Geomop 1.0.0 - reference guide

Geomop is a collection of supporting tools for the Flow123d simulator of transport processes in the fractured porous media.
Currently it consists of tools:

Layers - preparation of layered computational geometry from the GIS data
Model Editor - editor of the main Flow123d input file in YAML format
Jobs - running and management of job on distributed computational resources
Analysis - arrangement of complex computing scenarios

1 Installation

Installation packages are available at http://geomop.nti.tul.cz/packages/1.0.0.
Standard graphical instalator is available for Windows OS. To run the installation execute file geomop_1.0.0_x86.exe and go through installation wizard. Individual tools can be executed by their shortcuts or by batch files that are located, assuming default install location, in the folder:
c:\Program Files (x86)\GeoMop\bin\

Known issue in Win 10:
Terminal windows are displayed within the installation, in case of accidental click on the terminal, the installation process is suspended, you must press the ESC key to continue.

2 Layers

The Layers tool provides a 2D vector editor of individual layers, their composition with curved interfaces and a tool for composition of the layers into a 3D geometry and meshing via. external tool (GMSH). A GIS data (shapefiles) can be displayed at background in order to make the created model related to the real world features.

2.1 Overview

When creating a new file or starting the application you are asked for determine extent of your initial view. This can be given either by selecting a GIS shapefile for the background, or by providing the extent manually. Alternatively one can open an existing layer geometry file. The layout of the application is depicted on Figure 1. Key components are:
Topology editor - specialized 2d vector editor of the horizontal division of a single layer. Intersection of surface domains used in a single block of layers defines a valid domain of the block in the ground plan. The valid domain is depicted as a gray polygon. An extent of the selected surface is displayed as an orange rectangle if the surface panel is active.

Layer panel - scheme of the layers of the geometry. Allows work with whole layers and setting surfaces on interfaces between the layers.
Region panel - assignment of computational regions to the polygons and line segments in the topology editor window. Editor of region properties.
Surface panel - import of terrain grid files and construction of the surfaces as B-spline approximations.
Background panel - allows to set GIS layers to display for one or more *Shape files* opened through `File -> Add shapefile ...`

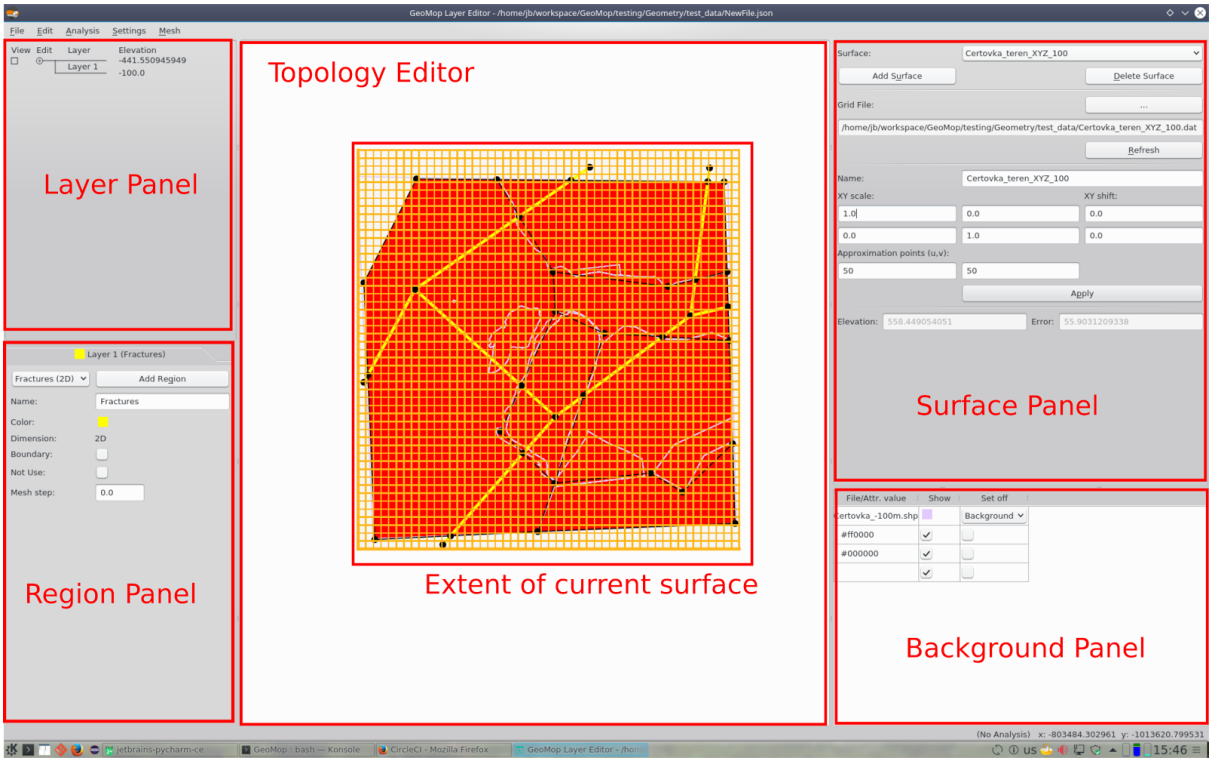

Figure 1: Layers editor - application layout.

