure 13 shows the toolset components; the arrows represent abstract information flows between the parts. Black arrows relate to automatic data transmission, grey arrows relate to manual data inputs using files. The yellow parts are components of the OpSim platform. The green part is provided by the grid calculation software pandapower. The blue parts have been specifically developed for the requirements of INTERPLAN.

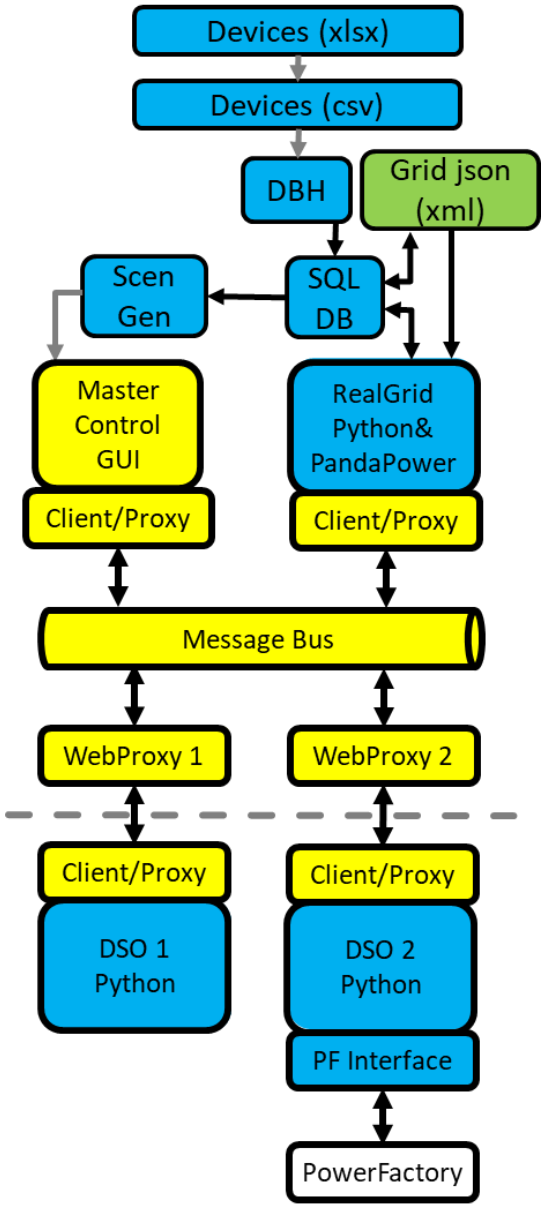

Figure 13: Co-simulation toolset for INTERPLAN

The following sections will focus on explanation of the toolset with special consideration of interfaces and data models.

5.2 OpSim data model, interfaces and scenario definition

OpSim can be used for simulating operation and control strategies and aggregators in smart grids with very high share of renewable generation [7]. Co-simulations can be executed both in real time and accelerated, multiple interacting control strategies can be analyzed, and there is a webservice component enabling remote co-simulation subcomponents accessing the centrally hosted OpSim core service. The latter approach is called “OpSim as a service”.

The components, interfaces and data model of OpSim are described in detail in the OpSim final report [8], and are summarized as follows.

5.2.1 OpSim interfaces and components

An OpSim as a service co-simulation consists of the following components:

- A core master control program (MCP) which is running at a central server and acts as central co-simulation coordinator. The MCP is controlled using a Graphic User Interface (GUI) running within the same network,
- A RabbitMQ MQTT (Message Queuing Telemetry Transport) message bus which acts as central message transmission and coordination system,
- Clients which are subscribing to the message bus and are receiving and sending simulation messages,
- Proxies which translate the message contents between clients and co-simulation subcomponents,
- Web proxies which provide remote access to the message bus for clients running outside the central MCP server network,
- Co-simulation subcomponents which are participating in the co-simulation as active elements, and are connected to the co-simulation by one Client/Proxy module each.

A minimal OpSim co-simulation would consist of the MCP with GUI, MQTT message bus, and exactly two Clients, Proxies and subcomponents, all running in the local MCP server network. In this setup, no web proxies would be needed.

Essential functionality is provided within the OpSim API in form of Java interfaces. The “*ClientInterface*” includes functions for sending messages of the type *OpSimMessage* (cp. chapter 5.2.2), as well as functions for starting, pausing, stopping and resetting of subcomponents. Usage of this interface is encapsulated by Java and Python libraries used for implementing subcomponents. The “*MasterControlProgramCoreBase*”-Interface provides control functions for starting, pausing, and stopping of the co-simulation. Also, co-simulation status and subcomponent status can be queried using this interface. This interface is mainly used by the MCP graphical user interface. The “*MCPStateListenerBase*”-Interface allows subscribing to event-based updates about co-simulation or client status.

In addition to that, there are several Proxy-Interfaces. First, there is a "*MemoryHandler*"-Interface which provides functions for processing of logging messages. It is implemented by clients. The "*Matlab*"-Interface is used for interaction with the MATLAB software. The "*ProxyClientInterface*" is used for interaction between Client and Proxy. It especially provides an init and setup function, which can be executed by the client in order to initialize and setup connection to a server. It also provides a callback method which is called by the client and returns a list of received messages. By using another interface function, a concrete proxy implementation can be loaded into a client. This is essentially done by each subcomponent; a simple subcomponent implementation would consist of a proxy implementation and simple control mechanism within a single Java class or Python code file, inheriting from a derivative of the *ProxyClientInterface* class.

