FlexPlan

Advanced methodology and tools taking advantage of storage and FLEXibility in transmission and distribution grid PLANning

Planning tool software, including GUI

D3.1

Distribution Level	CO (PU: this file)
Responsible Partner	N-SIDE
Checked by WP leader	Maxime Hanot (N-SIDE) – WP3 Leader Date: 07.11.2022
Approved by Project Coordinator	Gianluigi Migliavacca (RSE) Date: 08.11.2022

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 863819

Issue Record

Planned delivery date	30.06.2022
Actual date of delivery	08.11.2022
Status and version	FINAL

Version	Date	Author(s)	Notes
0.1	30/09/2022	Alberte Bouso (N-SIDE) Maxime Hanot (N-SIDE) Samuel Monroe (N-SIDE)	Initial version.
0.2	15/10/2022	Alberte Bouso (N-SIDE) Maxime Hanot (N-SIDE)	Corrections and sections 4.3, 4.4, and 5.
0.3	07/11/2022	Alberte Bouso (N-SIDE) Maxime Hanot (N-SIDE)	Corrections, typos and small additions.

About FlexPlan

The FlexPlan project aims at establishing a new grid planning methodology considering the opportunity to introduce new storage and flexibility resources in electricity transmission and distribution grids as an alternative to building new grid elements. This is in line with the goals and principles of the new EC package *Clean Energy for all Europeans,* which emphasizes the potential usage of flexibility sources in the phases of grid planning and operation as alternative to grid expansion. In sight of this, FlexPlan creates a new innovative grid planning tool whose ambition is to go beyond the state of the art of planning methodologies, by including the following innovative features: integrated T&D planning, full inclusion of environmental analysis, probabilistic contingency methodologies replacing the N-1 criterion as well as optimal planning decision over several decades. However, FlexPlan is not limited to building a new tool but it also uses it to analyse six regional cases covering nearly the whole European continent, aimed at demonstrating the application of the tool on real scenarios as well as at casting a view on grid planning in Europe till 2050. In this way, the FlexPlan project tries to answer the question of which role flexibility could play and how its usage can contribute to reduce planning investments yet maintaining (at least) the current system security levels. The project ends up formulating guidelines for regulators and for the planning offices of TSOs and DSOs. The consortium includes three European TSOs, one of the most important European DSO group, several R&D companies and universities from 8 European Countries (among which the Italian RSE acting as project coordinator) and N-SIDE, the developer of the European market coupling platform EUPHEMIA.

Partners

Table of Contents

List of Abbreviations and Acronyms

Abbreviation/Acronym	Meaning
GUI	Graphical User Interface
WP	Work Package
API	Application Programming Interface
OPF	Optimal Power Flow
TNEP	Transmission Netqork Expansion Problem
AWS	Amazone Web Services

Executive Summary

This deliverable includes a description of the software for the new FlexPlan planning tool and its Graphical User Interface. In particular, the report provides an overview of the software architecture and of the adopted security measures. It also presents a critical discussion on the technology alternatives for the different components of the final tool and provides the reasons for the adopted choices. Moreover, the document covers a description of the different elements of the tool, highlighting the main implementation challenges and concludes by summarizing the adopted testing procedures.

Regarding the software technology, a large list of possible alternative choices for the different software modules is proposed. Then, based on some criteria and the examination of the different options, the most suited technology is chosen and adopted for the final tool. The same process is repeated for programming language, solver and input-output format.

Once the agreement of the developers is obtained on the most suited technologies for the project, the next steps define the general architecture of the tool: workflows, connections and call-backs. Another relevant must-have feature is the inclusion of security measures, as depicted in the present deliverable.

The authors briefly describe the developed software, by covering: optimization engine, integration with the pre-processor (generation of expansion candidates) and scenario reduction. Such a development presented important challenges, which are detailed in a dedicated section of this deliverable.

During the overall software engineering process, to ensure a reliable integration of the new features, the contributors implemented a set of testing procedures. This process is also illustrated in detail.

1 Introduction

The FlexPlan tool implements a grid expansion optimization engine able to assess flexibility sources candidates along with customary elements, such as new lines and cables. Flexibility sources are meant to include different storage technologies as well as demand-side management. Moreover, the tool assesses candidates in both Transmission and Distribution networks, which bring the possibility to optimize the procurement of flexibility on both sides of the network simultaneously.

Figure 1.1 shows the outline of the tool, covering the input data, the optimization engine and the types of available investment decisions. Due to the size of the problem at hand, it is of utterly importance to select highly efficient technologies that are suited for the specific case. Therefore, the team analysed the best technologies to carry out each of the tasks (modelling, transferring of data, storage, flows of information).

