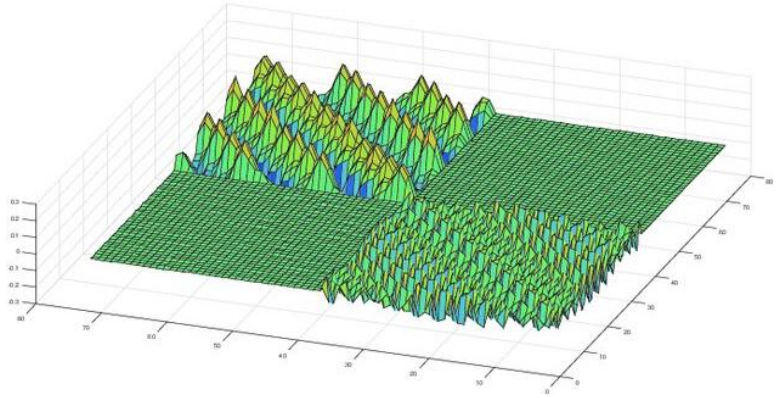
pyAML

python accelerator middle layer

Joint technology platform for design, commissioning and operation of particle accelerators.



https://github.com/python-accelerator-middle-layer

Jean Luc Pons

NOBUGS 2026

pyAML

ESRF | The European Synchrotron

PYAML

pyAML is a ControlSystem independent set of tools to operate a high energy particle accelerator and to provide a compatible interface to a simulator (Accelerator Toolbox) to test scripts and applications. Simulated Commissioning is also planned. pyAML is still in a development and specification phase.

For some specific operational tools, access to both **peered** simulator and control system are required at once. For instance, for tools which are model dependent.

```
# Get peer simulator handle
design = self.peer.peer.design

# handles
quad = design.magnet.get(quadname)
sth = design.magnet.get(steererhname)
stv = design.magnet.get(steerervname)
orbit = design.bpms.get(bpmname).positions
tune_design = design.get_tune_tuning(tunename)
tune_live = self.peer.get_tune_tuning(tunename)

# Get tune from live and adjust the model to improve quad response phase
tune0 = tune_design.readback()
tune = tune_live.readback()
logger.debug(f"Live tune: {tune}")
logger.debug(f"Model tune: {tune0}")
tune_design.set(tune)
logger.debug(f"Model tune: {tune_design.readback()}")
```