2.3 Layered geometry

Whole 3d geometry consists of individual geological layers. Layers can be of two types: *Fracture layer* - a surface layer representing a horizontal fracture and *Stratum layer* - a volume layer. The stratum layers are delimited by top and bottom *interfaces* while the fracture layers lay on a single interface. An interface is a horizontal surface given either by constant depth or by a B-spline surface (defined i the Surface panel) shifted and scaled in the vertical (Z-direction).
Horizontal structure of layers is called *Topology. Layer block is a* continuous block of layers sharing a same topology. Topology consists of nodes, segments and polygons. For stratum layers these objects corresponds to wells, fractures, and bulk regions in the 3d geometry, respectively. For fracture layers the segments and polygons corresponds to wells and fractures in the 3d geometry, respectively. Wells, fractures and bulks across the layers can

be grouped into regions with a same meaning as in the Flow123d input file. All horizontal and vertical interfaces given are captured by the resulting mesh.

2.4 Layers panel

Layers panel displays vertical structure of the model as a list of layers and separating interfaces in the order form the top to the bottom. Layers are organized into blocks of layers that share a common topology. Interfaces between stratum layers are depicted by horizontal lines annotated with the depth of the interface in the center of the valid domain. Click on the layer opens a menu allowing to: split the layer, rename, or remove the layer. Click on the depth allows to: set the surface of the interface, set or remove a fracture layer on the interface. For the first and last interface you can prepend and append the new layer, respectively.

Surface dialog

The surface of an interface can be set either to plane with constant elevation or to a B-spline surface created in the Surface panel. A shift and scale of the surface in the Z-direction can be specified which is particular to the single interface.

2.5 Surface panel

The surface panel located on the right top is used to import terrain grid files and construct their approximation by B-spline surfaces.
List of existing B-spline approximations is in the top part. Following two sections displays data related to the selected surface.
The second section serves to load or reload a grid file. The grid file has to be a text file with three floating point numbers per line representing XYZ coordinates of individual points. In order to get good approximation the XY projection of the points should evenly fill an rectangle. This rectangle is determined automatically as a minimum area bounding box. The bounding rectangle determines UV to XY mapping of the surface as the XY coordinates of the B-spline poles forms a regular grid. Only the Z coordinates of the poles are determined in terms of the regularized least-square approximation of the Z-coordinates of the grid.
The third and last section is configuration of the approximation: surface name, XY linear transformation, and quality of the approximation determined by the number of B-spline subintervals in U and V direction. The elevation of the approximation in the center and the maximal error of current approximation is reported at the bottom.

2.6 Region panel

Region panel have a single tab for every layer. On a tab you can select actual region from the list of all regions, create a new region and edit the actual region. Region properties are:
Name - name of the region (leading dot for the boundary regions is added during meshing).
Color - used to highlight objects of the selected layer belonging to the region
Dimension - topological dimension, set when region is created.

Boundary - indicator for boundary regions.
Not-used - indicator for regions excluded from the geometry. Three predefined None regions for every dimension are always “not-used”. These are default regions for new topology objects if no actual region is selected.
Mesh step - maximal size of mesh elements in the region. Actual size of elements may be smaller in order to satisfy conditions from neighbouring regions or due to complexity of the geometry.

2.7 Background panel

One or more ESR shapefiles (*.shp) can be loaded via `File -> Add shapefile …`
Attributes can be used to select GIS objects to display (or highlight). Every layers have individual color.

2.8 Topology editor

Central view of the application provides a vector editor for horizontal structure of the geological layers. The GIS objects selected in the Background panel are displayed at the background. The valid domain of the current block of layers is displayed as a light gary polygon. A single layer topology consists of points, line segments and polygons. The points and segments are draw by the user, while the polygons are detected automatically.

Editing operations (left mouse button)
left click - start drawing a polyline, Esc - drop the moving segment and stop the line drawing
ctrl + left click - create single node, stop the line drawing
left click + drag - moving a node or a segment

Selection and region operations (right mouse button)
right click - select an object (node, segment, polygon)
shift + right click - add/remove object to/from selection
right click + drag - move the canvas
wheel - zoom in / out

ctrl + right click - select objects of the actual region on actual tab
ctrl + alt + right click - set regions selected at tabs of individual layers to the pointed object if the dimension match

Region can be assigned to all selected object with matching dimension by choosing that region in the current tab of the region panel.