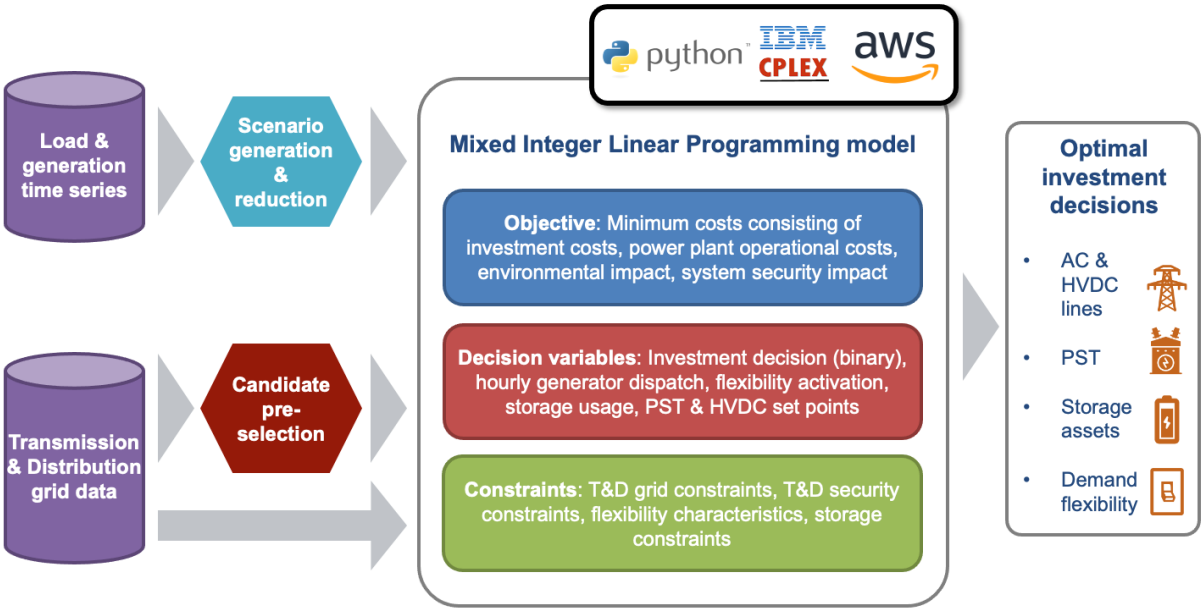

Figure 1.1 – High-level outline of the planning tool features

Another important point is the need to integrate the FlexPlan pre-processor (i.e., the application creating the list of the candidates to be assessed), which is an external application. This tool is deployed by using a docker image manager, known as Kubernetes [1]. The pre-processor is a heuristic-based algorithm which can propose investment candidates for a network expansion problem, based on the outputs of an optimal power flow from a given grid. The methodology behind the pre-processor is out of the scope of this deliverable and detailed in Deliverables 2.2 [2] and 2.3 [3] of the FlexPlan project.

2 Technology choices

This section describes the first phase of the software development of the FlexPlan tool: i.e., the choice of the most suitable software technologies to use.

2.1 Language

First, the team discussed which language should be chosen for the backend of the tool. It should be noted that, among other requirements, this part of the software should allow to model a large-scale optimization problem, manage big amounts of data (grid model, and scenario time-series) and parse the results to the Graphical User Interface (GUI). Moreover, the language should be a standard within the skillset of the software engineers working in the project. Four candidates were discussed. They are listed below. Pros and cons are presented.

- **Python** [4], is a standard language among software engineers, data scientists and energy engineers. It provides a good integration with different commercial solvers for optimization (e.g., DOcplex) and has thousands of packages open-source available for the developers. However, it is known that moving from a prototype to a product may be challenging, although progress has been done in this regard in recent year, and several production tools are compatible with Python nowadays.
- **Java** [5], is a common language used for developing and producing large optimization problems, with a strong integration with commercial solvers and facilities when it comes to moving into production from a prototype. However, the knowledge of the language is less common among the operations research community.
- **Scala** [6], is popular among N-SIDE software developers, provides versatile features for the task. However, the knowledge is not so widespread and it provides a harder integration with commercial solvers.
- **Julia** [7], is an emerging programming language for scientific purposes and very popular among the operations research community. It was considered as a suitable choice for the project. Nevertheless, the team found this option to be risky as there is less documentation available online as well as integration with production tools.

Python was chosen as the language for the backend because this is a popular choice within the scientific community and due to its advantages when it comes to prototyping, as well as for being robust and known among the available team.

2.2 Solver

Second, the backend should be connected to a solver for solving the large-scale Mixed Integer Linear Program. Commercial and open-source software is available for this purpose. However, due to the expected complexity of the tool, commercial options may be preferred, proven they are cost-efficient. The team based the technical assessment on references such as [8], [9] and [10]. As a result, several solvers were considered:

- **CPLEX** [11], offering a high-performance, with a good interface with Python and Java, has the only drawback to be a commercial software. However, the license is available at no cost to the developers of the tool.
- **GUROBI** [12], also offers high-performance and interfaces with common languages. However, it is a costly commercial solver and the developers of the tool do not already have a license of this software.
- **GLPK** [13], open-source, without Python or Java interface, and low performance compared to commercial solvers.
- **SCIP** [14], open-source but with license need for commercial use, with Python and Java interface, and mid performance compared to commercial solvers.
- **CBC** [15], open-source, with Python interface, but low performance compared to commercial solvers.
- **lp_solve** [16], open-source, with Python and Java interface, but very low performance compared to commercial solvers.

Finally, the team opted for choosing CPLEX.

2.3 Interface with solvers

The next steps cover the technology to be used as an interface between the programming language and the solver. The classic approach requires putting model constraints and objective form in the correct form (large matrix) for the solver. Such a process is generally tedious, error-prone, hard-to-debug, using solver-specific formats. However, nowadays solvers have more friendly Application Programming Interfaces (APIs) and open-source options are available that are compatible with multiple solvers at the same time. Modelling languages are designed to simplify the development of applications based on optimization, by allowing to write the model in human-readable form, thus simplifying maintenance, reducing complexity and allowing an easier switching to other solvers (as long as the precise API exists). Therefore, the team assessed the libraries listed in the following. Note that at this stage, CPLEX was already selected as the reference solver.

- **DOcplex** [17] Python Modelling API, a native Python library for modelling optimization. Although it only works with CPLEX, it is compatible and optimized for common libraries such as NumPy and pandas, and it is easy to use for modelling optimization problems.
- **IBM Python interface for the CPLEX Callable Library** [18]. The library, built as a default interface with CPLEX from Python, uses less memory and CPU than other modelling languages and has a high solver interaction capability. Among the negative aspects, it is only valid for CPLEX, not easy to model and is a commercial solution.
- **IBM ILOG Concert Technology** [19]. Similarly, to the previous, but for Java instead of Python.
- **IBM Optimization Programming Language** [20], offers a similarity of syntax with mathematical formulation. In the same fashion as other IBM solutions, is commercial and only compatible with CPLEX, while requesting high memory and CPU and limited interactions available.

- **AMPL** [21], offers a similar syntax as mathematical formulation, with the added value of being solver agnostic. On the contrary, it does not provide data manipulation capabilities, high memory and CPU usage and being a commercial solution.
- **GAMS** [22], same as AMPL.
- **PYOMO** [23], built specifically for Python, this tool is open-source and has multiple APIs with different solvers. On the contrary, high memory and CPU usage are expected.
- **Google-OR** tools [24], has an interface with Java and Python, open-source, and compatible with most common solvers (Gurobi, CPLEX, CBC…). However, a high memory and CPU overhead are expected.

The team chose DOcplex as the interface between CPLEX and Python. It was selected as it was the right compromise between easiness of modelling and performance.

