High-Level Applications for the SARAF Accelerator

Between EPICS and Python

Antoine Choquet - CEA Paris Saclay

Presentation outline

1. Definition and requirements
2. Graphical User Interfaces (GUIs) integration
3. Architecture and technical implementation
4. Status

1. Definition and requirements

Why called: high-level?

These **applications** :

- do not interact directly with hardware
- rely on several other local control systems

Their **purposes** :

- acquire and compute data automatically
- help the operator configuring the accelerator
- reduce repetitive manual procedures

Requirements

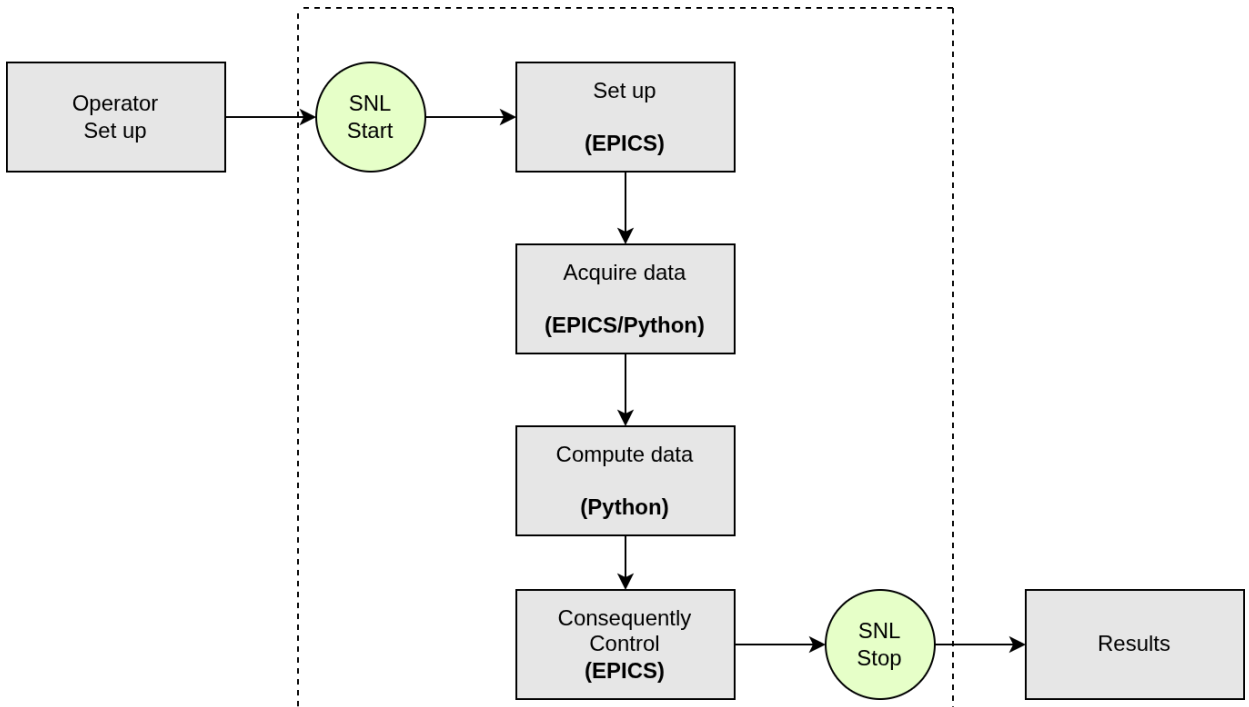
Accelerator experts from CEA :

- define requirements :
  - algorithms
  - general information and variables accessible from the **GUI**
  - expected results
  - involved equipments
  - ...
- write **Python analysis scripts**
- validate the applications with SNRC operators and EPICS developers

Automated process : EPICS State Notation Language (SNL)

why SNL ?

- just a few steps
- simple code



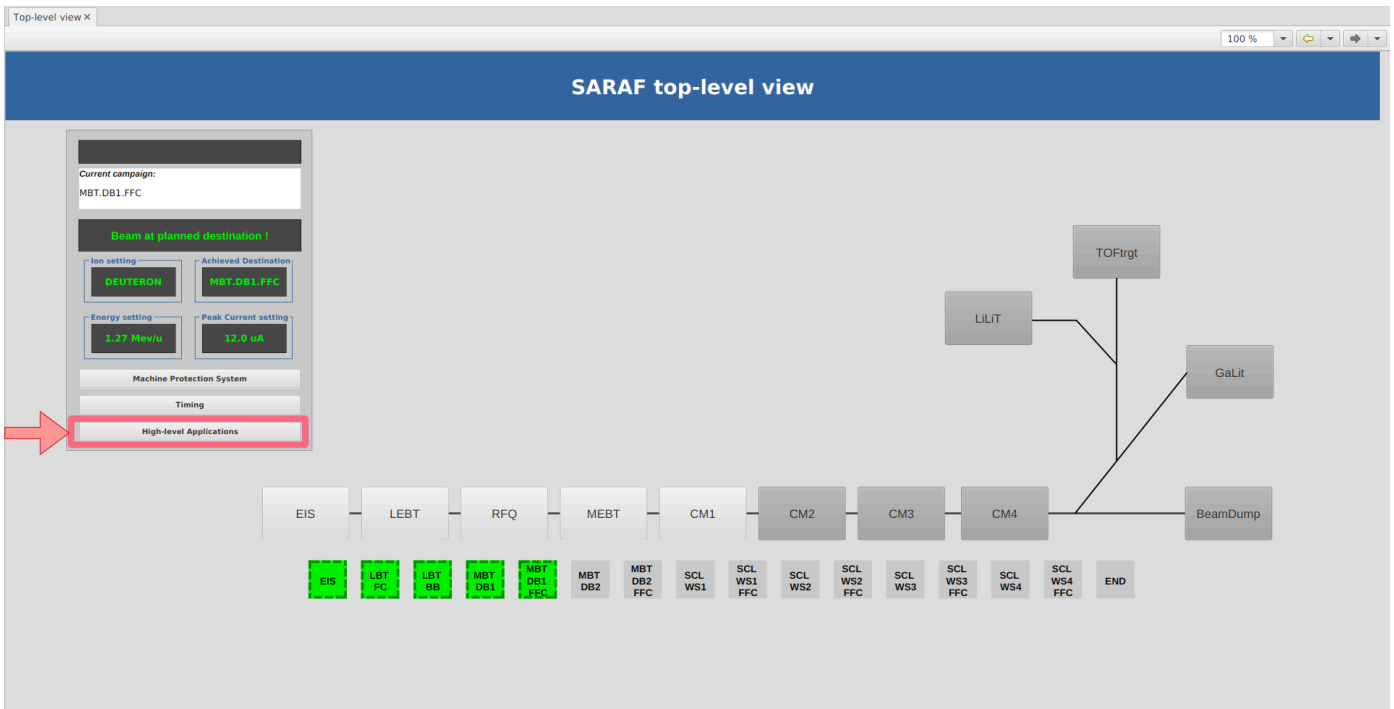
List & synthetic description

Having the same design:

- **Time of Flight (ToF)** : measuring the beam energy,
- **CavityPhasing** : calibrating the cavities,
- **BpmCalib** : calibrating the Beam Position Monitor (BPM),
- **BeamAligner** : aligning the beam,
- **EmitScan** : calculating longitudinal and transversal emittance.