Finally, the "*ProxyComponentInterface*" provides init, step and stop functions for each subsimulation. These are core functions that need to be implemented by subsimulation developers. The init function in this case ("*initComponent*") is called by the client with a serialized version of the simulation setup as parameter, which is used for setting up the subsimulation. The step function is a callback for implementing a single simulation step. It is called by the client with a list of received messages and a timestamp which indicates the time the subsimulation may advance to.

The aforementioned interfaces are mostly part of the OpSim infrastructure and, except from the *ProxyComponentInterface* and *ProxyClientInterface*, are hidden from subsimulation developers. Furthermore, there is no need to adapt named interfaces for INTERPLAN; they are used as-is. The implementation of the *ProxyComponentInterface's* init, step and stop functions is specific to the controller type represented by the subsimulation in question.

5.2.2 OpSim data model

All messages transmitted over the OpSim MQTT bus follow a specific pattern fixed by the OpSim data model. This data model is quite flexible; it comprises many standard parameters of an electrical energy system, but can be extended by new parameters easily. Such extension can either be done by adding new data model classes during simulation runtime through a lookup mechanism, or use a generic message as a container for arbitrary data formats agreed between two or more subsimulations. Typical co-simulations only use a fraction of the data model elements, as shortly introduced in the following subchapters.

5.2.3 OpSimMessage

This is a generic base class which all message type classes inherit from. It contains two informations essential for co-simulation operation: *assetId* and *delta*. The *assetId* is a unique identifier for an object in the simulated electric power grid. An object can for example be a line, a transformer, a generator or a load. The *delta* is a long integer value which represents the time (linux epoch time) until which the message is valid. This time is usually in the future and is used for conservative synchronization between subsimulations; in case a message is outdated, it is ignored by clients and the client connected subsimulation. A *delta* value of -1 indicates that a message never expires until it is overwritten by a new one for the same asset.

5.2.4 OpSimScheduleMessage

This message type contains a list of *OpSimScheduleElements*, and represents an operation

schedule, e.g. for active power setpoints for the future, representing a power plant operation plan.

5.2.5 OpSimScheduleElements

This data type represents a setpoint that is decorated with a timestamp given in ISO 8601 format. It is defined by said timestamp, a scheduled value and an according value type, which is one of the *SetPointValueTypes*.

5.2.6 OpSimAggregatedSetPoints

This message type contains a list of setpoints for a specific asset, each setpoint consisting of a value and a value type. The setpoints are to be executed by the asset model as soon as the message is received. The value type is one of the *SetPointValueTypes*. If the asset is e.g. a wind power plant, an *OpSimAggregatedSetPoint* message could contain an active and reactive power setpoint.

5.2.7 OpSimAggregatedMeasurements

This message type contains a list of measurements taken from a specific asset, e.g. voltage amplitude and phase at a busbar. Each measurement consists of a timestamp, a value and a value type. The value type is one of the *MeasurementValueTypes*.

5.2.8 OpSimFlexibilityForecastMessage

This message contains a list of *forecastMessages* for a specific asset. Each such *forecastMessage* contains a future timestamp, a value type and a numerical minimum and maximum value. The value type is one of the *MeasurementValueTypes*. The minimum and maximum values represent a forecasted value interval.

5.2.9 OpSimGenericMessage

This message type can be used for subsimulation components which communicate, but don't use the OpSim data model. However, developers of said components must ensure that they can mutually interpret the message contents correctly, as they are not checked, translated or modified by OpSim clients, proxies, message bus or the MCP core.

The INTERPLAN co-simulation mainly relies on the OpSim standard data model, which foresees value types as listed in Table 3 and Table 4. There was no need for usage of generic messages identified as of yet; in case a controller would need to transmit or receive information that cannot be modeled by standard types – e.g. for transmission a three-phased voltage measurement - using the generic message type would be the preferred way though.

Table 3: OpSim standard MeasurementValueTypes (selected)

ACTIVE_POWER	Total or phase 1 active power
ACTIVE_POWER_2	Phase 2 active power
ACTIVE_POWER_3	Phase 3 active power
CURRENT	Total or phase 1 current absolute
CURRENT_2	Phase 2 current absolute

CURRENT_3	Phase 2 current absolute
CURRENT_ANGLE	Total or phase 1 current phase angle
CURRENT_ANGLE _2	Total or phase 2 current phase angle
CURRENT_ANGLE _3	Total or phase 3 current phase angle
FAULT_CODE	
REACTIVE_POWER	Total or phase 1 reactive power
REACTIVE_POWER_2	Phase 2 reactive power
REACTIVE_POWER_3	Phase 3 reactive power
SOC_VALUE	Battery state of charge
SWITCH_POSITION	Power switch position
TAP_POSITION	Tap changer position
VOLTAGE	Voltage absolute
VOLTAGE_ANGLE	Voltage phase angle

Table 4: OpSim standard SetPointValueTypes

ACTIVE_POWER	Total active power
REACTIVE_POWER	Total reactive power
SOC_VALUE	Battery state of charge
SWITCH_POSITION	Power switch position
TAP_POSITION	Tap changer position
VOLTAGE	Voltage absolute
VOLTAGE_ANGLE	Voltage phase angle

5.3 Automated scenario definition for INTERPLAN

5.3.1 General OpSim scenario definition

All subsimulations of an OpSim co-simulation, their respective assets, the messages exchanged between them, as well as metadata about the co-simulation itself, e.g. the start and end time, are defined within a single XML file, the so called scenario file. For each subsimulation and its client, the scenario file defines the permitted set of assets about which data may be exchanged over the MQTT message bus. The scenario file follows a specific format defined by OpSim and is manually uploaded into the OpSim MCP component using the GUI. The MCP sends it to according clients who can query it by a service discovery. Using this information the clients organize their subscriptions to the message bus.