2.4 Input-Output format

The workflow of the developed software involves interaction between different modules, each one led by a different developer. A considerable amount of information must be exchanged between the tool users (the regional cases developed by FlexPlan), the optimization engine developers and developers of the pre-processor tool. Thus, it needs to be (a) customizable, (b) easy to understand, and (c) lightweight. The options considered in order to comply with the above needs were:

- Custom **JSON** [25] files, which are common lightweight files, standard for HTTPS APIs, and they are easy to translate to a data model. However, they may be hard to manipulate directly for humans.
- **CSV** [26] files, can be opened with Excel and are also a common standard practice. On the other hand, the size can become a problem and may not be convenient to model electrical grids.
- **XLSX** [27] files, are easy to be manipulated and read for human. However, they are commercial solutions, difficult to parse and a long time is expected for loading and saving.
- **CIM/CGMES RDF XML** [28], is a standard format pushed by ENTSO-e and have detailed information about the grid. However, these files can be really heavy, difficult to manipulate for humans and they are not yet fully mature.
- **UCTE-DEF** [29], have a light text format and are an existing European standard. However, they are not future proof as they should be replaced by CGMES files.
- **.raw PSS/E** [30], are proprietary solutions, although they present an attractive format since they are composed by light text.

Therefore, the format chosen by the team for the exchange of information is JSON, which constitutes a common practice for both industry and research. Among other properties, JSON allows to represent objects in many programming languages as human-readable text that can be easily exchanged through APIs and databases. Also, it is easy to parse and generate for machines, which reduces the execution time of the tool. Furthermore, it is supported by Python natively (both for mathematical modelling or results generation, as well as for internal and external communication through several API layers), meaning the expected performance when reading the files is high, as well as numerous useful built-in functions are available.

2.5 Graphical User Interface

The Graphical User Interface (GUI) is one of the most challenging aspects of the implementation, as it needs to provide a user-friendly design, while being able to connect and coordinate the different engines built in the software. Based on the knowledge and past experience of the developers of the tool, Django [31] is being used for the back-end of the GUI and Vue.js [32] is being used for the frontend. The main parts and features of the Graphical User Interface are depicted in the following subsections.

2.5.1 User Management

Among others, one of the main features included under the user management section covers:

- **Login**, as a user, in order to access the FlexPlan engine.
- **Password update**, as a user, in order to secure the account
- **User profile**, to visualize as a user your account information in the application

2.5.2 Simulations Dashboard

The simulation dashboard includes the considerations gathered during the design phase from the different potential users from the tools (mainly partners and system operators). Some of the features covered include:

- **List of the simulations,** in order to track the simulations launched by the user, in a table format, and have a quick view of the status and important information
- **Create a simulation**, in order to trigger the FlexPlan engine to launch new simulations
- **Simulations ownership**, in order to keep sensitive data private, such as other users cannot access your simulations
- **Simulations search**, to find quickly previous simulations launched
- **Simulation results download**, in order to download the JSON output
- **Simulation details**, in order to browse key metrics and summarized information about a simulation's results owned by the user

One of the attention points is that the expected input files can weight up to 500 MB per year, and the data contained in the input files is mandatory for further computation. Moreover, the data is structured, so the tool can reflect this modelling in a relational database. Finally, a change in the input file means a new simulation and new results.

2.5.3 Visualization Tool

In order to visualize the grid as it is today (including where it might be congested), as well as the results from the pre-processor (expansion candidates), and the expanded grid (selected candidates), we need to ensure the technology choice fulfils the following criteria:

- It has to be **interactive**, such as the user should be able to click on vertices and get information as popups. Moreover, the user should be able to hover over elements and get information as popups. The user shall be available to move around the map, add or delete elements on the fly.
- It has to take care of the **geographical** aspect, since the grid model is a geographical construct and the user should be able to see it as is.

- It has to be as **lightweight** as possible, since the tool should be able to display a lot of elements, as well as handling them. A mitigation measure that can be implemented is using a cluster of what is displayed.
- It has to be **customizable**, both functional and visually. The first one implies that the package of choice should be compatible with new functionalities within the maps. The second one, less of a preference, the tool should allow adapt the style to the FlexPlan branding, and the lines for drawing the power grid can be modified in colour and thickness.

Based on these criteria, the following candidates were selected: Mapbox [33], Plotly [34], Maplibre [35], Google Maps [36] and Deck [37]. The final choice was **Maplibre**, since it offers out of the box a large set of features, and is an open-source library. It would provide a fast way to implement a first version of the tool.

3 Software architecture and security measures

Figure 3.1 presents an overview of the architecture of the planning tool developed for the FlexPlan project. In this picture, we present mainly the backend, as we consider that the details for the GUI are out of the scope for this report.