2. GUIs integration

Access from top-level view

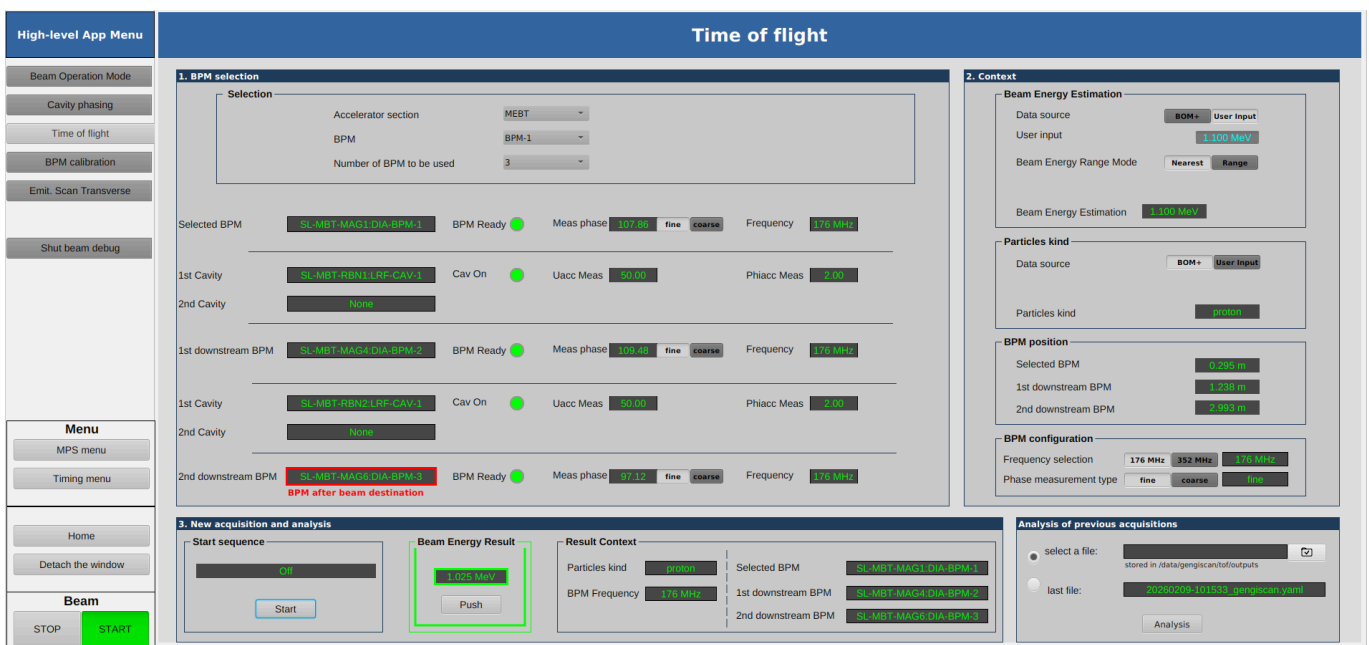


The **top-level view** is always available from the Phoebus **HOME** button

High-level applications menu - Time of Flight

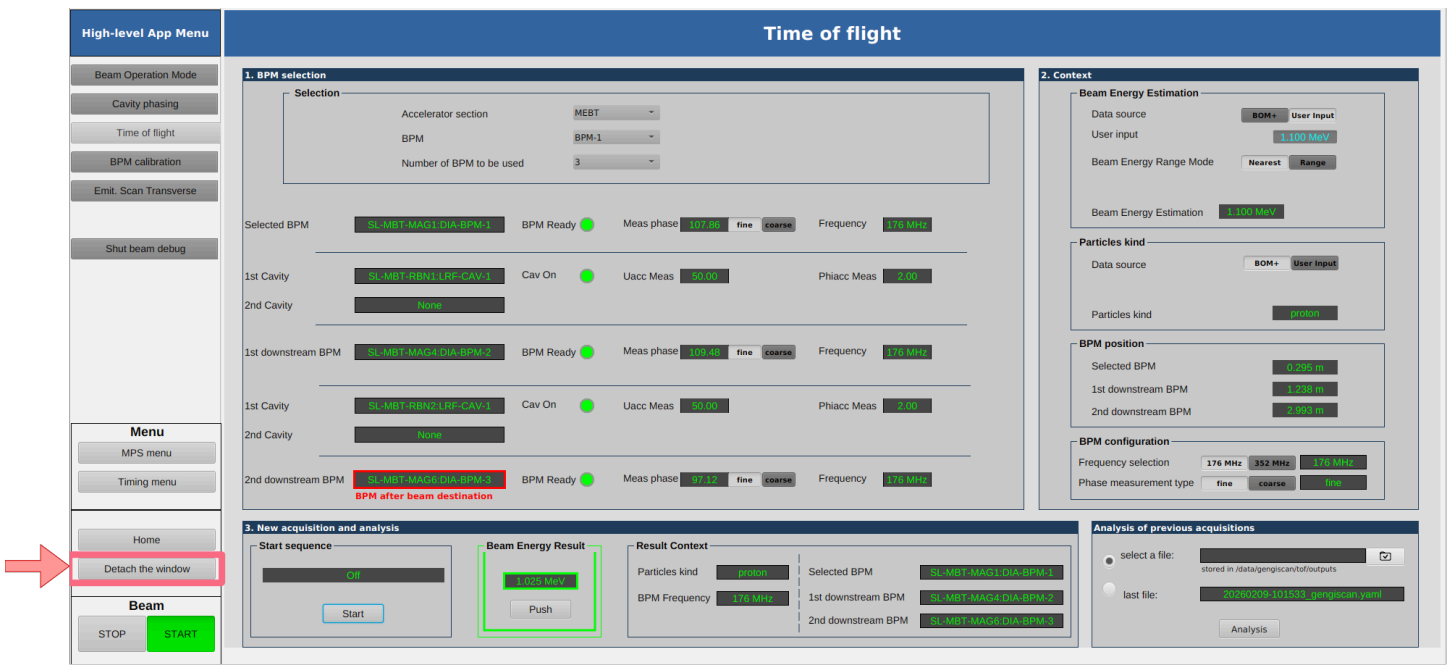
Standard menu (navigation panel on the left)