The scenario file specifically contains information about the following:

- The grid model and assets used,
- the components that are part of the co-simulation,
- the data the components shall communicate over the message bus,
- the frequency of activity of each component,
- the start and end time of the simulation,
- the type of time series and prognosis data the components need, and
- the server directory where time series and prognosis data are found.

The following XML code shows the general data format layout using an example simulation setup with only two subsimulations. Metadata about the scenario creation and the simulation is defined in lines 2-8. Lines 9-12 point to the grid data file which is to be used. Starting from line 13, the first subsimulation is defined, which represents an asset operator named “RealGrid”. The asset operator type indicates that it is a physical grid simulation, which is typical to occur in an OpSim co-simulation. The file only defines one asset, a generator “G1”, which is both controllable and readable asset for the RealGrid subcomponent. This is also the case for the second subsimulation called “DSO1”. This subsimulation is connected to OpSim over a WebProxy, using the IP port 34119 as specified in line 44. The address of the WebProxy server is not specified in the scenario file, but in the subsimulation code. Also, as defined in lines 21 and 36, both subsimulations are active each 60,000 ms, hence every minute of simulation time.

```
1       <ns3:ScenarioConfig [...]>
2       <latestModificationTime>2018-11-20T15:06:00.489+01:00</latestModificationTime>
3       <realTimeSimulation>false</realTimeSimulation>
4       <scenarioId>1a9d0477-59a0-45e7-b79f-c3a29105d074</scenarioId>
5       <scenarioName>RefDemoWeb</scenarioName>
6       <simulationIntervalEndTime>2018-11-20T01:00:00+01:00</simulationIntervalEndTime>
7       <simulationIntervalStartTime>2018-11-20T00:00:00+01:00</simulationIntervalStartTime>
8       <weatherYear>2018</weatherYear>
9       <GridInformation>
10              <gridName>RealGrid</gridName>
11              <gridPath>C://User//INTERPLAN//Scenario//RefDemoWeb</gridPath>
12      </GridInformation>
13      <AssetOperator>
14              <ns2:assetOperatorName>RealGrid</ns2:assetOperatorName>
15              <assetOperatorType>SIM</assetOperatorType>
16              <controlledAssets>
17                      <assetType>GENERATOR</assetType>
18                      <gridAssetId>G1</gridAssetId>
19                      <measurableQuantities>ACTIVE_POWER</measurableQuantities>
20              </controlledAssets>
21              <operationInterval>60000</operationInterval>
22              <readableAssets>
23                      <assetType>GENERATOR</assetType>
24                      <gridAssetId>G1</gridAssetId>
25                      <measurableQuantities>ACTIVE_POWER</measurableQuantities>
26              </readableAssets>
27      </AssetOperator>
28      <AssetOperator>
29              <ns2:assetOperatorName>DSO1</ns2:assetOperatorName>
30              <assetOperatorType>DSO</assetOperatorType>
31              <controlledAssets>
32                      <assetType>GENERATOR</assetType>
33                      <gridAssetId>G1</gridAssetId>
34                      <measurableQuantities>ACTIVE_POWER</measurableQuantities>
35                      </controlledAssets>
36              <operationInterval>60000</operationInterval>
37              <readableAssets>
38                      <assetType>GENERATOR</assetType>
39                      <gridAssetId>G1</gridAssetId>
40                      <measurableQuantities>ACTIVE_POWER</measurableQuantities>
41              </readableAssets>
42              <parameters>
43                      <key>port</key>
44                      <value>34119</value>
45              </parameters>
46      </AssetOperator>
47      </ns3:ScenarioConfig>
```

In order to define such scenario file, OpSim provides a scenario generator which takes one CSV file per asset operator as input. The files needed for generation of the example above are shown below. They contain only one line for each readable or controllable measurable quantity. The simulation

metadata can be selected using a dedicated scenario generator. The scenario file is then generated with a simple mouseclick.

```
R;G1;GENERATOR;ACTIVE_POWER
W;G1;GENERATOR;ACTIVE_POWER
```

```
W;G1;GENERATOR;ACTIVE_POWER
R;G1;GENERATOR;ACTIVE_POWER
P;port;34119
```

5.3.2 Device definition for co-simulation

For the INTERPLAN co-simulation, several hundreds of grid assets were expected, and the number of controllable or readable measurable quantities may be even higher. Hence, compiling the input files manually as explained above would be unfeasible. Consequently, an alternative solution for generating the scenario file was developed taking a look at the requirements of the control function validation. In particular, the following requirements were identified and respected:

- General asset types would be generators, loads, storages, and transformers with tap changers,
- Generators could be of different types, e.g. wind, photovoltaic, fossil etc.
- Each asset could be controllable or non-controllable,
- For each device controlled, there could be different types of control (e.g. active power, reactive power, both, etc.),
- For each controllable device, an attribution to an asset operator would be needed,
- A controlled device is controlled by at least one asset operator, but might be controlled by more than one; e.g. one asset operator could control a generator's active power, while another asset operator could control the same generator's reactive power,
- The definition file should allow definition of core asset parameters, e.g. nominal apparent power or storage capacity,
- The definition file should allow to specify a generic device subtype for implementing specific device models, e.g. a PV inverter with a specific P/Q operation curve,
- Physical devices are typically equipped with a local controller which receives remote control signals. It should be possible to specify a generic control subtype for specific local control schemes,
- The definition of readable and writable assets directly depends on the definition of controllable assets and the type of their control. Furthermore, the amount of subsimulations would be given by the set of operators defined. E.g. for a co-simulation including a generator which is controlled in terms of active power by operator 1 and reactive power by operator 2, there would be three subsimulations: one for the real physical grid, and two for the two operators. For the physical grid, both generator active and reactive powers would be writable and readable by and from the message bus, such that both operators could access. However, for the operator 1 subsimulation, both power values might be readable, but only the active power would be writable to the message bus, because this operator is not allowed to control reactive power.