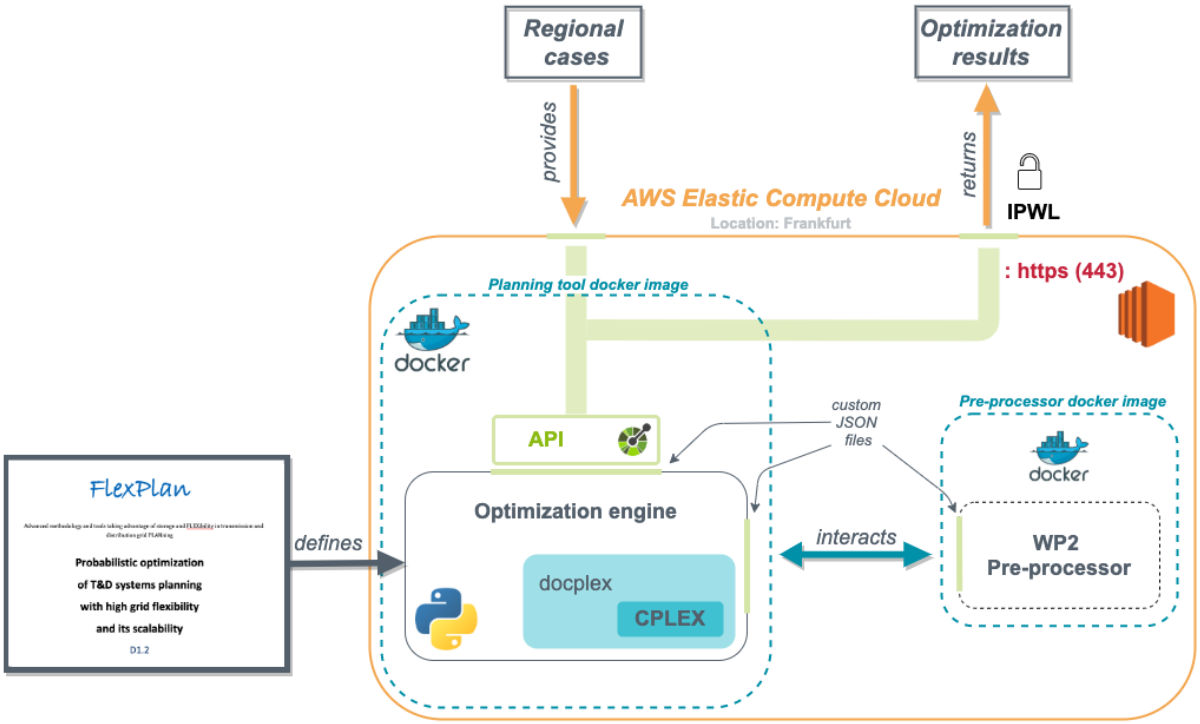

Figure 3.1 – Global architecture for the planning tool engine

The core of the solution builds upon the optimization engine, which uses Python as the main programming language and CPLEX as the optimization solver. DOcplex serves as an interface between both tools. The definition of the set of constraints and the objective function of the multiple optimization problems solved is depicted in Deliverable 1.2 [38] from the FlexPlan project.

In order to interact with external users, as well as the pre-processor, several APIs are developed, so as to allow the planning tool to exchange the different input and output files written in the JSON format. The APIs are defined using OpenAPI specifications 3.0 [39]. The workflow is established as follows:

- The end-user provides the grid model and the scenario data to the planning tool.
- The planning tool solves an optimal power flow on the grid without any investment (first optimization problem), to identify bottlenecks of the power system, as well as the nodes with higher need for flexibility. Those results are then sent and analysed by the pre-processor to do a selection of potential candidates.
- The output of the pre-processor is handed to the planning tool and used to build the optimization model with investment candidates in the engine, known as transmission network expansion planning (TNEP) problem.
- The end-user receives the final results (candidate selection), through the API.

Both the optimization engine and the pre-processor are built as docker images, following industry best practices. In order to deliver the tool to different partners, as well as provide enough computing power, the overall solution is hosted on Amazon Web Services (AWS) servers [40]. A dedicated server was employed for one of the regional cases, due to a peculiar simulations data confidentiality.

Concerning the security measures, the software development team has implemented the following privacy features:

- **IP whitelisting**, a security feature used to limit and control the access to the tool to trusted users
- Extension of the HTTP protocol to **HTTPS**, commonly used for secure communication over a computer network (encryption during transfer).
- **Basic authentication,** ensuring that each regional case has its own username and password, so their simulations can only be accessed by those who have those keys available.

Further security layers were considered, such as a Web Application Firewall or Encryption during processing, but these were not included in the final version, since it was thought that the aforementioned measures were sufficient for the purpose of this application.

4 Tool description

This section covers in detail the description of the software developed under the FlexPlan project. In particular, the API, the optimization engine, the integration with the pre-processor and the scenario reductions are covered, providing details on how the tool was developed and what characteristics it includes.

4.1 API

The APIs, developed to receive and send data, are used to interact between the Optimal Power flow (OPF) solver, the pre-processor, the Grid Expansion Planning (GEP) solver, and the Graphical User Interface (GUI). The workflow is presented on Figure 4.1. The descriptions of the data models used for the interactions (Steps 1 to 4 on the Figure 4.1) are detailed in the tables 4.1 to 4.4. Examples of generic parameters are given at the end of this subsection.

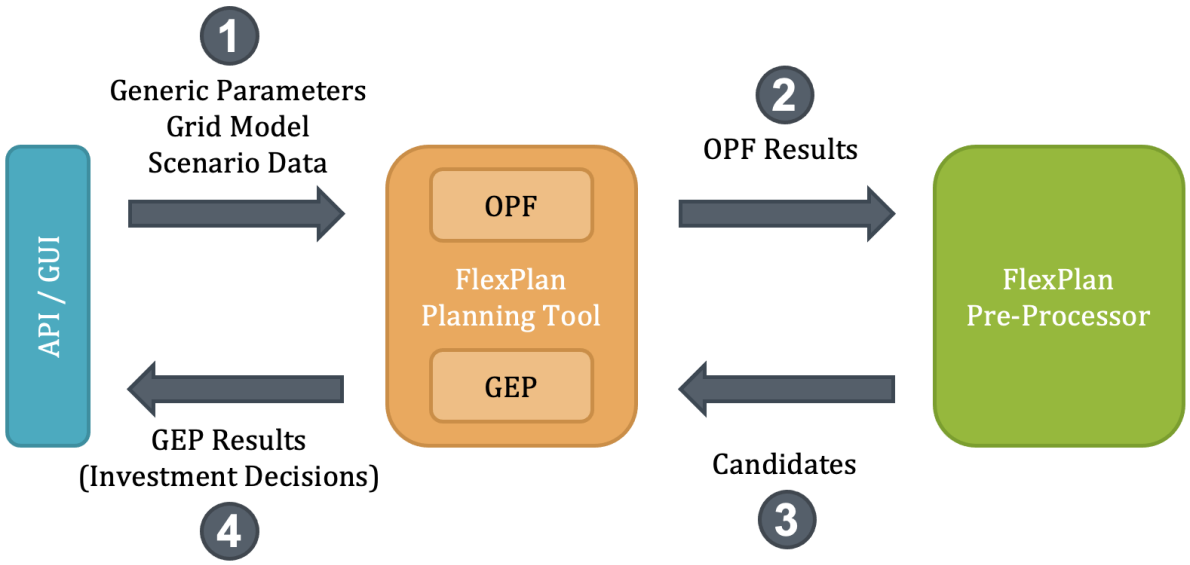

Figure 4.1 – The workflow between the Planning Tool and the Pre-Processor.

Step 1	
Grid Model	· Includes the topology of the power system, providing information related to buses, lines, converters, transformers, loads, generators, and storage devices. · This data model also includes equipment technical characteristics, i.e., static values which do not vary through the year (e.g., generators' nominal capacity or branches' thermal rating). · It is similar to equipment profile (EQ) data files, used in the Common Grid Model Exchange Standard (CGMES) by ENTSO-E.