Easy integration using the **Phoebus widget** : Navigation tabs



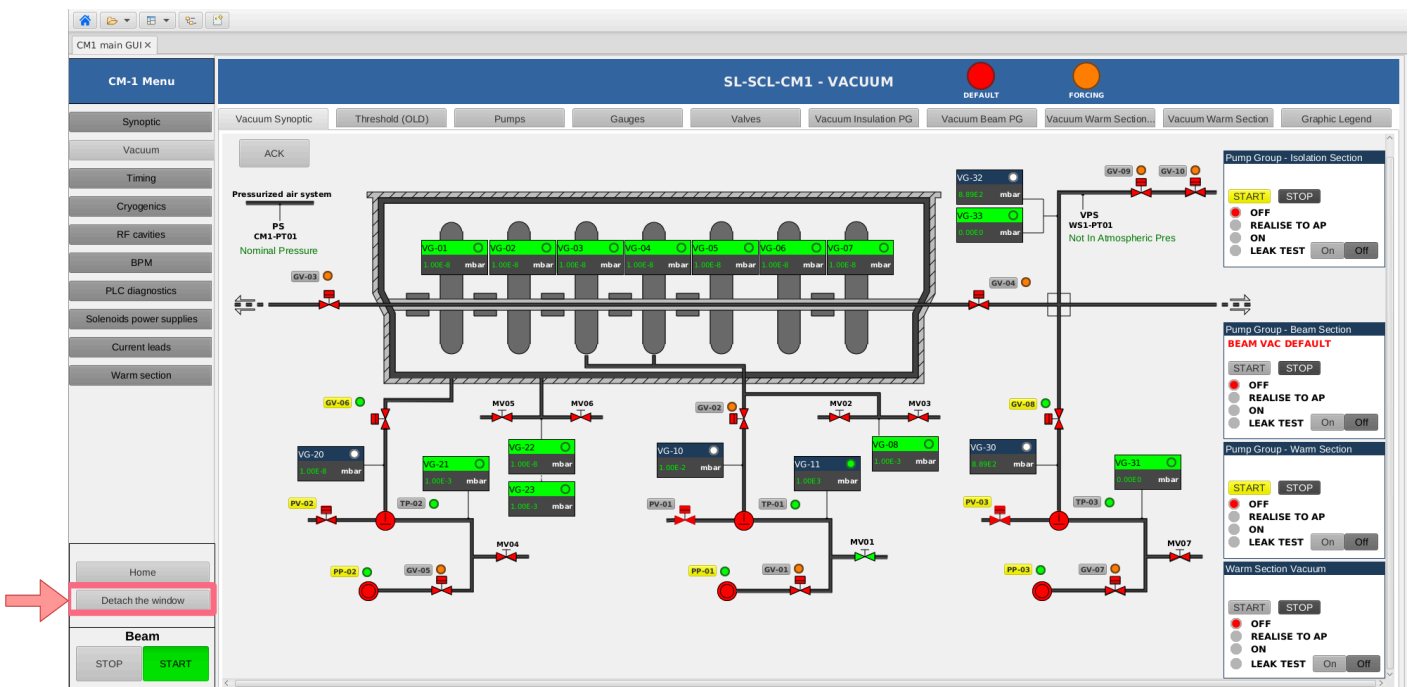
Operate simultaneously on different systems

Meanwhile using the high-level applications, operators might need to check other systems → **Detach the window**



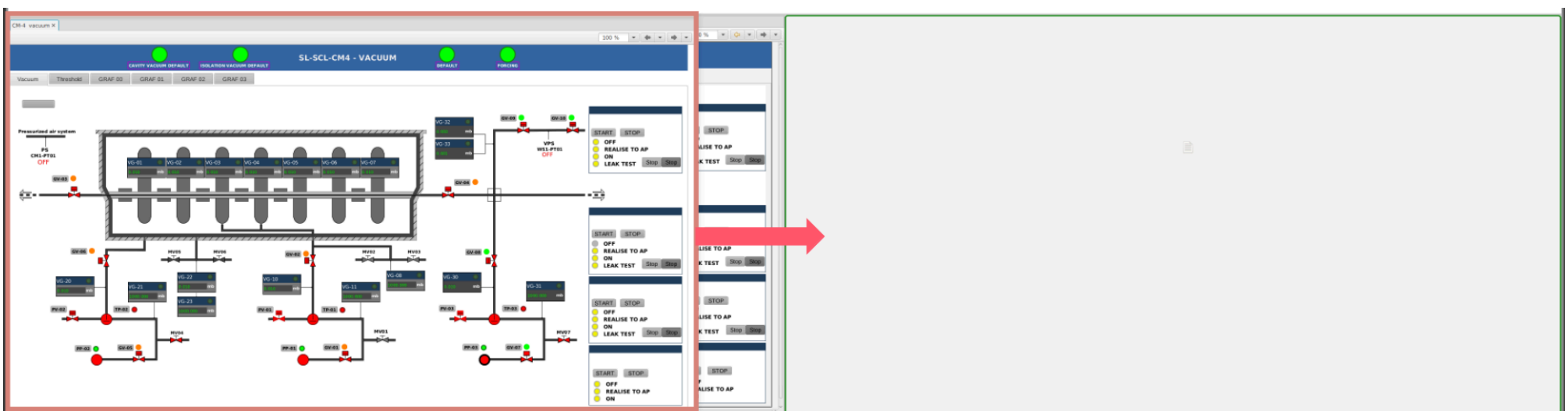
Operate simultaneously on different systems

example



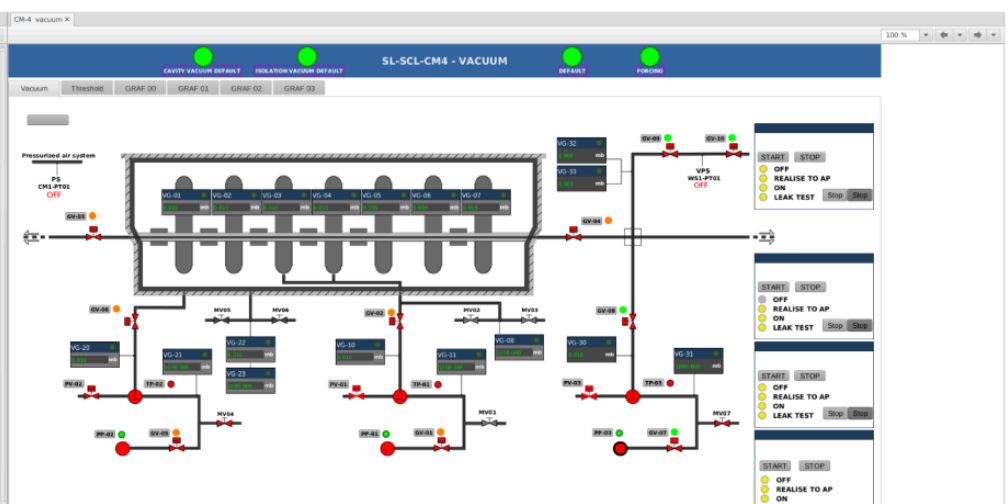
Operate simultaneously on different systems