Ex: Model dependent beam based alignment tool

PYAML STATUS

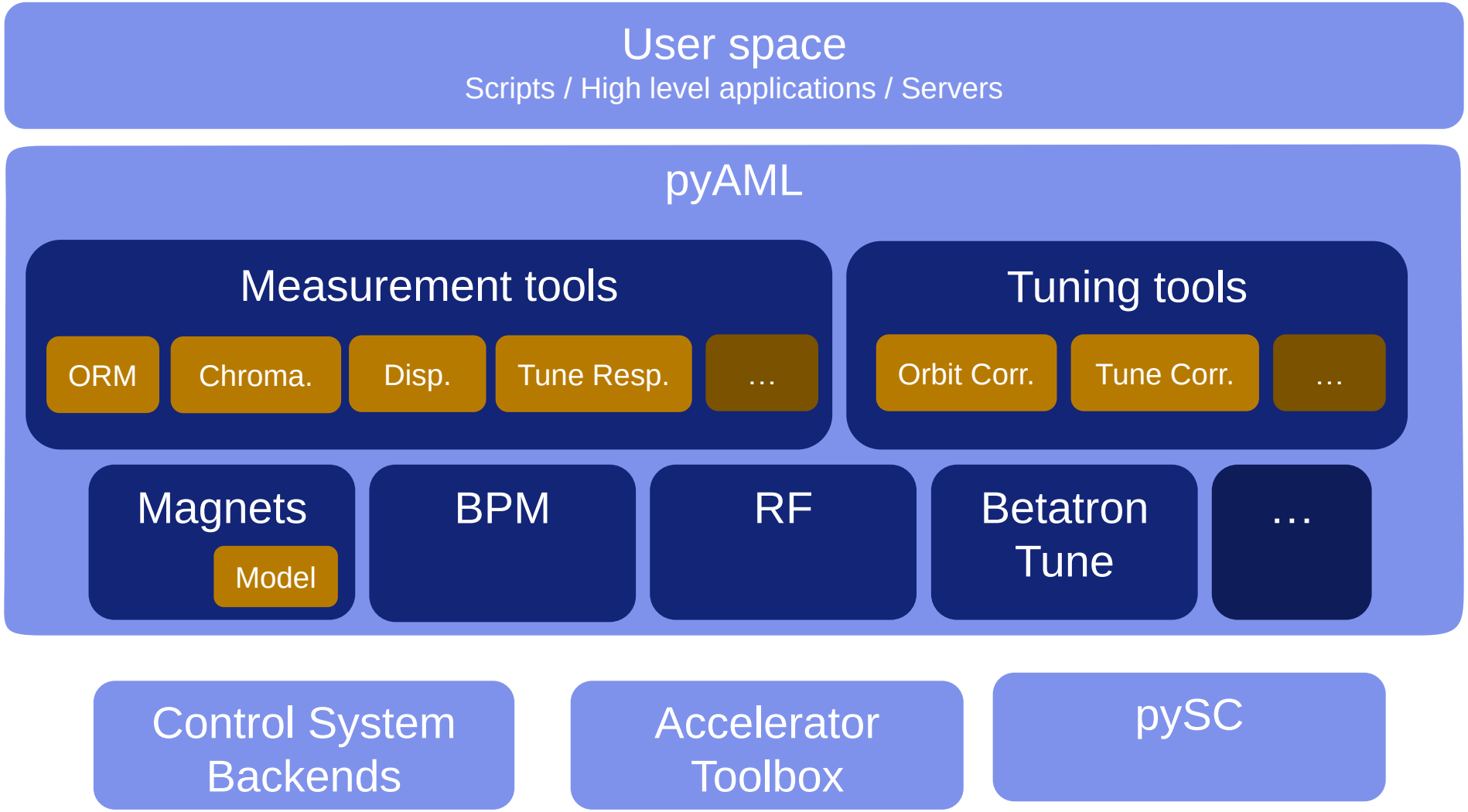
DONE

- Tune response matrix measurement
- Tune adjustment
- Chromaticity response matrix measurement
- Chromaticity adjustment
- Orbit Response Matrix measurement
- Orbit correction
- Dispersion measurement
- Beam Based Alignment

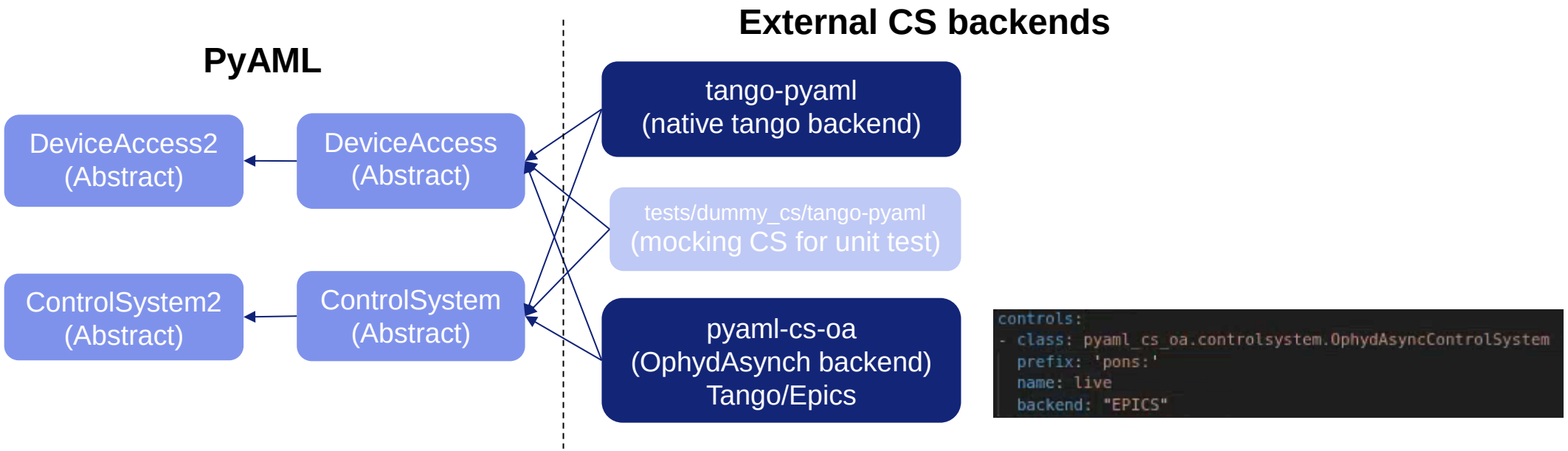
TODO

- Linear optics correction
- Non linear optimization
- Complex combined function magnet (several elements, several multipoles)
- Improve abstraction of element layer for portable code
- …