Scenario Data (2030/2040/2050)	· Time-series with data related to buses, loads, generators, and energy storage devices. · Here it is included data which is time-dependant through a year (e.g., load and generation profiles). · Similar to the steady-state hypothesis (SSH) data files, used in CGMES.

Table 4-1 – Data models for OPF, composed of a grid model and future scenarios

Step 2	
Optimal Power Flow (pre-investment)	· Covers the outcomes of the non-expanded optimal power flow for years 2030, 2040 and 2050. · Expressed as Locational Marginal Prices (LMPs) at the buses, power flow directions through branches, transformers and converters and Lagrange Multipliers (LMs) associated with each branch. Additionally, the PTDF matrix of the network is also provided.
Grid Model	· Same as the grid model in Step 1

Table 4-2 – Output model of the OPF, composed of the grid model as well as the outcomes of the non-expanded OPF

Step 3	
Investment Candidates	· Characteristics of the investment candidates. · Expressed as a combination of resource location, type of flexibility technology candidate, alongside with its size and cost.
Grid Model	· Same as the grid model in Step 1
Scenario Data	· Same as scenario data in Step 1

Table 4-3 – Data model to run a stand-alone grid expansion planning problem, composed of investment candidates, grid model and future scenarios

Step 4	
Investment decisions	· List of the optimal investment decisions based on the provided candidates expressed as Boolean variables.

Table 4-4 – Output model of the expansion planning problem, composed of investment decisions

Moreover, the end-user can also provide additional parameters to fine tune the definition of its regional case. Those parameters are included in the generic parameters (Step 1 on. Figure 4.1). Those parameters cover, on one hand, the meta parameters of the simulation such as:

- **The number of scenarios** included in the input files, as well as the probability of each scenario
- **The number of years** covered in the process, as well as the number of hours for each year and scenario

- **The reference year** used as reference for the discounting of investment and operational costs
- **The maximum number** of candidates per node
- The use or not of the **Transmission & Distribution decomposition**
- **The base power** used for the per unit notation in the input files

On the other hand, the user can define parameters used for the tunning of the solver, i.e., CPLEX. The user can define the following parameters:

- **CPXPARAM_Read_Scale**: decides how to scale the problem matrix.
- **CPXPARAM_Barrier_StartAlg**: sets the algorithm to be used to compute the initial starting point for the barrier optimizer.
- **CPXPARAM_Barrier_Crossover**: decides which crossover is performed at the end of a barrier optimization
- **CPXPARAM_Benders_Strategy**: specifies whether CPLEX should apply Benders algorithm as a strategy to solve a model
- **CPXPARAM_LPMethod**: controls which algorithm is used to solve continuous linear models or to solve the root relaxation of a MIP.
- **CPXPARAM_MIP_Tolerances_MIPGap**: sets a RELATIVE tolerance on the gap between the best integer objective and the objective of the best node remaining.
- **CPXPARAM_TimeLimit**: Ssts the maximum time, in seconds, for a call to an optimizer (for an OPF, this the time per variant and per year).
- **CPXPARAM_MIP_Tolerances_AbsMIPGap**: sets an absolute tolerance on the gap between the best integer objective and the objective of the best node remaining.

4.2 Optimization engine

As mentioned earlier, the optimization problem is built using DOcplex library. Since Python is an object-oriented language, the team has developed a set of classes which create objects for the different elements in the grid, such as:

- AC branches (incl. transformers
- DC branches
- AC/DC converters
- Storage units
- Loads (flexible and non-flexible)
- Generators
- PSTs
- AC and DC buses

Each element of the grid has associated a set of variables and constraints that need to be declared for each asset, both OPF and TNEP problems. Therefore, when reading the input data and creating the optimization model, every time there is an AC bus in the system, the model will generate an object that adds the needed optimization elements to the master problem (e.g., nodal balance constraint). The same process is repeated for each asset class.

Two master problems are formulated, the OPF and the TNEP, which are built upon the input data received in the optimization engine. For each problem, we first initiate the pertinent DOcplex model, followed by setting up the instances of the internal models (i.e., grid assets). Finally, we set the corresponding objective function. Once all the models have been declared, we proceed to solve the optimization problem and retrieve the solutions of the variables of interest (e.g., the dispatch of power plants, the flow through the branches, etc.).

4.3 Integration with pre-processor

As specified in section 3 of this report, the pre-processor has been developed as a separate software of the planning tool. Nevertheless, it has to be integrated within the planning tool software. Such a coordination between both tools is done via orchestration of Docker containers. Indeed, the pre-processor algorithm is deployed in a separate Docker on the same AWS servers than for the planning tool. The input and output files are presented in Tables 4.2 and 4.3, respectively.

4.4 Scenario reduction

In order to manage and extract representative weeks from the large time-series constituting the future scenarios (e.g., load profiles, generation from renewable energy, etc.), the team also implemented the so-called scenario reduction algorithm, depicted in Figure 4.2

The developed software relies on the implementation of the k-means algorithm, from the scikit-learn Python library [41]. The end goal is to reduce to k^* dimensions the original k dimensions, for each node of the network. Such reduction can be performed yearly, monthly, weekly and daily. More details can be found in Deliverable 1.1 [42].

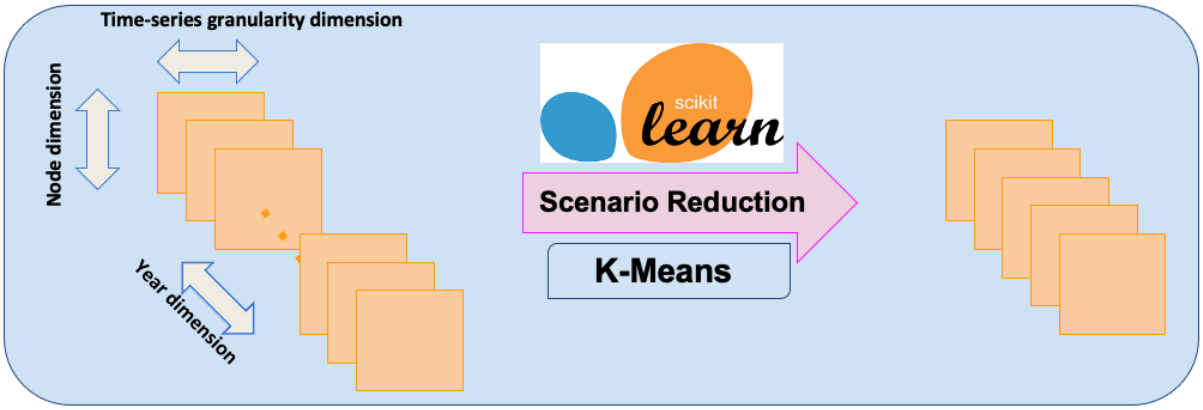

Figure 4.2 – Visual representation of the scenario reduction algorithm

The integration of the service within the tool is presented in Figure 4.3. The user can request the service in a similar fashion as the grid expansion planning, the optimal power flow or the full FlexPlan process.