example



Operate simultaneously on different systems

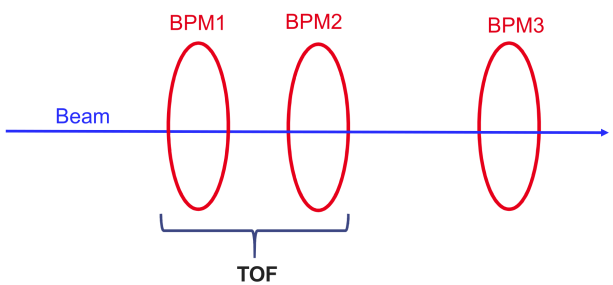
example



3. Architecture and technical implementation

Time of Flight (ToF) - example

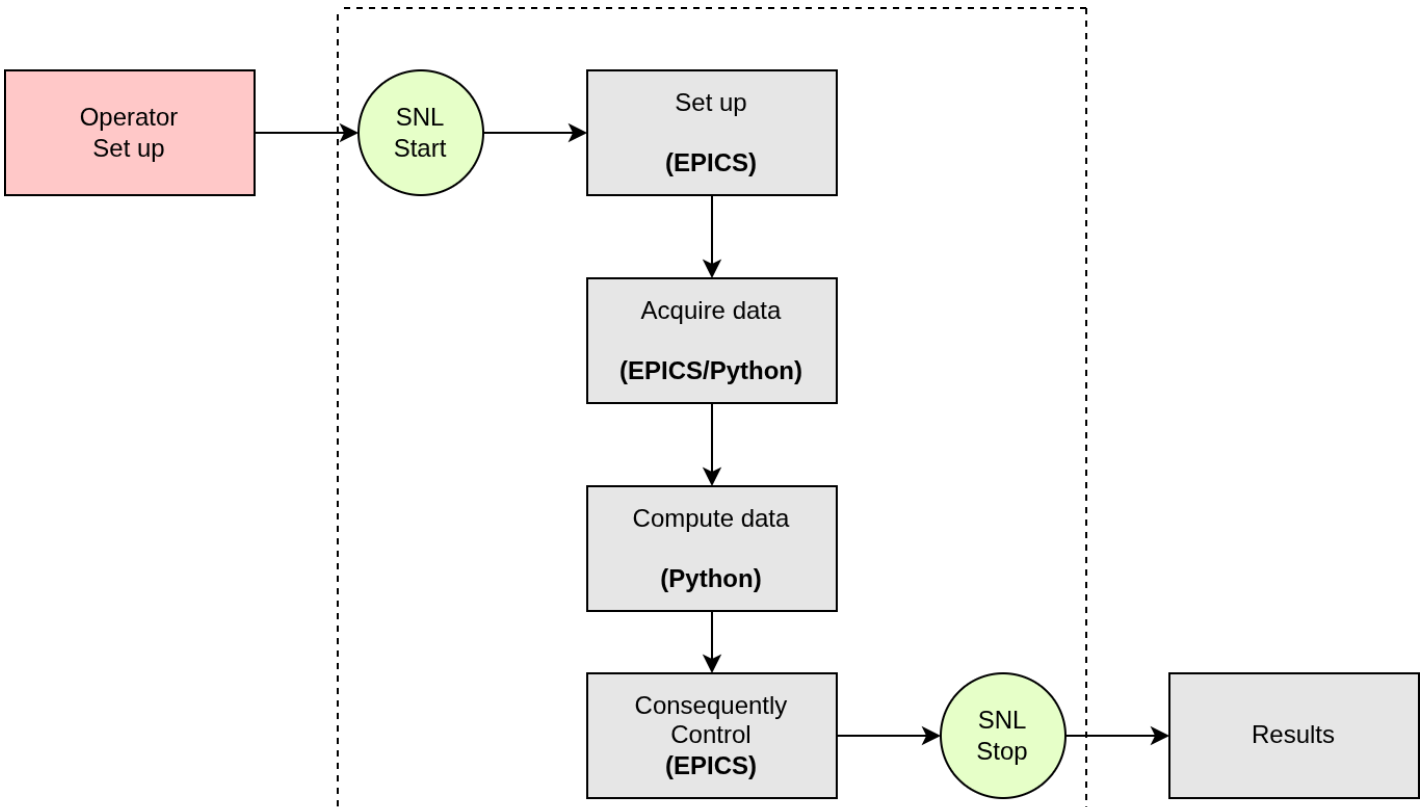
Application for measuring the beam energy anywhere in the SARAF LINAC



Synthetic requirements :

- recording beam phase in 2 or 3 BPMs – ***Analysis*** –> calculates beam energy
- if 2 BPMs used, a first guess is required for discrimination
- the beam should not be accelerated between BPMs during ToF process

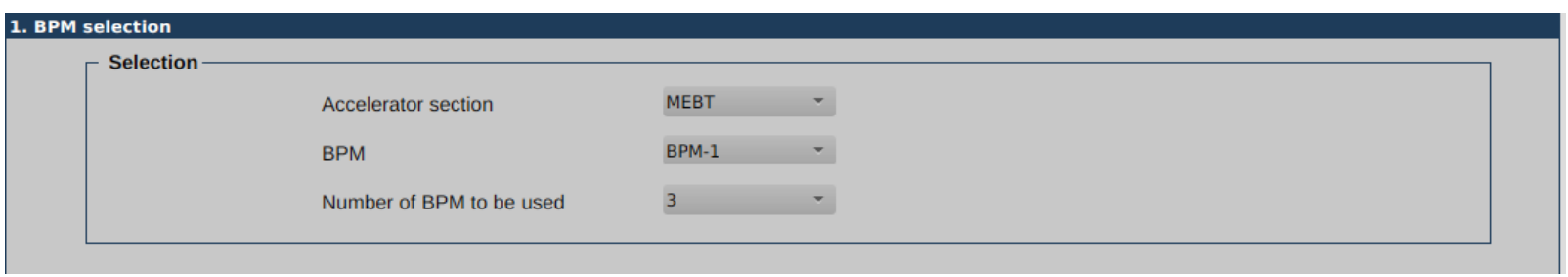
Operator set up



Operator set up - ToF - select BPMs

Operator selects :

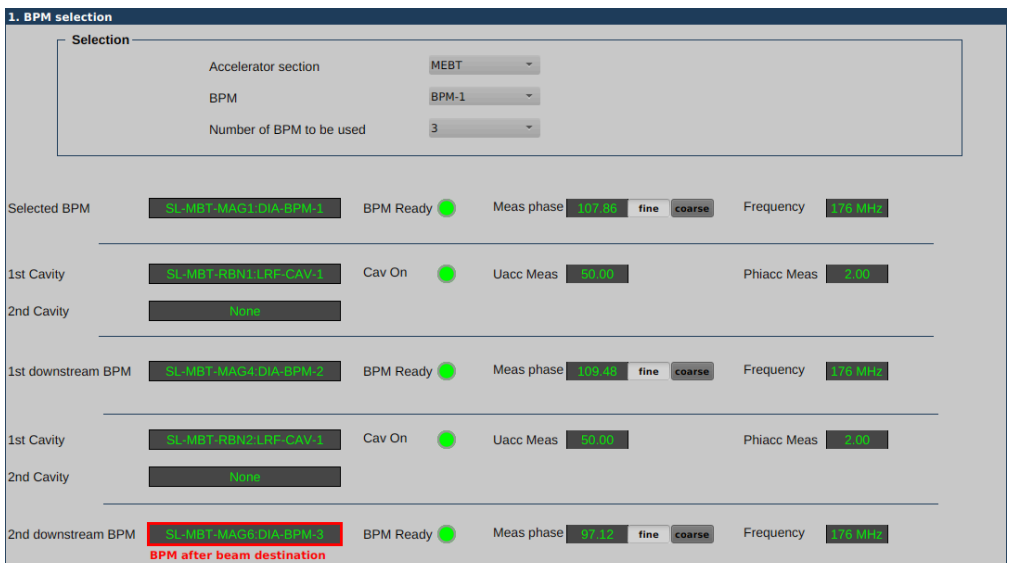
- the first BPM
- the number of BPMs (2 or 3) to use in the process



24 BPMs along the SARAF accelerator

Operator set up - ToF - select BPMs

Once first BPM selected, it is deduced the involved BPMs and cavities :

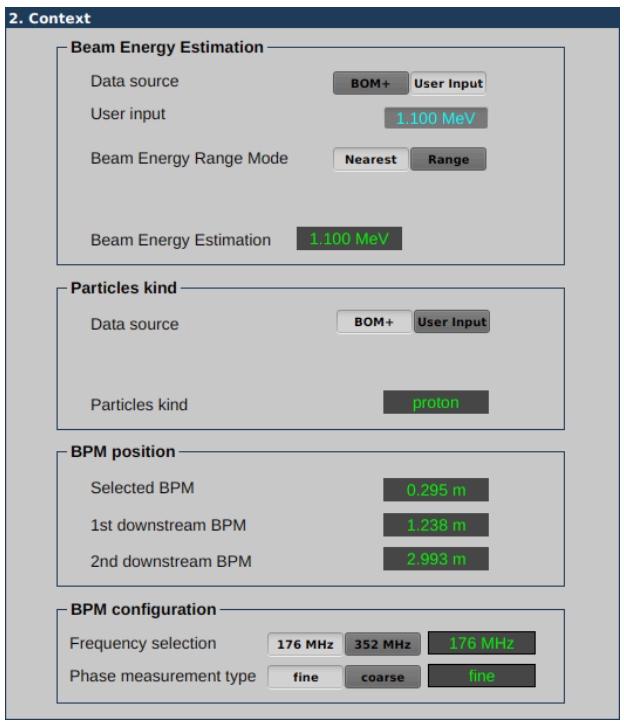


The whole logic is based on `scalcout` records.