The developed solution works based on a single Excel file which follows a specific template format. It contains the following tables:

- An overview table, containing an explanation about how to use the file,
- One table each for generators, loads, storages and transformers, containing a list of according assets with parameters.

The asset definition tables contain one line for each asset controlled by a specific operator. If one asset is controlled by two operators, there are two lines. The columns in the tables give parameters depending on the individual table. The following columns are present in every table for generators, loads and storages:

- The asset name. This name must be unique unless an asset is controlled by different operators. Also the name must match the name of the asset in the grid model used since it is the unique identifier for all the co-simulation. The physical grid subsimulation contains a consistency check for ensuring this. Assets present in the definition file, but not present in the grid model will be ignored. Assets present in the grid model, but not in the definition file will be uncontrolled and be operated statically according to their initial active and reactive power values.
- The bus id of the grid node the asset is connected to. This is only given for information as this is defined in the grid model too.
- The control type (*ctype*). This must be one of the literals given in Table 6. The allowed literals are different depending on the asset type as given in the table.
- The control subtype (*csubtype*). This is an integer number which indicates that a specific local control behavior shall be used. The control behavior must be hard coded in the physical grid simulation. Currently the control subtype is unused.
- The nominal apparent power in MVA *sn_mva*. This is a floating point number.
- The asset operator (*operator*). This is an integer number. For uncontrolled assets, the number is ignored.

The following columns are present in only a part of the tables:

- In the generators table, there is a column for the generator type and subtype. The type must be one of the literals given in Table 5. The subtype is an integer number and can be used for indicating that a specific device model shall be used. This device model needs to be hard coded in the physical grid simulation subcomponent. Currently the subtype is unused.
- The generators table contains columns for maximal and minimal active powers (*pmax_mw, pmin_mw*), as well as a *cosphi* column. The latter is for information only.
- The loads table contains columns for maximal active power (*p_mw*) and *cosphi*. The latter is for information only.
- The storages table contains a maximal charge and discharge power (*pcmax_mw*, *pdmax_mw*), a capacity in MWh (capacity), as well as charge, discharge, and self-discharge times (*charge_time, discharge_time, self_discharge_time*). The charge and discharge time give the time in minutes which is needed to charge/discharge the storage across its full capacity using the maximal charge and discharge powers, under nominal conditions. The self discharge time gives the time in minutes until the fully loaded storage is completely discharged if a charge power of 0 (zero) is applied.
- The transformers table includes a maximal and minimal as well as a neutral tap setting (*tap_max, tap_min, tap_neutral*). The effect of the tap setting depends on the according transformer model which is stored in the physical grid model, but not mentioned in the device

definition file. The tap parameters must be integer numbers. Usually, the maximal tap setting is positive, the minimal tap setting negative. The neutral setting must be within the min/max tap interval. A transformer may only have one operator, as the tap setting is the only independent control variable.

Table 5: Current co-simulation generator types

Generator type	Literal
Photovoltaic	pv
Wind	wpp
Other (synchronous machine, hydro power, bio power, or fossil)	sm

Table 6: Control types

ctype	Asset types	Remote controllability
U	All	Uncontrolled, active power fixed by time series, initial cos phi is held constant
P	All except trafos	Active power control, don‘t change reactive power
Q	Generators	Reactive power control, don‘t change active power
PQ	Generators	Reach active and reactive power setpoints as close as possible, maintain cos phi
PU	PV, WPP, STO	Apply local P(U) static. Not yet implemented.
QU	PV, WPP, STO	Apply local Q(U) static. Not yet implemented.
B	Loads	Binary switchable; off: active and reactive powers zero; on: active and reactive powers nominal
T	Trafos	On-load tap changer

For controllable assets that are controlled by more than one operator, only the first line must contain the whole parameter set. The following lines need only contain the name, the control type and control subtype for the according operator, and the operator number. Having two operators controlling the same asset with the same control type will result in undefined behavior of the simulation. Hence, for a specific controllable asset, every operator controlling it must use a different control type.

5.3.3 Grid model and Time Series

Next to the device definition file, two very central parts of the INTERPLAN co-simulation are the physical grid model and the time series data. The latter data represents the behavior of all generators and loads when no further control is applied, and is generated according to the underlying scenario adapted to the grid model.

The grid data is primarily used by the component “RealGrid” as shown in Figure 13 in order to simulate a full physical network, without using grid equivalents. This simulation is performed using pandapower as a grid calculation software; thus, the basic format for the grid model is the XML/JSON file format as defined by pandapower 2.0 [9]. This format is directly used by the RealGrid component.

However, the network data is also needed by the operator subsimulations (in Figure 13 shown as “DSO1” and “DSO2”). These subsimulations essentially implement the tools and controllers as developed in WP4 and WP5, thus need the grid model as basic input data. Since the WP5 developments rely on PowerFactory as grid calculation software, the pandapower format is converted into PowerFactory using an automatic converter, which was available at IEE from previous projects, and was further developed in INTERPLAN (cp. chapter 3.1). The converter also constructs a graphical model of the network, taking into account busbar geolocations as stored in the original pandapower grid model.