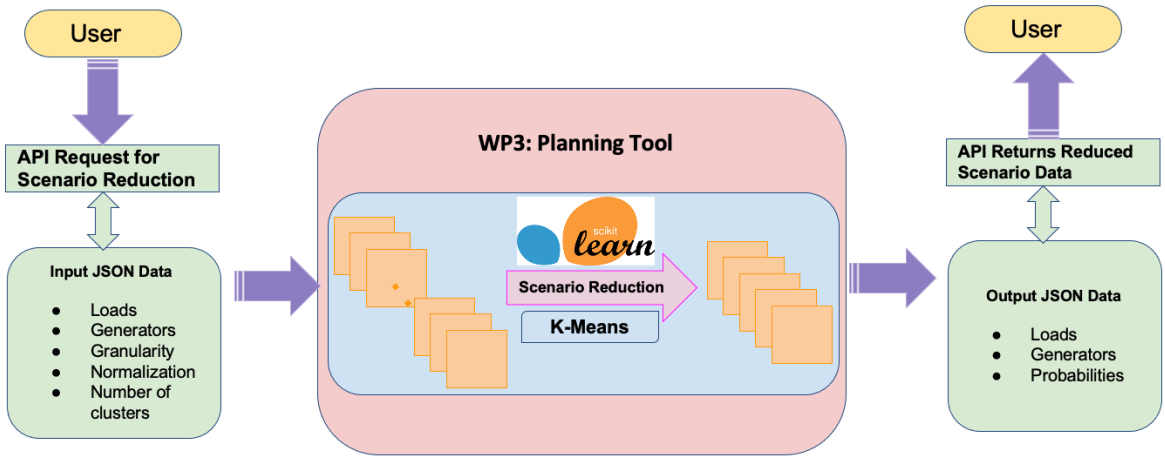

Figure 4.3 – Workflow of the scenario reduction algorithm within the FlexPlan tool

5 Implementation challenges

Numerous challenges have been encountered during the development of the tool, among all:

A. Incongruences in the results obtained from the proof-of-concept testing of the mathematic models, provided by another team, and the results obtained by the FlexPlan planning tool
B. Large and complex cases to solve, both computational and in size
C. Privacy concerns

Concerning (A), the team worked closely with the developers of the proof-of-concept library (FlexPlan.jl – see Deliverable D1.2 for details [38]) to ensure a correct formulation of the optimization problem, by verifying that, given the same input file, both tools would output the same results. The approach is detailed in Section 6. Some tests, in particular for the distribution system formulation and the T&D Decomposition, resulted challenges. For instance, multiple optimal solutions equally valid arise as solutions of the problem. However, thanks to a smooth collaboration and frequent exchanges between both teams, implementation errors and incongruences were identified and solved.

(B) was, by far, the main challenge the development experience. While some of the first limitations were overcome by increasing the size of the servers (in terms of memory available, and computing power) and implementing two types of decompositions:

- Benders decomposition,
- Transmission and Distribution decomposition,

it was needed to reduce the size and complexity of the Regional Cases. In particular, it was found that units that interconnect multiple time-stamps (such as storage units and pumped hydro) tend to scale exponentially the difficulty of the optimization problems. Similarly, increasing the number of candidates in the transmission system is one of the main parameters defining the computational complexity of a regional case.

Finally, due to privacy concerns, i.e. (C), it was first needed to implemented several security mechanisms described in chapter 3 of this document. Moreover, it was needed to integrate the planning tool software with the data storage asset of regional case owners, such as the input data wouldn't be persisted outside of the premises of the regional case owner. By doing so, the team ensured the right handling of data for the users of the tool.

6 Testing

Different types of testing are usually employed in software development projects. For instance:

- **Unit testing**: used to verify the behaviour of the smallest testable parts of an application (units), which are properly and independently scrutinized
- **Data testing**: which usually relies on an input dataset, ensuring that it verifies the execution preconditions and input content required to execute a unit test
- **Integration testing**: testing of several units of a software as a combined entity to ensure that the different components of the software behave correctly together
- **Performance testing**: to test the run-time performance of a software and fine-tune its implementation and parameters in order to speed-up the program

In addition, **parallel testing** can be used in order for both developers and testers to concurrently test several components of the application with the goal to reduce the total test time.

As the codebase matures, more and more tests are added to guarantee sustainable growth without corrupting previous functionalities. Therefore, the team has implemented a standard procedure to establish the best practices available, without compromising the speed of the development. In particular, the team developed unit and integration tests for the different optimization problems, using data testing (i.e., example files of input data), verifying that they meet the APIs requisites.

6.1 Unit and integration testing

The general procedure followed by the developers is depicted in Figure 6.1. By extracting the output files after an optimization problem in the proof-of-concept library [38], it was possible to obtain the expected results of a simulation. Then, the team transforms, via a converter, the input file used in that test case (usually from a small case, e.g., IEEE6 test case) into the FlexPlan API.