Selected objects can be deleted through `Edit -> Delete` or by the Del key. The first delete action just removes all assigned regions, the second delete removes the actual objects.

Undo and Redo operations are accessible through `Edit -> Undo/Redo` or by Ctrl-Z/Ctrl-Y shortcuts, respectively.

2.9 Meshing

Meshing process can be started through **`Mesh -> Make mesh.`** Simple dialog allows to prescribe a default mesh step to the region with the default setting and start the meshing process. This consists of following steps:

1. Save the file with layer geometry (e.g. hills.json).
2. Create the 3D geometry and save it in the BREP format (e.g. hills.brep).
3. Create the script for the GMSH mesh generator with mesh step sizes (e.g. hills.tmp.geo)
4. Call GMSH. Temporary mesh is produced. (e.g. hills.tmp.msh)
5. Create the final mesh by removing unwanted elements and setting region names (e.g. hills.msh).

3 Model Editor

3.1 Overview

Model Editor is a specialized editor of the YAML configuration files of the Flow123d simulator.
Main features of the editor:

- validation of the configuration files (see: Messages tab)
- context-sensitive help widget describing the data types (see: Documentation tab)
- visual representation of the whole data structure (see: Tree window)
- autocompletion of keys and values while typing
- import of old-format (CON) configuration files and conversion between format of different versions of Flow123d
- data structure transformations of the edited files
- usual text editor functions

Structure of the input YAML file for the simulator Flow123d is described in a machine readable form that allows basic validation of the input, but contains the documentation of the input structure as well. Using the format description the Model Editor adopts to various versions of the simulator and is also able to do semi-automatic conversion of various versions.

The application consist of three windows. On the left there is the *Tree window* displaying the content of the file in the tree structure. On the right-top there is the *Text editor window* and finally on the right-bottom there is either the *Notification tab* reporting errors in the input file, or the *Documentation tab* with context dependent documentation of the file format. Basic layout is on Figure 2.