MAIN ARCHITECTURE



CONTROL SYSTEM ABSTRACTION



- Change of PyAML abstract interfaces can be handled by overriding ***DeviceAccess*** and ***ControlSystem*** classes to avoid breaking backward compatibility. CS backends can be updated later to get new features
- Implementation of new CS backend has to be clearly documented. However, some example of backend implementation are already available in the example section of pyAML:
  https://github.com/python-accelerator-middle-layer/pyaml/tree/main/examples/other_examples/ControlSystem
- Code stabilization (handling of backward compatibility) is planned for the end of this year.

YAML (OR JSON) BASED CONFIGURATION

YAML configuration is just a way to save a pyAML setup. It is not mandatory, you can create your pyAML setup by code.

```python
from pyaml.accelerator import Accelerator
from pyaml.bpm.bpm import BPM
from pyaml.lattice.simulator import Simulator

bpm = BPM('BPM_C04-01')
simulator = Simulator(name='design',lattice='config/sr/lattices/ebs.mat')
sr = Accelerator(
    facility='ESRF',
    machine='SR',
    energy=6e9,
    simulators=[simulator],
    devices=[bpm]
)

print(sr.design.bpm.get("BPM_C04-01").positions.get())
```

Creation of an **Accelerator** by code

```yaml
class: pyaml.accelerator.Accelerator
facility: ESRF
machine: SR
data_folder: null
energy: 6e9
simulators:
- class: pyaml.lattice.simulator.Simulator
  lattice: 'config/sr/lattices/ebs.mat'
  name: design
devices:
- class: pyaml.bpm.bpm.BPM
  name: BPM_C04-01
```

```python
from pyaml.accelerator import Accelerator
sr = Accelerator.load("simple_bpm.yaml")
print(sr.design.bpm.get("BPM_C04-01").positions.get())
```

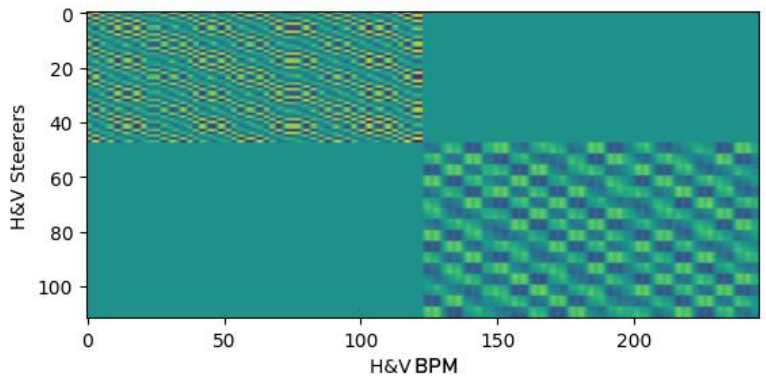
Creation of an **Accelerator** by loading a yaml file.

An advantage of yaml file is the possibility to use macro ex: **${env:USER_NAME}** and links.

```yaml
devices:
  - sr/quadrupoles/QF1AC01.yaml
  - sr/correctors/SH1AC01-C02.yaml
  - class: pyaml.bpm.bpm.BPM
    name: BPM_C04-01
    x_pos: srdiag/bpm/c04-01/SA_HPosition
    y_pos: srdiag/bpm/c04-01/SA_VPosition
  - class: pyaml.bpm.bpm.BPM
    name: BPM_C04-02
    x_pos: srdiag/bpm/c04-02/SA_HPosition
    y_pos: srdiag/bpm/c04-02/SA_VPosition
```