The time series data is generated by IEn and calculated against the converted grid model in order to obtain a valid mode of operation for one day with time resolution of 15 minutes. The result is provided in form of an Excel file which contains one line per each generator or load, a column for the grid asset name, and 96 columns for the time series values. The grid asset names are the same unique names as used in the devices definition file and the grid model.

5.3.4 Scenario file generation

As soon as a simulation setup has been fixed, meaning that a device definition Excel file has been written according to the template, a time series Excel file and an according grid model in JSON format were prepared, the rest of the steps to obtain a scenario file for OpSim and prepare the co-simulation are mostly automated. The process is schematically depicted in Figure 13, where the device definition file can be found as “Devices (xlsx)” at the top. This process is meant to be executed at a simulation engineer’s computer which has direct access to the OpSim MCP core, and is also used for setting up the OpSim simulation using the Master Control GUI, as well as the RealGrid physical grid simulation. At the same computer, a postgres SQL database needs to be installed as a prerequisite. This database is used for storing all co-simulation scenarios, setups and results, and is depicted in Figure 13 as “SQL DB”.

In a first step, the generators, loads, storages and trafos tables from the Excel file have to be exported as CSV files, as indicated in Figure 13 as “Devices (csv)”. Those files should be stored in a scenario-specific folder on the developer’s hard drive and named “generators.csv”, “loads.csv”, “storages.csv”, and “trafos.csv”. Also the time series Excel file is exported as CSV file, named “timeseries.csv” and put into the same folder. It is recommended to copy the grid model file into this folder as well.

The next step uses a piece of software written in Python that is called “DataBase Handler”. This software is depicted by the module “DBH” in Figure 13. The database handler’s purpose in this case is to read all information in the CSV files into a database schema which represents the simulation scenario. This can be done by calling a single function which takes the path where the CSV files are stored as a parameter. Upon creation of a new simulation scenario, such schema can be created manually by copying a template schema, which is also contained in the database. The database handler additionally contains a function to write base data that is not contained in the CSV files into the database. This base data comprises:

- The name of the scenario,
- the file name and path of the grid model file to be used (“Grid json (xml)”),
- the start and end simulation time,
- the simulation timestep width,

- the names of the asset operators, and
- the webproxy ports for the asset operators.

Note that this step does not involve a consistency check of information contained in the CSV files vs. information contained in the grid model file. In particular, it does not check if the asset names in the grid model file and the CSV files are unique and match. Such consistency check is performed on start of the co-simulation itself. Also, it is mandatory that the time series definition spans the whole simulation time interval. The time resolution of the time series data is fixed by the simulation timestep width. In case the time series data includes more data than needed, only the first part of the data is used by the simulation.

After execution of this step, all data which is needed for generation of an OpSim scenario file (thus, needed for setting up the OpSim MCP core component) is in the database. This generation is thus carried out fully automatically by a Java component called “ScenGen” (cp. Figure 12) which directly accesses the database and writes the scenario file. Latter file can be directly loaded into the Master Control GUI for preparing the co-simulation. Note that the time series data and grid model are not part of the scenario file, but are later used by the RealGrid component, which also accesses the SQL database.

5.3.5 Database structure and result storage

By the previous explanation it is obvious that the SQL database is the central data hub of an INTERPLAN co-simulation. It contains an SQL schema for each simulation scenario. Within each such schema, there are the following tables:

- A *base* table containing the base data as mentioned in subsection 5.3.4,
- a *devices* table containing the device definitions,
- a *timeseries* table containing the time series of the active powers in MW,
- a *results_assets_p* table containing the active power of all grid assets for all simulation timesteps in MW,
- a *results_assets_q* table containing the reactive power of all grid assets for all simulation timesteps in MVar,
- a *results_buses_u* table containing the voltage absolutes of all buses for all simulation timesteps in V. Currently, the real grid simulation performs only symmetrical load flow, so there is only one voltage absolute value,
- a *results_tl_lod* table containing the loadings of all lines and transformers in percent,
- a *results_tl_plos* table containing the active power losses of all lines and transformers in MW, and
- a *results_tl_qlos* table containing the reactive power losses of all lines and transformers in MVar.

The table for the grid assets and buses is indexed with the asset and bus names taken from the grid model. The transformer and line tables are indexed with according names, but additionally contain the names of the buses the transformer or line connects with.

The table structure can mostly be automatically generated by the database handler component or cloned from an existing database schema. Since the database entries are easily accessible from a

Python script, the results can be used and combined with e.g. the device definition data for automated calculation of KPIs. For example, one could filter the PV generators from the results table using the information contained in the devices definition table, and accordingly add the generated active powers in order to obtain the generated energy from PV for the whole simulation time interval.

The following screenshot shows the database viewed with Eclipse DBeaver. The view opened shows the contents of the *results_tl_lod* table after a test simulation run. The columns “t0”, “t1” … represent the simulation timesteps.