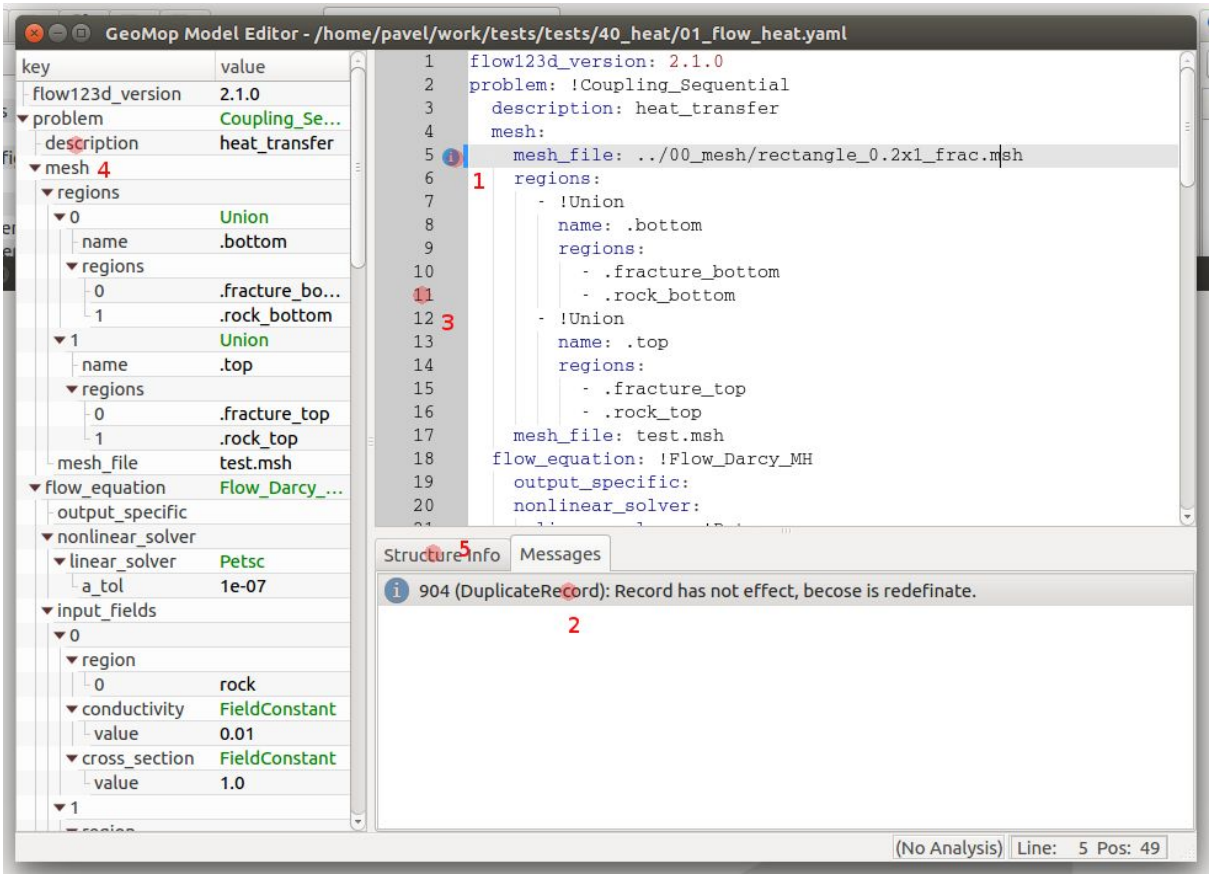

Figure 2: Layout of the application, interplay of the Tree, Editor and Messages.

3.2 Text editor window

Displays the input YAML file and allows its free editing. The editor offers a context-sensitive completion feature that can be activated by pressing Ctrl+Space. Automatic completion can be turned on through `Settings -> Options ...`
Click to the line number on the left part of the editor window (see Figure 2/3) finds and highlights corresponding place in the tree window.
Content of the window is continuously validated against format of the Flow123d given by the 'flow123d_version' key. Errors are marked next to the line numbers (see Figure 2/1). Click to the error mark opens the Messages tab and highlights corresponding error message.

3.3 Tree window

The tree window displays the structure of the data in the input file as it is seen by Flow123d after automatic conversions. Three conversions are applied: conversion from single Record key to the whole record, conversion a value to the single element list, and transposition, i.e. conversion of records of lists to a list of records. See Flow123d documentation for the details. The tree window does not allow any edit operations.
Click to an element of the tree (see Figure 2/4) highlights corresponding part of the input file.

3.4 Messages tab

The messages tab reports errors (see Figure 2/2) in the YAML input when compared to the Flow123d format. Four types of messages are reported:

- **fatal error** - error that prevents further file parsing, e.g. wrong indentation
- **error** - local error in validation, e.g. missing obligatory key, unknown value of a selection
- **warning** - possible error, e.g. unknown key which is ignored, but may be a misspell
- **info** - warning of law importance, e.g. duplicate key

Click on the message highlights corresponding place in the Text editor window.