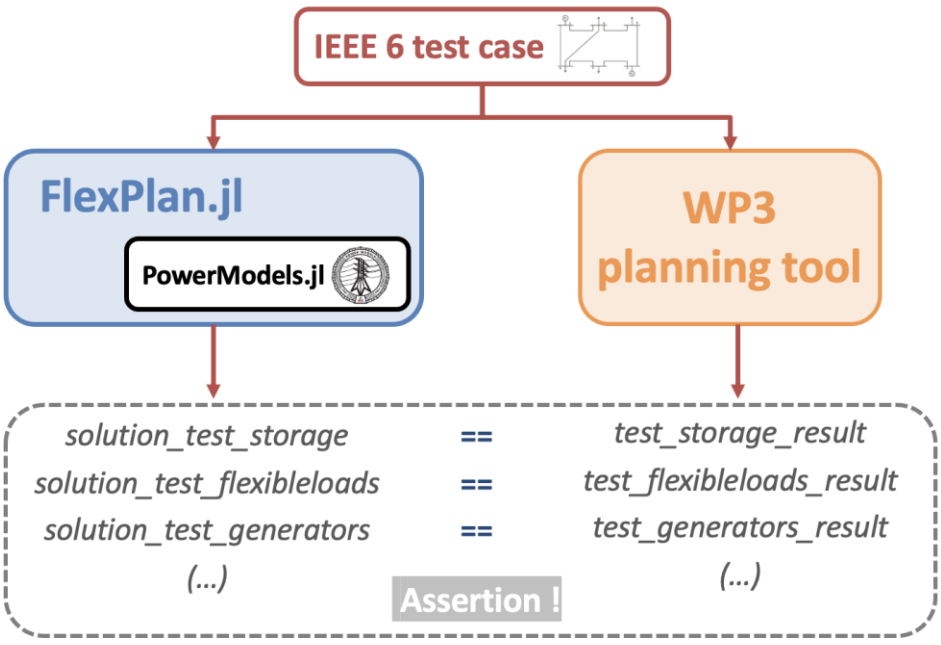

Figure 6.1 – General behaviour and structuring of the tests in the FlexPlan tool

Once this is done, it is possible to run an optimization problem with the FlexPlan tool. Finally, an automated assertion of the results is done by comparing the expected results (from the proof of concept) and the simulation results (from the FlexPlan planning tool). Following this procedure, the team implemented:

- Tests for the different parts of a power system:
  - Tests for AC systems (incl. transformers)
  - Tests for DC systems
  - Tests for generators (conventional and RES)
  - Tests for loads (flexible or not)
  - Tests for storage units
  - Tests for distribution systems
- Tests for per-unit and absolute units
- Tests for reliability costs
- Tests for PTDF matrix generation
- Tests for the different types of problems to be solved:
  - Tests for multi-year Optimal Power Flow (OPF), single and multi-period
  - Tests for the Grid Expansion Planning problem, single and multi-period
  - Test for the T&D decomposition (including creation of surrogate models)

In order to verify that a software is sufficiently tested, a developer might want to compute the testing coverage of its software. The coverage is defined as a software testing metric that determines the number of lines of a software that are successfully validated under a test procedure. It allows to analyse how comprehensively a software is verified, and, therefore, reliable. Of course, the target for a developer is always to come as close as possible to 100% of testing coverage. However, in practice, it is often difficult to reach this target as it can be really time-consuming to test all the possible corner cases in a software.

The FlexPlan planning tool has the following coverage of the code via the tests described before:

- API: 80%
- Models (only optimization): 90%

Tables 6.1 and 6.2 give the details of the testing coverage for the different optimization problems solved by the FlexPlan planning tool and for the different types of assets of a power system.

Class	**Statements**	**Missed**	**Coverage**
OPF	532	92	83%
GEP	352	16	95%
T&D decomposition	350	19	95%
TOTAL	**1234**	**127**	**90%**

Figure 6.1 – Testing coverage for the different optimization problems solved by the FlexPlan planning tool

Class	Statements	Missed	Coverage
AC Bus	72	6	92%
AC Branch	223	9	96%
DC Bus	34	4	88%
DC Branch	25	0	100%
Converter	46	0	100%
PST	50	40	20%
(Flexible) load	139	3	98%
Generator	51	1	98%
Storage	115	3	97%
TOTAL	**755**	**66**	**91%**

Table 6.2 – Testing coverage for the different elements of a power system

It should be noted that the 10% of the missing coverage for the models is mainly due to the implementation of Phase Shifting Transformers (PSTs), since this is not implemented in the FlexPlan.jl proof-of-concept, and thus we cannot verify the exactness of the results. However, the behaviour of these models has been verified qualitatively.

Following the best practices of versioning, every time the tool suffers a modification (e.g., new feature added, bug fix), all tests are run in the new code. This allows the team to identify potential issues or damages in the structure of the code before any deployment is done in the servers, and thus available to the end-users.

6.2 Performance testing

In complement, to the unit and integration testing, a few performance tests have also been performed. This has been done mainly to fine-tune the CPLEX parameters for the resolution of the non-expanded optimal power flow as well as for the testing of the performance of the Benders decomposition implementation.

Those performance tests have been performed on a 2018 MacBook Pro with the following specifications:

- Processor: 2,2 GHz 6-Core Intel Core i7
- Memory: 16 GB 2400 MHz DDR4
- Graphics: Intel UHD Graphics 630 1536 M
- CPLEX: version 22.1

Table 6.3 shows the results of a non-expanded optimal power flow for different sets of CPLEX parameters: Simplex or Barrier method, Primal or Dual crossover, scaling or not. The case used for this test

correspond to the Italian regional case grid with one target year (2030) and one load and generation variant with 84 hours. It highlights the importance of choosing the right algorithm to solve an optimization problem. Indeed, the computation time is reduced by a factor 5.7 between the standard CPLEX parameters and the fine-tuned parameters.

	Simplex	**Barrier + dual crossover**	**Barrier + primal crossover**	**Barrier + primal crossover + scaling**
Time	2612 seconds	> 3000 seconds	880 seconds	458 seconds
Objective value	51961917	/	51961917	51961917

Table 6.3 – performance of the non-expanded optimal power flow

Table 6.4 shows the results of a grid expansion planning problem for different algorithm:

- Without Benders decomposition
- With Benders decomposition, with the decomposition automatically generated by CPLEX (automated annotations)
- With Benders decomposition, with the decomposition supplied by the Flexplan planning tool developers (custom annotations)

The case used for this test correspond to the IEEE67 grid with three target years (2030, 2040 and 2050), three load and generation variants per year and 24 hours per variant. The CPLEX problem has 95442 variables, including 174 binaries.