MEASUREMENT TOOLS

ORM measurement



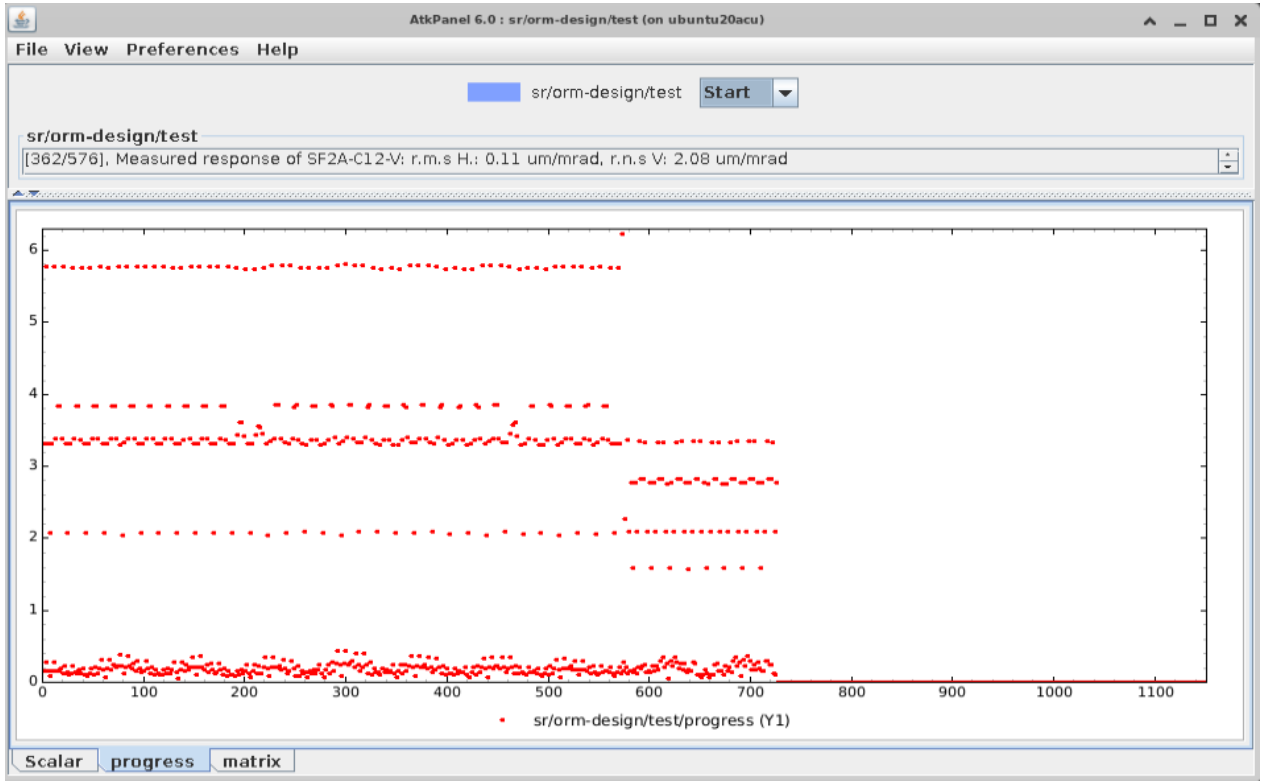
Callback workflow

```
for corr in correctors:
   set k0 - delta    -------> ACTION_APPLY
   get orbit         -------> ACTION_MEASURE
   set k0 + delta    -------> ACTION_APPLY
   get orbit         -------> ACTION_MEASURE
   set k0            -------> ACTION_RESTORE
```