3.4 Documentation tab (Structure Info)

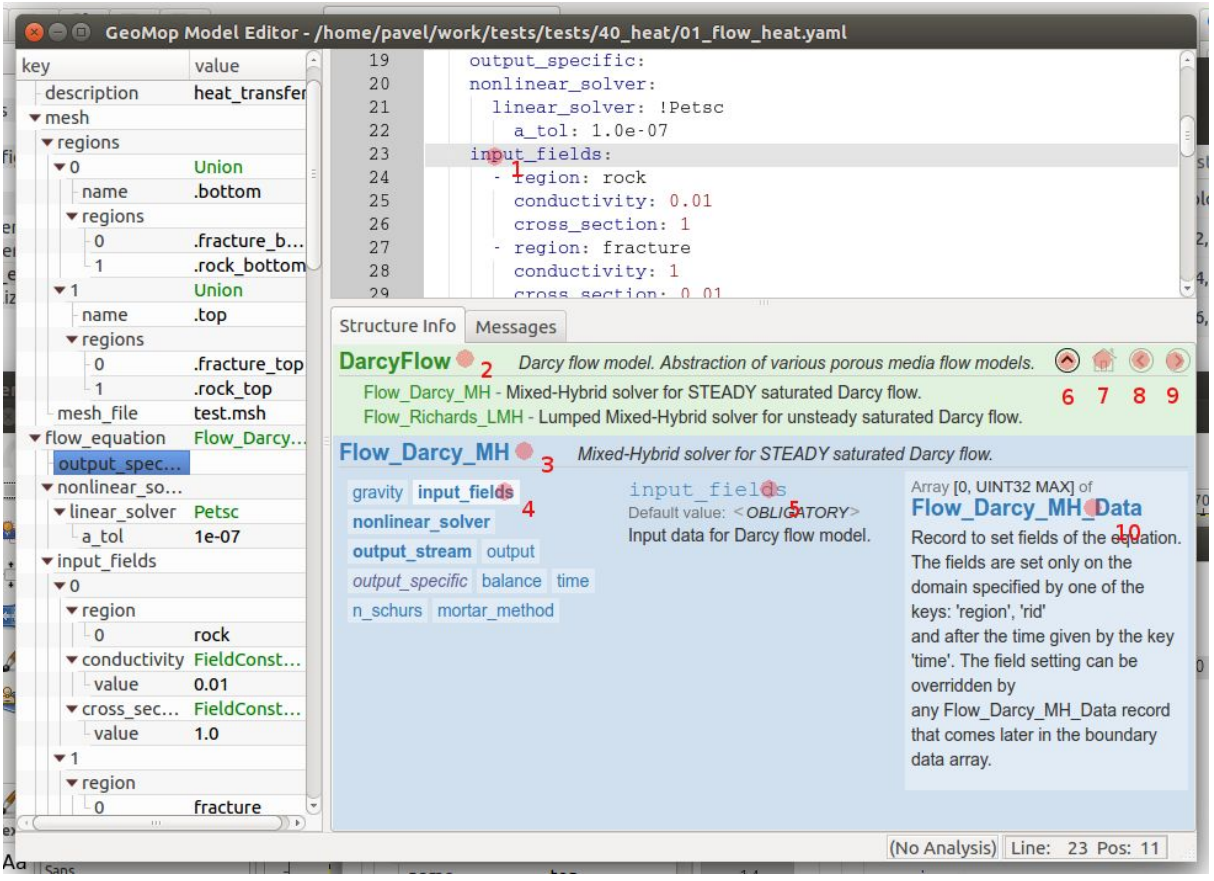

Figure 3: Structure of the Documentation (Structure Info) tab with and its relation to the Text editor window.

The documentation tab (entitled ‘Structure Info’ in Figure 3) contains a context dependent documentation of the Flow123d format for current position in the Text editor window.
For example for the selected line 23 (see Figure 3/1) the documentation tab displays documentation of the record ‘Flow_Darcy_MH’ (see Figure 3/3) containing the active key ‘input_fields’. Since the record ‘Flow_Darcy_MH’ is an instance of the abstract ‘DarcyFlow’ the documentation of the abstract is displayed as well containing links to all existing implementation records (see Figure 3/2).
For actual record, the list of its keys is located on the left (see Figure 3/4) with actual key highlighted by the white background. Different types of keys are distinguished by the font:

- **bold** for obligatory keys

- *italic* for optional keys
- normal for keys with a default value

Text description of the key type is also shown in tooltips while hovering over the key.

The documentation of the actual key is in the center (see Figure 3/5) and documentation of the type of the key value is on the right (see Figure 3/10).
Different types are distinguished by colors: green for abstract, blue for record and violet for scalar types (Integer, Double, String, Selection). Arrays of a type use the same color as the type of their elements.

Navigation over the whole documentation is possible through:

- list of implementations of an Abstract (see Figure 3/2)
- keys in the left part (see Figure 3/4)
- record key types on the right (see Figure 3/10)
- up icon (see Figure 3/6) - to display parent record of the actual record
- home icon (see Figure 3/7) - to navigate to the documentation of the actual element in the Text editor window
- previous icon (see Figure 3/8) - go back in the history
- next icon (see Figure 3/9) - go forward in the history

4 Analysis

The analysis module provides Python classes to compose and execute complex modeling calculations from simpler tools called *actions.* In order to pass the data between actions we use *data type* classes to describe the structure of the data. Conversion of the data structures between actions is performed by *convertors.* The complete pipeline of the actions interconnected by convertors forms the *analysis.* The analysis can be executed as a batch job. The validation of the pipeline is performed before the job is queued for the bath processing. When executed via a *multi job* (see Job Panel documentation), the pipeline can execute actions in parallel as a child batch jobs. Currently the analysis can be composed only by a Python script, however the design assumes creation of a GUI for this task.

4.1 Data types

Data type classes in the analysis are used for a type checking and data transfer between the actions. The basic data types are: Bool, Int, Float, String, Enum. The composite data types include:

Struct is like the Python's dictionary where the keys are string-only. A single instance of Struct have fixed set of keys and types of corresponding values.

```
# type definition
a = Struct(x=Int(), y=String())
```

Tuple is an array of a fixed length with values of the prescribed (but possibly different) types. It is like the Struct type, where natural numbers (from zero) are used as keys.

```
# type definition
a = Tuple(Int(), String())
```

Ensamble is a set of values with given common type and without any prescribed order. This implies that individual elements of an ensamble can be processed in parallel.

```
# type definition
a = Ensemble(Int())
```

Sequence is like the Ensamble but with defined order of elements.

```
# type definition
a = Sequence(Int())
```

4.2 Converters

It serves only to merge, select and reorganize data, does not make any calculations. It is primarily used in action connectors, but one converter can also be used inside another converter, especially for Ensemble and Sequence operations.

Reformatting. Basic tool of conversion is creation of the new structure through extraction from the input data structures.

Example:

```
Converter(Struct(a=Input(0).c, b=Input(1)[2]))
```

Make a structure from two inputs setting key ‘a’ to the key ‘c’ of the struct on input 0 and setting key ‘b’ to the second element of the tuple on the input 1.