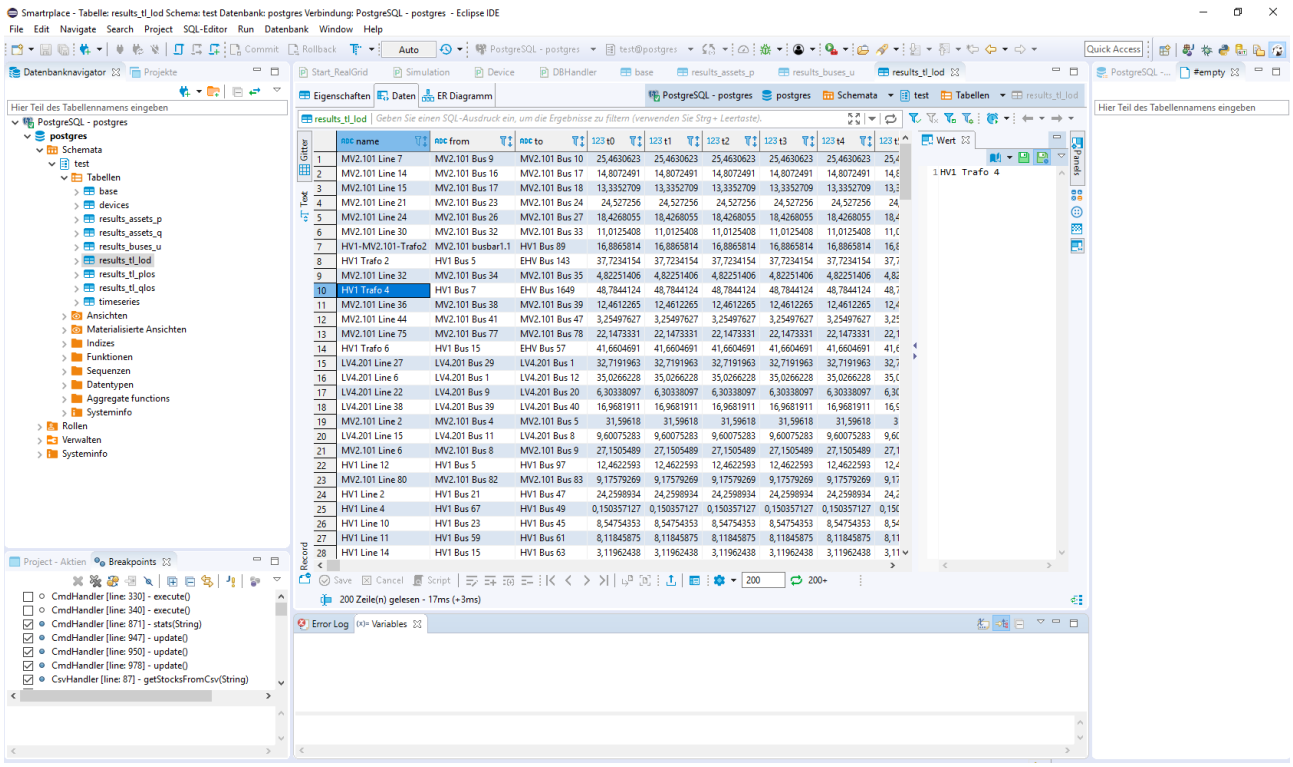

Figure 14: SQL database view in Eclipse DBeaver

The advantage of concentrating the data at the database is that all data which relates to a co-simulation run is bundled at one place. A drawback of this implementation is that write operations into the database are currently rather slow. For the WP5 test network, writing all results from a single time step would take about 6 seconds. However, since each time step currently represents 15 minutes of real time, this still fulfils the timing requirements, hence the performance is currently considered sufficient.

5.3.6 PowerFactory interface

The PowerFactory interface [7] is primarily intended for connecting a remote operator subsimulation which represents an OpSim Proxy with the PowerFactory grid calculation software and environment. It is able to utilize PowerFactory for carrying out e.g. load flow calculations at the remote subsimulation during co-simulation runtime. The PowerFactory interface is thus depicted at the bottom of Figure 12, in this case connecting the DSO2 subsimulation with a PowerFactory instance. The following section contains a technical description of this interface.

The interface is implemented as Python script and can either be used directly by modifying the part that is implementing the example DSO2 control, or as a template for connecting an already existing simulation using Python and PowerFactory to OpSim. The main part of the interface is encapsulated in a class called *PowerFactory_interface*. This class basically bundles all PowerFactory related functions, assuming that the whole script is to be executed from within the PowerFactory environment with a grid model project opened in the background.

The class contains a list of PowerFactory assets, where each such asset is an instance of a sub-class called *PowerFactory_asset*. It can represent a line/cable, load, node, generator or transformer. It contains a name, a PowerFactory handle, a type, a controllability flag, a three-phased flag, as well as data fields for the asset's state and calculation results, e.g. the active power, losses, or switch state. It also contains a list of controllable properties, which is used to store identifiers for OpSim controllable quantities according to Table 3. The *PowerFactory_asset* provides two main functions for interacting with these assets: an update function which reads states and calculation results from all known assets from the PowerFactory application, and a push function which writes the setpoint values of all such assets to PowerFactory.

Accessing these data structures, the PowerFactory interface class provides the following functions:

- A constructor which fetches a handle to the PowerFactory Application and opens a project, resets the PowerFactory calculation and prints a startup message at the PowerFactory terminal,
- A function *do_ldf* which writes all internal PowerFactory assets to the PowerFactory application, triggers a load flow and updates all internal assets by reading back the results
- A function *addAsset* which searches the PowerFactory project for a given asset. The asset is identified by its name and type, where the type can be line/cable, node, load, generator, switch, or transformer. If the asset is found, it is added together with its type and PowerFactory handle to the internal asset list. The function also takes the controllability flag as an input, and sets it accordingly in the newly added asset,
- Several functions for getting assets, asset PowerFactory handles, checking for asset existence, and setting asset values. Each asset is generally identified by its PowerFactory name.