Operator set up - ToF - context data

Operator enters :

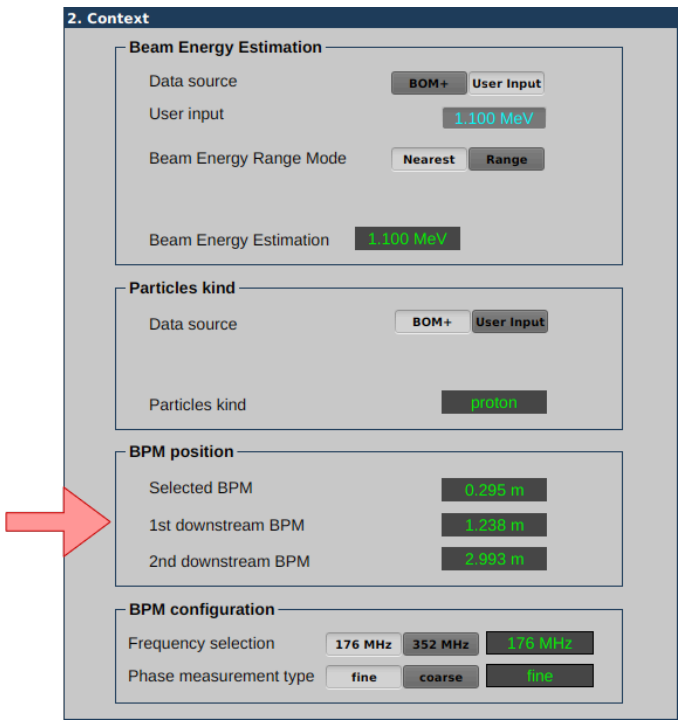
- first guess beam energy
  - whether providing a range or selecting “nearest” mode
- particle kind
- BPMs frequency
- phase measurement type (coarse or fine)



Operator set up - ToF - context data

The BPMs position are defined in a yaml file :

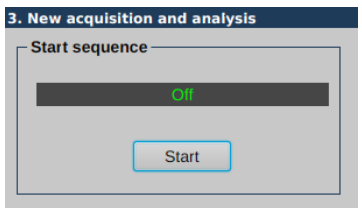
```yaml
...
elements:
  SL-MBT-MAG1:DIA-BPM-1:
    length: 0
    pos: 0.29460000000000003
  SL-MBT-MAG4:DIA-BPM-2:
    length: 0
    pos: 1.2384000000000002
  SL-MBT-MAG6:DIA-BPM-3:
    length: 0
    pos: 2.9933
...
```



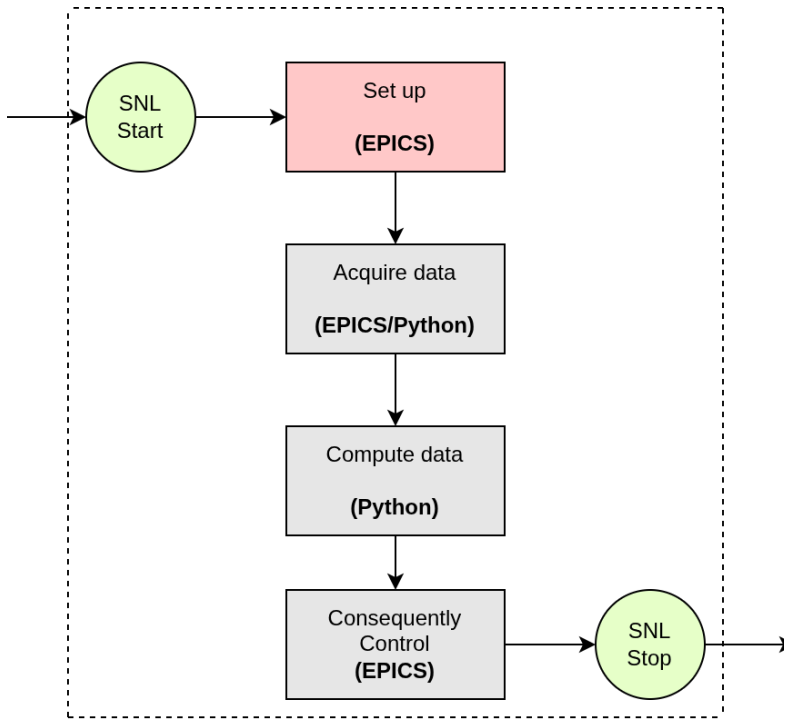
when operator selects the first BPM :