Each. Is used to apply the converter to each item of Ensemble. The converter, which is the ‘each’ parameter, must have just one Input(0) input.

Example:

```
# from Ensamble of Structs, make Ensamble of Tuples
Converter(Input(0).each(Adapter(Tuple(Input(0).a, Input(0).b))))
```

Select. The ‘select’ operation extract elements of Ensamble or Sequence satisfying a given *Predicate*.

Example:

```
# Extract elements smaller than 3.
Converter(Input(0).select(Predicate(Input(0) < 3)))
```

Sort. Sort Ensemble or Sequence according to a given value and producing a Sequence. Sorting value is given by a converter that returns a scalar (or, more generally, a comparable type) which is called *Selector.*

Example:

```
# Sort an Ensamble of structs by the value of the key ‘b’
Converter(Input(0).sort(Selector(Input(0).b)))
```

4.3 Actions

One action is a specific computational or graphic operation. The specific details of the performed operations can be affected by the action configuration. Each action has a single Input Slot and single Output Slot. The input and output data type is fixed by the action type and its configuration. The configuration of an action can affect the data type of the input and output slot. The configuration can not be result of another action, its specification is part of the workflow definition.

4.3.1 Generator actions

VariableGenerator The data passed in the configuration, provides on the output.
Example:

```
var = Struct(x1=Float(1.0), x2=Float(2.0))
gen = VariableGenerator(Variable=var)
```

RangeGenerator Generator for generation parallel lists.
Example:

```
items = [
    {'name':'a', 'value':1, 'step':0.1, 'n_plus':1,
     'n_minus':1,'exponential':False},
    {'name':'b', 'value':10, 'step':1, 'n_plus':2,
     'n_minus':3,'exponential':False}
]
gen = RangeGenerator(Items=items, AllCases=False)
```

4.3.2 Parametrized actions

The common denominator of these actions is that they call non-trivial external programs and assume the definition of the data of these programs in specific contexts, the input of actions can only affect the declared parameters in the input data (files) of each program.

Flow123dAction
Action calls the Flow123d simulator. Configuration includes a YAML file and a computing mesh.
Example:

```
gen = VariableGenerator(Variable=Struct(par=Float(1.2)))
flow = Flow123dAction(Inputs=[gen], YAMLFile="test.yaml")
```

FunctionAction
This action compute some mathematical expressions on input data and return results on output.
Example:

```
var = Struct(x1=Float(1.0), x2=Float(2.0), x3=Float(math.pi/2))
gen = VariableGenerator(Variable=var)
fun = FunctionAction(
    Inputs=[gen],
    Params=["x1", "x2", "x3"],
    Expressions=["y1 = 2 * x1 + 3 * x2", "y2 = sin(x3)"])
```

4.3.3 Wrapper actions

ForEach
The action configuration includes a workflow with IN input type and OUT output type. The workflow is executed on all input elements of type Ensemble(IN), resulting in output of type Ensemble(OUT).
Example:

```
var = Ensemble(Float(), Float(1.0), Float(2.0), Float(3.0))
gen = VariableGenerator(Variable=var)
w = Workflow()
f = FunctionAction(
    Inputs=[w.input()],
    Params=["x"],
    Expressions=["y = 2 * x"]
)
w.set_config(
    OutputAction=f,
    InputAction=f
)
for_each = ForEach(
    Inputs=[gen],
    WrappedAction=w
```

```
      )
```

Calibration

Calibrating a generic model for given data, ie identifying model parameters so that the result of the model was in some sense the closest measured data.

Example:

```
gen = VariableGenerator(
    Variable=(
        Struct(
            observations=Struct(
                y1=Float(1.0),
                y2=Float(5.0)
            )
        )
    )
)
w = Workflow()
f = FunctionAction(
    Inputs=[
        w.input()
    ],
    Params=["x1", "x2"],
    Expressions=["y1 = 2 * x1 + 2", "y2 = 2 * x2 + 3"]
)
w.set_config(
    OutputAction=f,
    InputAction=f
)
cal = Calibration(
    Inputs=[
        gen
    ],
    WrappedAction=w,
    Parameters=[
        CalibrationParameter(
            name="x1",
            group="test",
            bounds=(-1e+10, 1e+10),
            init_value=1.0
        ),
        CalibrationParameter(
            name="x2",
            group="test",
            bounds=(-1e+10, 1e+10),
            init_value=1.0
```

```
        )
    ],
    Observations=[
        CalibrationObservation(
            name="y1",
            group="tunnel",
            weight=1.0
        ),
        CalibrationObservation(
            name="y2",
            group="tunnel",
            weight=1.0
        )
    ],
    AlgorithmParameters=[
        CalibrationAlgorithmParameter(
            group="test",
            diff_inc_rel=0.01,
            diff_inc_abs=0.0
        )
    ],
    TerminationCriteria=CalibrationTerminationCriteria(
        n_max_steps=100
    ),
    MinimizationMethod="SLSQP",
    BoundsType=CalibrationBoundsType.hard
)
```

4.3.4 Special actions

Workflow

Encapsulates the entire workflow into a new user-defined action.