This interface can be used by an OpSim subsimulation in the following way:

- On initialization of the subsimulation proxy, readable and writable OpSim assets are received from the MCP core according to the scenario file. For these, PowerFactory assets can be added using the interface's *addAsse*t function,
- Setpoints for assets received from OpSim can be written onto the internal assets before executing a simulation step,
- During execution of a simulation step, Powerfactoy asset setpoint values may be arbitrarily modified and one or multiple load flow calculations may be carried out, e.g. in order to calculate additional setpoints,
- Results from the PowerFactory calculations can be fetched after each simulation step using the interface's getter function and written back to the OpSim message bus in case the asset is writeable.

In order for the interface to work properly, it is absolutely vital to name the PowerFactory assets, which are to be used in OpSim, exactly like in the devices definition file and in the physical grid model. Names also need to be unique. If the interface is used as described above, assets with different names may appear in the PowerFactory model, but are ignored by the subsimulation unless they are created and used for internal purposes.

A drawback of the approach is that execution of the Python script currently sets PowerFactory into a non-interactive mode. Although simulation results can be observed taking a look at the PowerFactory terminal output and other subsimulations, the PowerFactory user interface is rendered nearly unresponsive during execution. However, this seems to be less a problem of the interface implementation than a property of PowerFactory which is currently accepted as-is.

6. Conclusion and outlook

The previous chapters introduced the interfaces and means used for data transmission for implementing control functions and simulations in INTERPLAN project, which mainly rely on predefined or proprietary data formats, data models and interfaces. Their selection was primarily governed by the selection of the software tools used to develop and test the project's software solutions (predominantly control functions), or by the ability of said tools to import/export or trace specific data formats. All interfaces and data formats were found to be suitable to support the projects requirements without the need of substantial changes. They will thus be used for further development, test and validation.

With regards to co-simulation, an extensive set of software helpers for implementing INTERPLAN OpSim simulations was developed. A first proof-of-concept, consisting in the co-simulation of an INTERPLAN control function, was successful and is reported in deliverable D6.3. This paves the way for validating the INTERPLAN tool as described in D5.2 [3] by the example of one of the showcases. Validation of individual showcases with special consideration of control functions will be going on using simulations mainly based on PowerFactory, Python, and AMPL.

7. References

[1] M. Kosmecki et al.: INTERPLAN Deliverable 5.1: „INTERPLAN showcases", 2018, Online: https://interplan-project.eu/resources/.

[2] A. Khavari et al.: INTERPLAN Deliverable 3.1: "INTERPLAN use cases", 2018, Online: *https://interplan-project.eu/resources/*.

[3] M. Di Somma et al.: INTERPLAN Deliverable 5.2: "Operation planning and semi-dynamic simulation of grid equivalents", 2019. To be published online, *https://interplan-project.eu/resources/*.

[4] Fraunhofer IEE and University of Kassel: Pandapower. Online, http://www.pandapower.org/

[5] University of Kassel: SimBench - Benchmark-Datensatz für Netzanalyse, Netzplanung und Netzbetriebsführung. Online, https://simbench.de/de/.

[6] Digsilent GmbH: PowerFactory Anwendungen. Online, *https://www.digsilent.de/de/powerfactory.html*

[7] J. Ringelstein et al.: INTERPLAN Deliverable 6.1: "INTERPLAN scenarios which will be validated in the simulation", 2018. Online, *https://interplan-project.eu/resources/*.

[8] C. Töbermann et al.: Test- und Simulationsumgebung für Betriebsführungen und Aggregatoren im Smart-Grid (OpSim). Abschlussbericht des Projektkonsortiums OpSim, 2018. Online: *https://doi.org/10.2314/GBV:1029773750*.

[9] Fraunhofer IEE and University of Kassel: Update to pandapower 2.0. Online: *https://pandapower.readthedocs.io/en/v2.1.0/about/update.html*.

Annex

7.1 List of Figures

7.2 List of Tables

7.3 Glossary of terms and definitions

7.3.1 Definition of project general terms

Term	Definition
Use Case	The specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.
Sub Use Case	Description of a specific situation a use case is applied to. A Sub Use Case is always attributed to one (main) use case, but one use-case may have multiple sub use cases which detail the main use case in at least one aspect.
Base showcase	Presentation of base use case(s) with no planning criteria and no controllers for emerging technologies, such as RES, DG, demand response or storages in the frame of chosen scenario, simulation type, test model, and time-series data. The base showcase allows to analyze the operation challenges of the related use case(s) and improvements achieved through the application of planning criteria with related implementation of controllers in the associated showcase.
Showcase	Presentation of use case(s) in the frame of chosen scenario, simulation type, test model, time-series data and planning criteria
Scenario	Definition of a future situation applying to a well-defined time (most often year). A scenario can be fictional or predicted from the present situation. In INTERPLAN, scenarios describe the future situation of the European electric network, typically including grid topology, generation mix, loads and diffusion of EV, RES and storages.
Dynamic Simulation	A simulation experiment which considers the time dependent behaviour of a physical system, looking at events occurring in real-time operation, with a frequency of occurrence of less than one second of real time. The simulation may run faster or slower than real time, and may, despite the fast event frequency, span a total time interval of several hours real-time.
Semi-Dynamic Simulation (also: Quasi-Dynamic Simulation)	A medium- to long-term simulation experiment based on steady-state analysis, considering the state of a physical system at discrete steps of real time through user-defined time step sizes. The real time between the steps is at least one minute.
Grid Cluster	A group of grids and parts of grids with similar characteristics
Grid Equivalent	A simplified network model, which approximately behaves like an associated complex physical network or a group of physical networks. The grid equivalent thus is a representation of the physical network(s), which is typically used for a simulation experiment.
Control function	A set point definition, which is determined based on the goals of each use case. A control function defines the set points of specific elements