- a **Python** script reads the yaml file and writes values in PVs

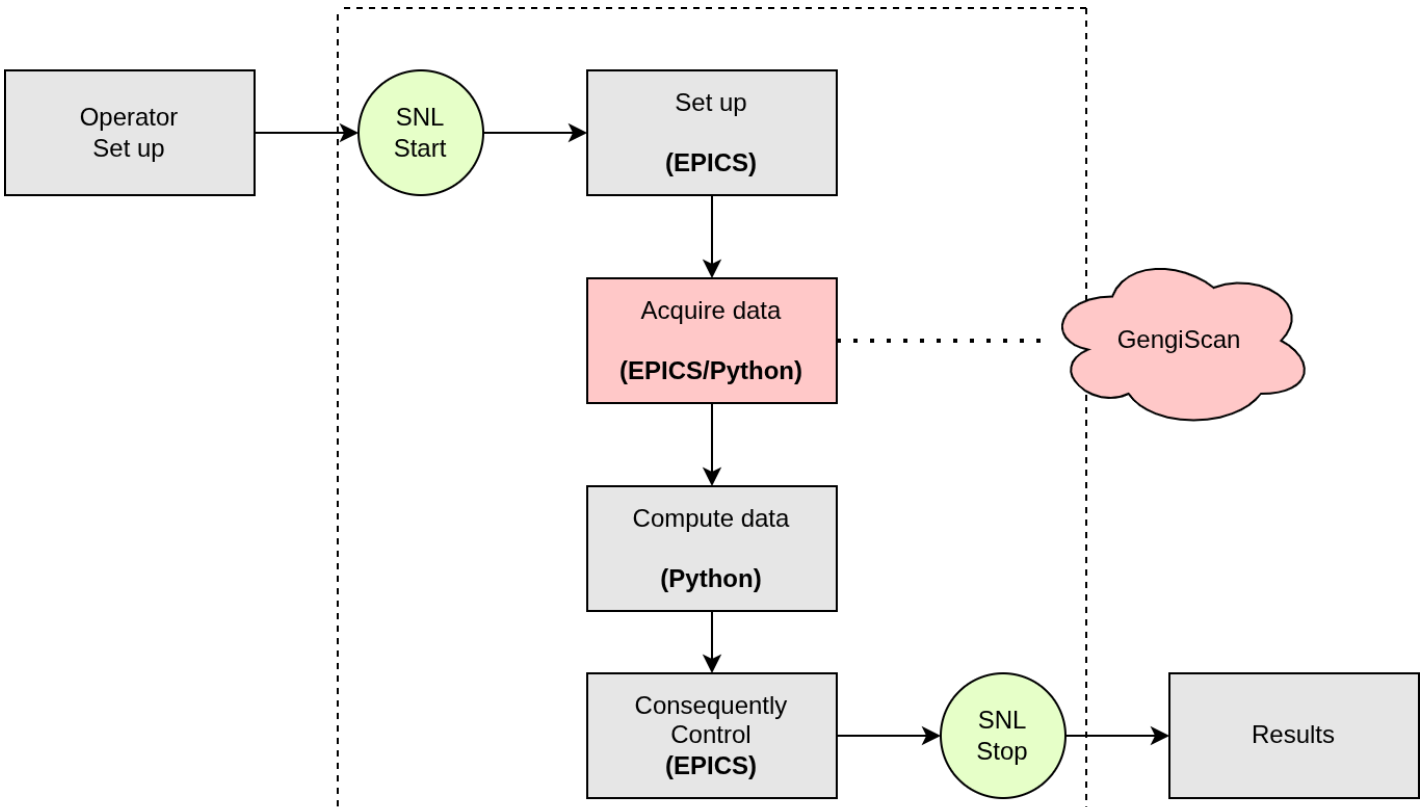
SNL start: EPICS set up



- get initial values
- set GengiScan PVs
- regarding **ToF** for example:
  - detune the involved cavities



GengiScan


Operator
Set up
SNL
Start
Set up
(EPICS)
Acquire data
(EPICS/Python)
GengiScan
Compute data
(Python)
Consequently
Control
(EPICS)
SNL
Stop
Results

GengiScan

Generic application to perform scan (half Python, half EPICS)

```
- Positioner: any actuator that affect the beam measurement
- Acquisitioner: system that acquire data for beam diagnostic
- Scan: a multiple-steps procedure used to save acquisitioner data depending on positioners position
```

see presentation *"GengiScan" by Stephane TZVETKOV*

GengiScan generates **formatted YAML** data file

- accelerator experts adapt their Python analysis scripts to this format

GengiScan - ToF

Simple use case :

- get BPM phases
- store data (with operator input) into **formatted YAML** file →

```yaml
context:
...
  beam info:
...
    particules: proton
  constant PVs:
    floats:
      - name: SL-APP:DIA-TOF-1:CtxCstFloat1Mes
        val: 1.3
        desc: estimated kinetic energy
        prec: 3
        egu: MeV
        inp: SL-APP:DIA-TOF-1:BeamEnergyContext NPP
      - name: SL-APP:DIA-TOF-1:CtxCstFloat2Mes
        val: 0.2
        desc: kinetic energy range
        prec: 3
        egu: MeV
        inp: SL-APP:DIA-TOF-1:BeamEnergyRange NPP N
      - name: SL-APP:DIA-TOF-1:CtxCstFloat3Mes
        val: 176.0
        desc: BPM frequency
        prec: 0
        egu: MHz
        inp: SL-APP:DIA-TOF-1:BpmFreqMes NPP NMS
      - name: SL-APP:DIA-TOF-1:CtxCstFloat4Mes
        val: 2.0
        desc: number_of_Bpm
        prec: 0
        egu:
        inp: SL-APP:DIA-TOF-1:BpmNbRb NPP NMS
      - name: SL-APP:DIA-TOF-1:CtxCstFloat5Mes
        val: -51.099998474121094
        desc: measured BPM 1 phase
        prec: 2
        egu: deg
        inp: SL-APP:DIA-TOF-1:BpmOffPhiMes NPP NMS
...
      - name: SL-APP:DIA-TOF-1:CtxCstFloat8Mes
        val: 1.2384000000000002
        desc: position BPM 1 phase
        prec: 3
        egu: m
        inp: SL-APP:DIA-TOF-1:BpmPosMes NPP NMS
...
    strings:
      - name: SL-APP:DIA-TOF-1:CtxCstString5Mes
        val: SL-MBT-MAG4:DIA-BPM-2
        desc: BPM 1 name
        inp: SL-APP:DIA-TOF-1:BpmNameTextIn NPP NMS
...
```

GengiScan - EPICS Side

GengiScanApp is integrated into the **topMcs** which includes all the high-level applications.

Several templates needs to be instantiated, for example to set the **positioners** and **acquisitioners** :

```
...
## gengiscan core
file gscan_core.template {
    pattern { P  }
            { "${P_GSCAN}" }
}
...
## gengiscan context constants
file gscan_context_constants.template {
    pattern { P           IDX  DESC_FLOAT                    INP_FLOAT                       DESC_STRING         INP_STRING          TYPE_WAVE  }
            { "${P_GSCAN}" 1   ""                            "${P_GSCAN}BeamEnergyContext"     ""                  ""                            }
            { "${P_GSCAN}" 2   ""                            "${P_GSCAN}BeamEnergyRange"       ""                  ""                            }
            { "${P_GSCAN}" 3   "BPM_frequency"               "${P_GSCAN}BpmFreqMes"            ""                  ""                            }
            { "${P_GSCAN}" 4   "number_of_BPM"               "${P_GSCAN}BpmNbRb"               ""                  ""                            }
            { "${P_GSCAN}" 5   ""                            "${P_GSCAN}BpmOffPhiMes"          ""     "${P_GSCAN}BpmNameTextIn"        "STRING" }
            { "${P_GSCAN}" 6   "first_downstream_BPM_Phi_Mes"  "${P_GSCAN}DownBpm1OffPhiMes"   ""     "${P_GSCAN}DownBpm1NameTextIn"   "STRING" }
            { "${P_GSCAN}" 7   "second_downstream_BPM_Phi_Mes" "${P_GSCAN}DownBpm2OffPhiMes"   ""     "${P_GSCAN}DownBpm2NameTextIn"   "STRING" }
            { "${P_GSCAN}" 8   "upstream_BPM_Pos_Mes"        "${P_GSCAN}BpmPosMes"             ""                  ""                            }
            { "${P_GSCAN}" 9   "first_downstream_BPM_Pos_Mes"  "${P_GSCAN}DownBpm1PosMes"      ""                  ""                            }
            { "${P_GSCAN}" 10  "second_downstream_BPM_Pos_Mes" "${P_GSCAN}DownBpm2PosMes"      ""                  ""                            }
}
```

```
...
## gengiscan positioners
file gscan_positioner.template {
    pattern { P            IDX }
            { "${P_GSCAN}" 1   }
            { "${P_GSCAN}" 2   }
...
}
...
## gengiscan acquisitioners
file gscan_acquisitioner.template {
  pattern { P            IDX NELM  FTVL    }
          { "${P_GSCAN}" 1   32768
"FLOAT" }
          { "${P_GSCAN}" 2   32768
"FLOAT" }
  ...
}
```