Example:

```
w = Workflow()
f = FunctionAction(
    Inputs=[w.input()],
    Params=["x1", "x2"],
    Expressions=["y1 = 2 * x1 + 2", "y2 = 2 * x2 + 3"]
)
w.set_config(
    OutputAction=f,
    InputAction=f
)
```

Pipeline
Pipeline groups all of other actions. It has not any input action and least one output action. Output action define action contained in pipeline. Output action would be actions, theirs results will be downloaded.
Example:

```
var = Struct(x1=Float(1.0), x2=Float(2.0))
gen = VariableGenerator(Variable=var)
print_action = PrintDTTAction(
    Inputs=[gen],
    OutputFile="output.txt")
pipeline = Pipeline(
    ResultActions=[print_action]
)
```

5 Job Panel

The Job Panel application simplifies execution of calculations on remote computing resources. It allows to setup and organise SSH connection to several clusters, configure PBS settings, execute the jobs, monitor the status of executed jobs and it takes care of data upload and download. Directly executed jobs are called *multijobs* since they execute an analysis, which in turn can call other jobs to perform individual actions.

Layout of the user interface is depicted on Figure 4. List of all queued multijobs are at the top part, the bottom part displays informations about current multijob in three tabs: Jobs - queued child jobs, Results - produced files to download, Logs - log of messages from processing.

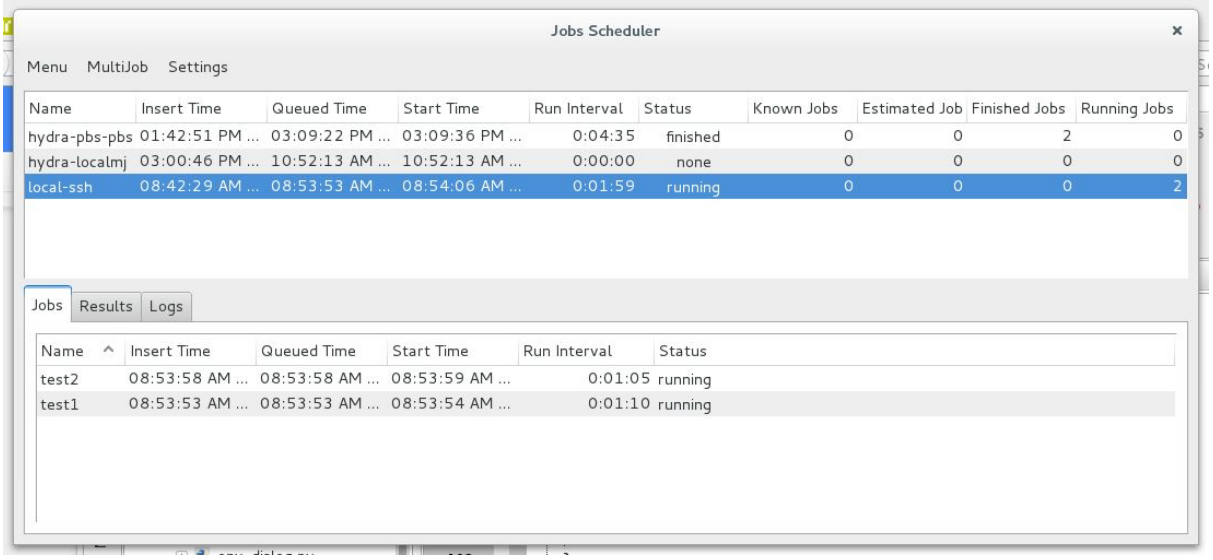

Figure 4: Layout of the Job Panel application

New multijob can be created through the Multijob menu. The multijob selected in the main list can be stopped or deleted there as well. New multijob is formed by an analysis and a (computational) *resource.* Resource is given by *SSH connection* and *PBS setting* for the multijob and for child jobs. SSH connection depends on the *environment* setting.

5.1 Configuration of resources

Environments panel (see Figure 5, left) is used to configure the Python interpreter and path to Flow123d simulator with possible fixed set of arguments. This forms a runtime environment for the multijob processing.

SSH hosts panel (see Figure 5, right) contains configuration of SSH connection to various remote machines. A single SSH connection consists of the address of IP of the remote host, credentials, type of PBS system and a selected environment. Actual SSH connection can be tested using the 'Test connection' button.