	(e.g. OLTC, DGs, RESs) or some programs (e.g. demand response) calculated by an operation objective in the network..
Interface	A means of transmitting information between two or more controllers or actors. It usually includes a specification about which information is to be transmitted, how this information is represented by data elements, and defines a physical means for transmission of those data elements.
Cluster Controller	A controller having the aggregated behavior of individual controller characteristic in a larger grid.
Interface Controller	A controller, which is intended to be installed in a specific "home" cluster, and uses information received through an interface from at least one other cluster data source outside the home cluster. This data source could e.g. be another cluster, but also e.g. an external weather forecast provider using an interface
Local Controller	A controller which is associated with a single specific generator, load or grid asset and which operation does not rely on remotely received information originating from any remote source. i.e. the operation only relies on information available within the local area network of the local controller's installation site.
Co-simulation	A simulation which consists of different parts that form a coupled problem and are modelled and simulated in a distributed manner (cp. Wikipedia). The parts are called "Co-simulation subsystems" and are exchanging data during the simulation. Different models and simulation means can be used in different subsystems. The Co-simulation (in the ideal case) is carried out by running the subsystems, which were individually tested and validated beforehand, in a black-box manner. In INTERPLAN, the data exchange between subsystems is done by the OpSim platform.
Co-simulation subsystem / Co-simulation subcomponents	A part of a Co-simulation which is developed, modelled and validated individually, while at the same time able to be integrated into the Co-simulation platform. In INTERPLAN, a subsystem might represent e.g. a DSO or TSO operation centre, a controller, or even the real physical network model.
Data model	An abstract model that represents a real-world entity, and defines, organizes and standardizes the description of the data elements related with that entity. Since real-world entities are typically consisting of other entities (e.g. an electric grid consists of lines, transformers etc.), a data model typically is hierarchically structured and also allows to define interrelations between entities.
V2G and G2V	Vehicle-to-grid (V2G) describes a system in which *plug-in electric vehicles* communicate with the *power grid* to sell *demand response* services by either returning electricity to the grid or by throttling their charging rate. When an EV is being charged, it's called G2V (Grid to Vehicle).

Allocation	With reference to the grid operation planning phase, it is the process deciding, which are the most suitable resources to commit and dispatch among *n* operating resources for a specific objective and under specific constraints.
Placement and sizing	With reference to the grid planning, it is the process deciding the most proper location (bus) and the size of a resource (active power) for a specific objective and under specific constraints.
Energy Not Supplied	Energy Not Supplied is defined as the amount of energy that would have been supplied to the customer if there had been no interruption.
Energy spillage	Energy spillage is the production (from Solar and Wind) that is unable to be accommodated due to demand being lower than production.

7.3.2 Definition of actors

Term	Definition
TSO - Transmission System Operator	Natural or legal person responsible for operating, ensuring the maintenance of the transmission system and, if necessary, developing the transmission system in a given area and, where applicable, its interconnections with other systems, and for ensuring the long-term ability of the system to meet reasonable demands for the transmission of electricity. The term 'transmission' means the transport of electricity on the extra high-voltage and high-voltage interconnected system with a view to its delivery to final customers or to distributors, but does not include supply.
DSO - Distribution System Operator	A natural or legal person responsible for operating, ensuring the maintenance of and, if necessary, developing the distribution system in a given area and, where applicable, its interconnections with other systems and for ensuring the long-term ability of the system to meet reasonable demands for the distribution of electricity. The term 'distribution' means the transport of electricity on high-voltage, medium-voltage and low-voltage distribution systems with a view to its delivery to customers, but does not include supply.
ESCO	Electricity supply company (sometimes also: Electricity service company). General term for a company which supplies end users with electric energy. An ESCO may offer additional services, e.g. electricity generation, metering or supply with non-electric energy.
Prosumer	Active energy consumer who consumes and produces electricity. Various types of prosumers exist: residential prosumers who produce electricity at home - mainly through rooftop PV, citizen-led energy cooperatives, commercial prosumers whose main business activity is not electricity production, and public institutions.
Generator	A device which produces electricity.
Load	A device which consumes electricity.
Producer	A natural or legal person generating electricity.

Consumer	A natural or legal person consuming electricity.
Distributed Energy Resource (DER)	A source or sink of electric power that is located on the distribution system, any subsystem thereof, or behind a customer meter. DER may include distributed generation, electric storage, electric vehicles and demand response.
Aggregator	Company who grouping distinct agents in a power system (i.e. consumers, producers, prosumers, or any mix thereof) to act as a single entity when engaging in power system markets (both wholesale and retail) or selling services to the system operator(s).
Distributed generation (DG) unit	Any source of electric power of limited capacity, directly connected to the power system distribution network. DG can be powered by photovoltaic system, micro-turbines, combustion engines, fuel cells, wind turbines, geothermal, etc.
Flexible Loads	A load which consumption can be influenced in terms of power, time, or total energy consumed while still serving its intended purpose. The influence may be exerted by manual means (e.g. switching the load on or off at arbitrary times) or automatic means (e.g. external control signal).