GengiScan - Python Side

Run as a daemon using **procServ**

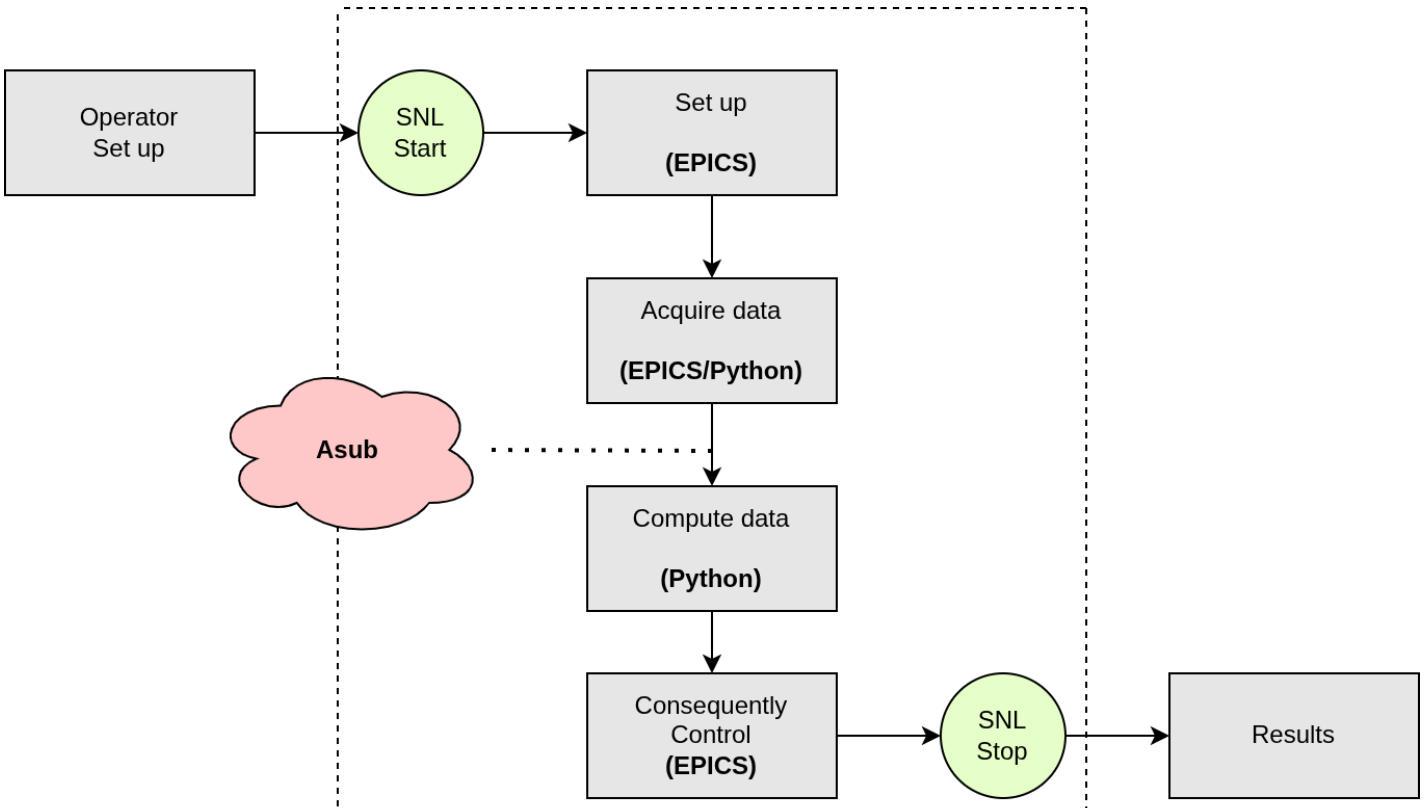
```
# envfile
PROCSERV_CHDIR=/iee/tops/epics-user/topSaraf/topMcs/gengiScanApp/gengiscan/dist/

PROCSERV_PORT=2007

PROCSERV_COMMAND="./gengiscan -p=SL-APP:DIA-TOF-1: --camonitor-timeout=6000 --output-folder=/data/gengiscan/tof/outputs --output-backup-folder=/data/
gengiscan/tof/outputs --log-folder=/data/gengiscan/tof/logs --log-backup-folder=/data/gengiscan/tof/logs"
```

ProcServ wrapper using **envfile** and **systemd services**

Same **GengiScan** executable for all high-level applications :

- arguments are adapted, such as the PVs prefix

Interface EPICS/Python



Interface EPICS/Python - ToF - using Asub

SNL writes in PV –> triggers **Asub** –> executes Python executable

```
executeAnalysisScript(aSubRecord *precord)
{
    int tofexecute = *(epicsUInt32 *)precord->a;       // execute tof
    l4e_info(MyApp,"%s(): Start; \n", __FUNCTION__);
    if (tofexecute == 1) {
        l4e_info(MyApp,"Analysis the data");

        const char * command = "../../../TofApp/TofAppPy/dist/tof ";
        system( command );

    }
    l4e_info(MyApp,"%s(): Stop; \n", __FUNCTION__);
    return 0;
}

epicsRegisterFunction(executeAnalysisScript);
```

System command & **relative path** –> not ideal but reliable because:

- Irfu Epics Environment (IEE) running on standard SARAF machines
- executables managed by Continuous Integration / Continuous Development

Main SARAF CI/CD

One job to get all **executables**, and **distribute them** at the right path:

```yaml
...
get-generated-executables:
    image: ldisc-container-registry.extra.cea.fr/iee-allinone:latest
    stage: build
    needs: [get-submodules]
    script:
        - *job-common-whoami
        - ./CICD/get_generated_executables.sh --output_folder ./CICD/builds/
    artifacts:
        paths:
            - "CICD/builds/"
            - "**/dist/"
...
run-saraf-delivery:
    stage: deploy
    needs:
        - job: get-generated-executables
          artifacts: true # Python executable
...
```

For more details about the main SARAF CI/CD :

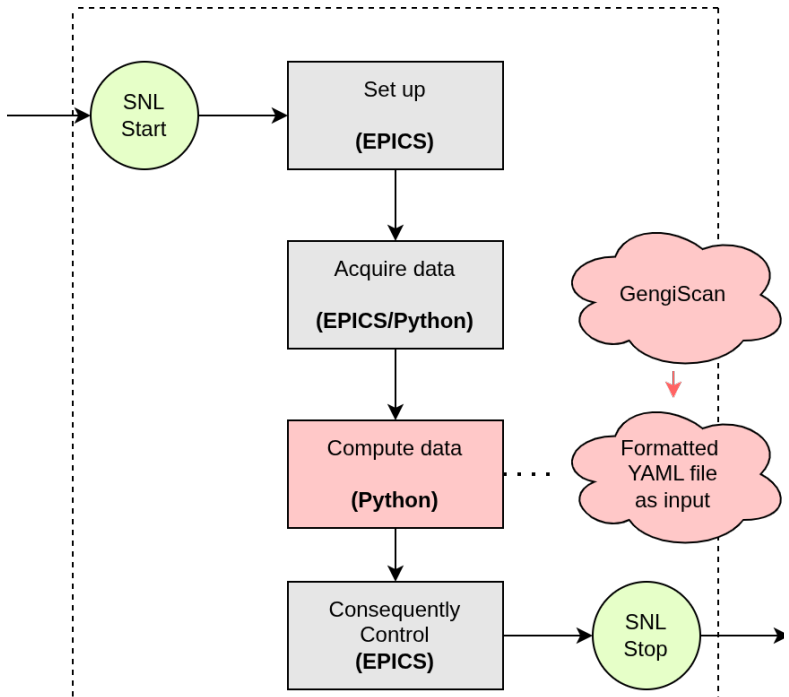
- see slide 8 of the presentation *"Remote integration and commissioning of an EPICS-controlled accelerator" by Gabriel DESMARCHELIER*

Python analysis scripts

Written by **accelerator experts** :

- use the **formatted YAML** data file generated by GengiScan as input

High-level applications CI/CDs generate the executables of the Python analysis scripts



Python analysis scripts - generating executable

Scripts are built as executables using **PyInstaller** and **Poetry**

```python
# build.py

import PyInstaller.__main__
from pathlib import Path

HERE = Path(__file__).parent.absolute()
path_to_main = str(HERE / "../tofapppy/tof.py")

def install():
    PyInstaller.__main__.run([
        "--clean",
        "--onefile",
        "--hidden-import",
        "epics.clibs",
        path_to_main,
    ])
```

Python analysis scripts - ToF - CI/CD

Scripts are built **automatically**

```yaml
generate-new-executable:
    stage: build
    allow_failure: false
    before_script:
        - poetry install -v
        - poetry run Python tofapppy/tof.py -h

    script:
        - poetry run build
        - ./dist/tof -h
```

Then push on Gitlab package registry.