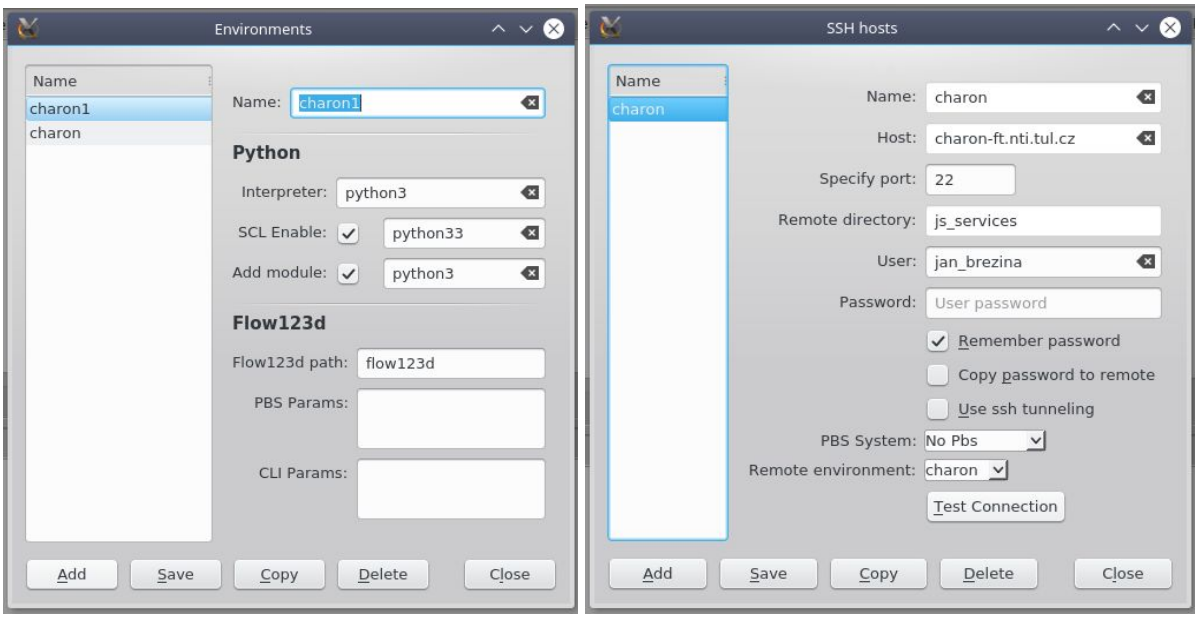

Figure 5: Environment configuration (left), SSH host configuration (right)

PBS options panel collects predefined set of PBS options.

Resources panel allows configuration of a resource be prescribing the pair SSH connection and PBS options for both the multijob execution as well as for the execution of the child Jobs.

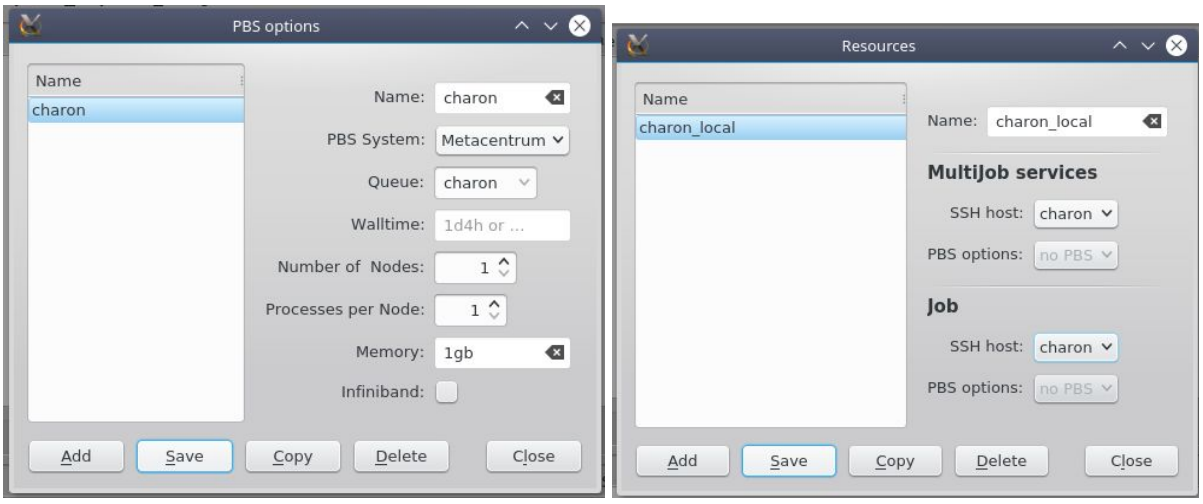

Figure 6: PBS configuration (left), Resource configuration (right)