It shows that decompositions have the potential to speed-up optimization problem resolution but that the decomposition has to be correctly defined. Indeed the automated decomposition is leading to a 5 times increase of the computation time while the custom decomposition is leading to a 3 times decrease of the computation time.

	Without Benders decomposition	**With Benders decomposition: automated annotations**	**With Benders decomposition: custom annotations**
Time	1079 seconds	5019 seconds	323 seconds
Objective value	2752.375	2752.375	2752.375

Table 6.4 – performance of the Benders decomposition

References

[1] The Linux Foundation, “Kubernetes,” [Online]. Available: https://kubernetes.io/.

[2] FlexPlan project, “ D2.2 Flexibility elements identification and characterization,” [Online]. Available: https://flexplan-project.eu/wp-content/uploads/2020/06/D2.2_20200622_V1.0.pdf.

[3] FlexPlan project, “D2.3 Flexibility elements analysis preprocessor simulation tool,” [Online]. Available: https://flexplan-project.eu/wp-content/uploads/2021/06/D2.3_20210628_V1.0.pdf.

[4] P. S. Foundation, “Python,” [Online]. Available: https://www.python.org/.

[5] Oracle, “Java,” [Online]. Available: https://www.java.com/en/.

[6] S. Foundation, “Scala,” [Online]. Available: https://www.scala-lang.org/.

[7] “The Julia Programming Language,” [Online]. Available: https://julialang.org/.

[8] H. Mittelmann, "The MIPLIB2017 Benchmark Instances," [Online]. Available: http://plato.asu.edu/ftp/milp.html. [Accessed 2020 August 2020].

[9] B. Meindl and M. Templ, “Analysis of commercial and free and open source solvers for linear optimization problems,” Institut f. Statistik u. Wahrscheinlichkeitstheorie, Austria, 2012.

[10] B. Meindl and M. Templ, “Analysis of Commercial and Free and Open Source Solvers for the Cell Suppression Problem,” *Transactions on Data Privacy,* vol. 6, pp. 147-159, 2013.

[11] IBM, “CPLEX OPTIMIZER,” [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer.

[12] Gurobi optimization, [Online]. Available: https://www.gurobi.com/.

[13] Free Software Foundation, “GLPK - GNU Project,” [Online]. Available: https://www.gnu.org/software/glpk/.

[14] Zuse Institute Berlin, “SCIP,” [Online]. Available: https://www.scipopt.org/.

[15] Coin-OR, “CBC,” [Online]. Available: https://github.com/coin-or/Cbc.

[16] MIT, “lp_solve,” [Online]. Available: https://lpsolve.sourceforge.net/5.5/.

[17] IBM, “DOcplex Python Modeling API,” [Online]. Available: https://www.ibm.com/docs/en/icos/12.9.0?topic=docplex-python-modeling-api.

[18] IBM, “Python API of CPLEX,” [Online]. Available: https://www.ibm.com/docs/en/icos/20.1.0?topic=cplex-setting-up-python-api.

[19] IBM, " Concert Technology," [Online]. Available: https://www.ibm.com/support/pages/introduction-concert-technology.

[20] IBM, "Optimization Programming Language (OPL)," [Online]. Available: https://www.ibm.com/docs/en/icos/12.8.0.0?topic=opl-optimization-programming-language.

[21] AMPL Optimization, "AMPL Software," [Online]. Available: https://ampl.com/.

[22] GAMS, [Online]. Available: https://gams.be/en/.

[23] Sanida National Laboratories, [Online]. Available: http://www.pyomo.org/.

[24] Google, "Google OR-Tools," [Online]. Available: https://developers.google.com/optimization.

[25] Open-Source, "JSON (JavaScript Object Notation)," [Online]. Available: https://www.json.org/json-en.html.

[26] Wikipedia, "Comma-separated values," [Online]. Available: https://en.wikipedia.org/wiki/Comma-separated_values.

[27] File Viewer Lite, "XLS and XLSX files," [Online]. Available: https://windowsfileviewer.com/open/xls_xlsx_files.

[28] ENTSO-e, "Common Grid Model Exchange Standard (CGMES) Library," [Online]. Available: https://www.entsoe.eu/data/cim/cim-for-grid-models-exchange/.

[29] Union for the Co-ordination of Transmission of Electricity, , "UCTE-DEF," [Online]. Available: https://www.powsybl.org/pages/documentation/grid/formats/ucte-def.html#:~:text=The%20UCTE%2DDEF%20(UCTE%20Data,interconnected%20extra%20high%20voltage%20network..

[30] PowSyBl, "PSS-E," [Online]. Available: https://www.powsybl.org/pages/documentation/grid/formats/psse.html#:~:text=A%20PSS%C2%AEE%20RAW,data%20needed%20in%20power%20flow..

[31] D. S. Foundation, "Django," [Online]. Available: https://www.djangoproject.com/.

[32] E. You, "Progressive JavaScript Framework," [Online]. Available: https://vuejs.org/.

[33] Mapbox, "Mapbox: maps and locations for developers," [Online]. Available: https://www.mapbox.com/.

[34] Plotly, "Plotly Python Open Source Graphing Library Maps," [Online]. Available: https://plotly.com/python/maps/.

[35] MapLibre (open-source), "MapLibre," [Online]. Available: https://maplibre.org/.

[36] Google, "Google Maps," [Online]. Available: https://www.google.com/maps/about/#!/.

[37] OpenJS Foundation, "WebGL-powered visualization framework for large-scale datasets," [Online]. Available: https://deck.gl/.

[38] FlexPlan project, "D1.2 Probabilistic optimization of T&D systems planning with high grid flexibility and its scalability," 2021. [Online]. Available: https://flexplan-project.eu/wp-content/uploads/2022/08/D1.2_20220801_V2.0.pdf.

[39] SmartBear, "Open API Specificatins," [Online]. Available: https://swagger.io/specification/.

[40] Amazon, "AWS Cloud Computing," [Online]. Available: https://aws.amazon.com/.

[41] scikit-learn, "KMeans," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[42] FlexPlan project, "D1.1 Monte Carlo scenario generation and reduction," [Online]. Available: https://flexplan-project.eu/wp-content/uploads/2020/12/D1.1_20201210_V1.0.pdf.

[43] SmartBear, "Swagger Codegen," [Online]. Available: https://swagger.io/tools/swagger-codegen/.