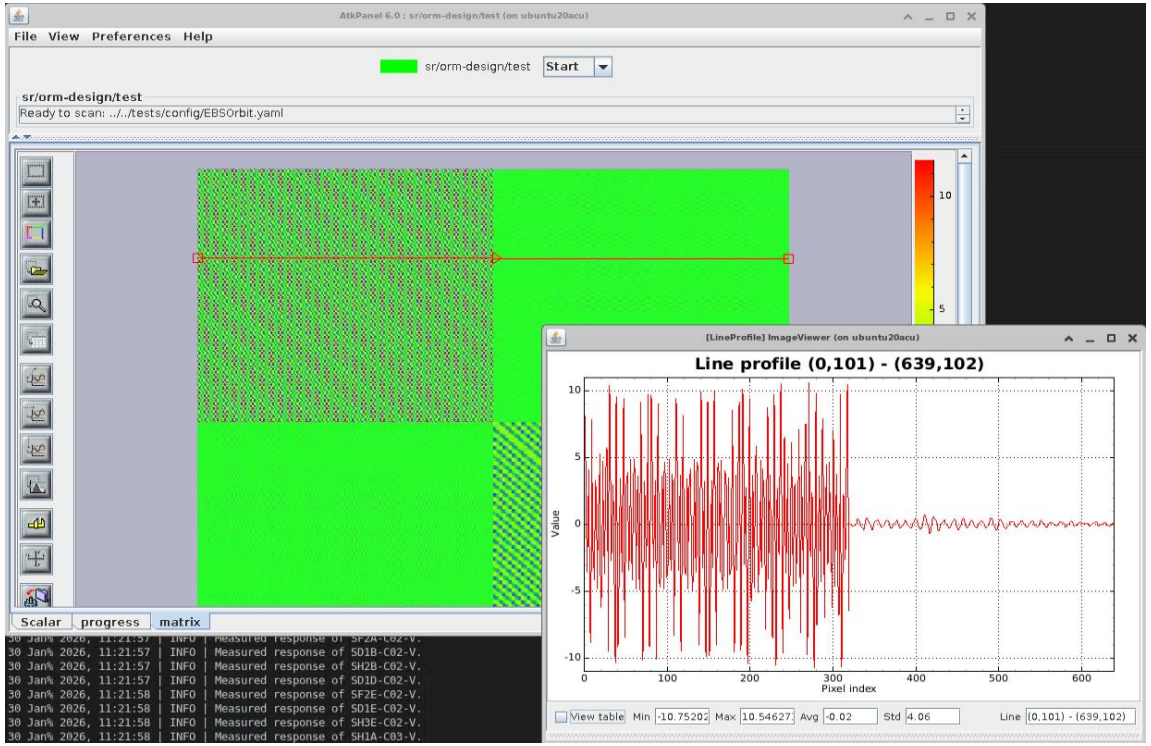
Callback is optional
Additional metadata must be serializable

```python
from pyaml.common.constants import Action
from pyaml.accelerator import Accelerator
from pyaml.tuning_tools.measurement_tool import MeasurementTool
import numpy as np
import matplotlib.pyplot as plt

# Load the accelerator
sr = Accelerator.load("/home/esrf/pons/pyaml/examples/use_cases/config/bessy2/bessy2.yaml")

# ORM callback
def orbit_callback(action: int, cdata):

    if action == Action.INIT:

        MeasurementTool.get_from_cb(cdata).latest_measurement["tune"] = []

    elif action == Action.MEASURE:

        # Add additionnal data to the scan
        tune = MeasurementTool.get_peer_from_cb(cdata).tune.readback()
        MeasurementTool.get_from_cb(cdata).latest_measurement["tune"].append(tune.tolist())

    return True

# Start scan
sr.design.orm.measure(callback=orbit_callback)
# Save measurement data including additionnal metadata
sr.design.orm.save("mat.json")

# Plot
orm = sr.design.orm.get()
mat = np.array(orm["matrix"])
plt.imshow(mat.T)
plt.ylabel("H&V Steerers")
plt.xlabel("H&V BPMs")
plt.show()
```

ORM MEASUREMENT TOOL IN A TANGO SERVER



Generic Tango client (ATKPanel) showing the scan progress.
Orbit h,v rms is plotted sequentially for each steerer. The effect of the beta function can be seen on top and the coupling on bottom.



Generic Tango client (ATKPanel) showing the ORM

Available in example section:
https://github.com/python-accelerator-middle-layer/pyaml/tree/main/examples/other_examples/ORM_Tango_server

CHALENGES

- Difficult to find agreement due to different work habits and different naming convention
- We communicate mainly using GitHub channel
- We have a zoom maintainers meeting every two weeks and a community meeting every 2 months
- We organize a 2 or 3 days workshop ~once a year
- Main active developers works in different institutes
  (DESY, BESSY, ESRF, SOLEIL, BNL)
- Progress is slow and conflicts due to disagreement are often difficult to solve using virtual channels

We would need more true human contact !