Python analysis scripts - ToF - CI/CD: docker image

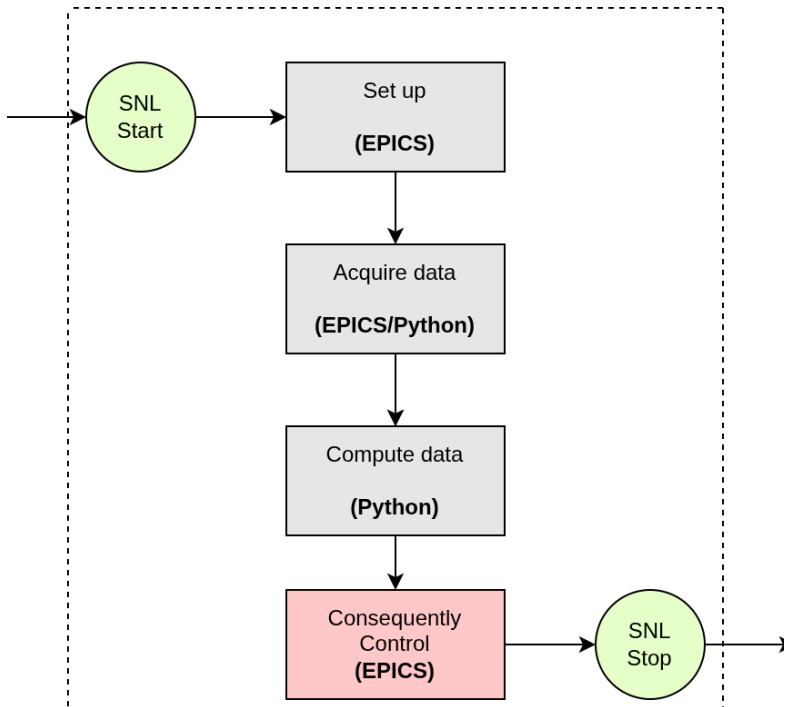
SARAF – IEE – ***CentOs 7 (deprecated)*** = specific environment

CIs use specific docker images from a self-hosted docker registry:

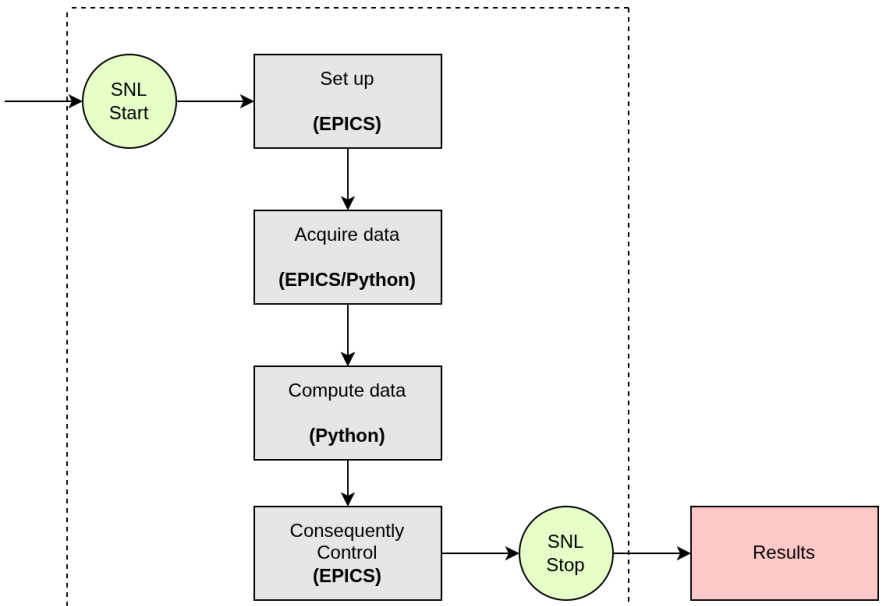
```
default:
    image: ldisc-container-registry.extra.cea.fr/iee-Python
```

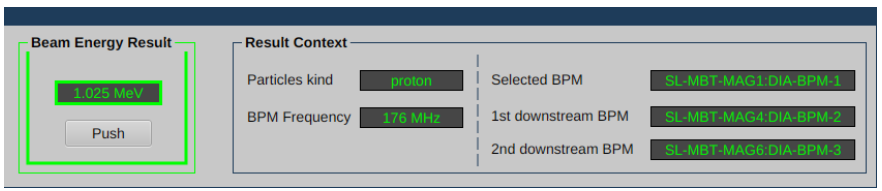
Before SNL stop

- Restore the original settings
- Use the result to configure equipments

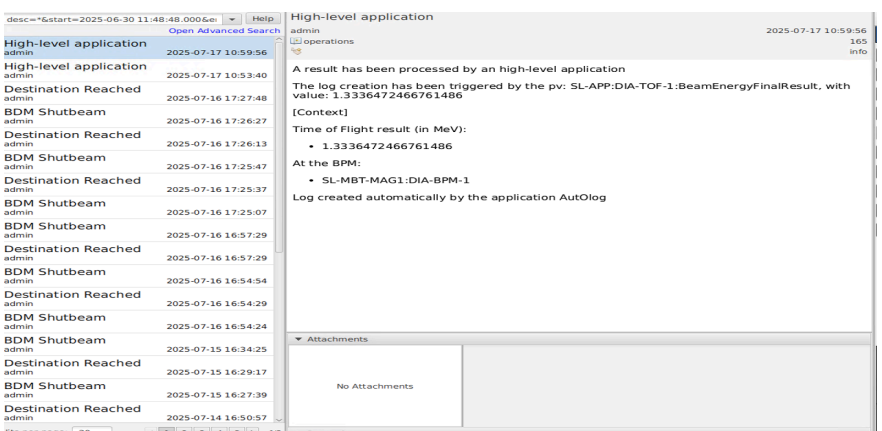


Results - ToF





AutOlog:



4. Status

Status

Latest ToF updates :

- removed automatic detuning of the cavities via SNL
- fixed an error in Python analysis script

ToF has been fully **tested and validated with beam** remotely at the beginning of 2026.

Status

Waiting for beam sessions for testing :

- BpmCalib
- EmitScan
- CavityPhasing

First version developed, requires simulated testing :

- BeamAligner

Conclusion

- Accelerator experts appreciate Python scripts.
- The choice of this architecture facilitates collaboration between accelerator experts and EPICS developers/integrators.

Thank you