ACCESING HARDWARE BLISS

Antonia Beteva (ESRF)

BLISS – Short Presentation

BLISS stands for BeamLine Instrumentation Support Software.

BLISS is a control system which provides a global approach to run synchrotron experiments requiring to synchronously control motors, detectors and various acquisition devices thanks to hardware integration, Python sequences and an advanced scanning engine.

As a Python package, BLISS can be easily embedded into any Python application. BLISS data management features enable custom scripts to perform online data analysis.

BLISS ships with tools to enhance scientist users experience:

- a web portal to get access to BLISS applications
- a centralized logs viewer
- a configuration application
- a powerful command line interface
- an online data visualization application

Repository:

https://gitlab.esrf.fr/bliss/bliss

Documentation:

https://bliss.gitlab-pages.esrf.fr/bliss/master/

BLISS – Technical Choises and Design Principles

Written in Python

All-in-one solution

configuration management, logging, communication protocols, hw drivers, shell (CLI), scanning engine, live data display, publishing and saving

Mini frameworks

motion control, regulation, monochromator, spectrometer, MCA...

I/O based on gevent

cooperative multi-tasking, asyncio interoperability

Built-in direct hardware control

TANGO extensions for interoperability with other systems, or in case of shared hw

Scanning with the acquisition chain

Tree structure to represent "trigger" relationship of devices in a scan

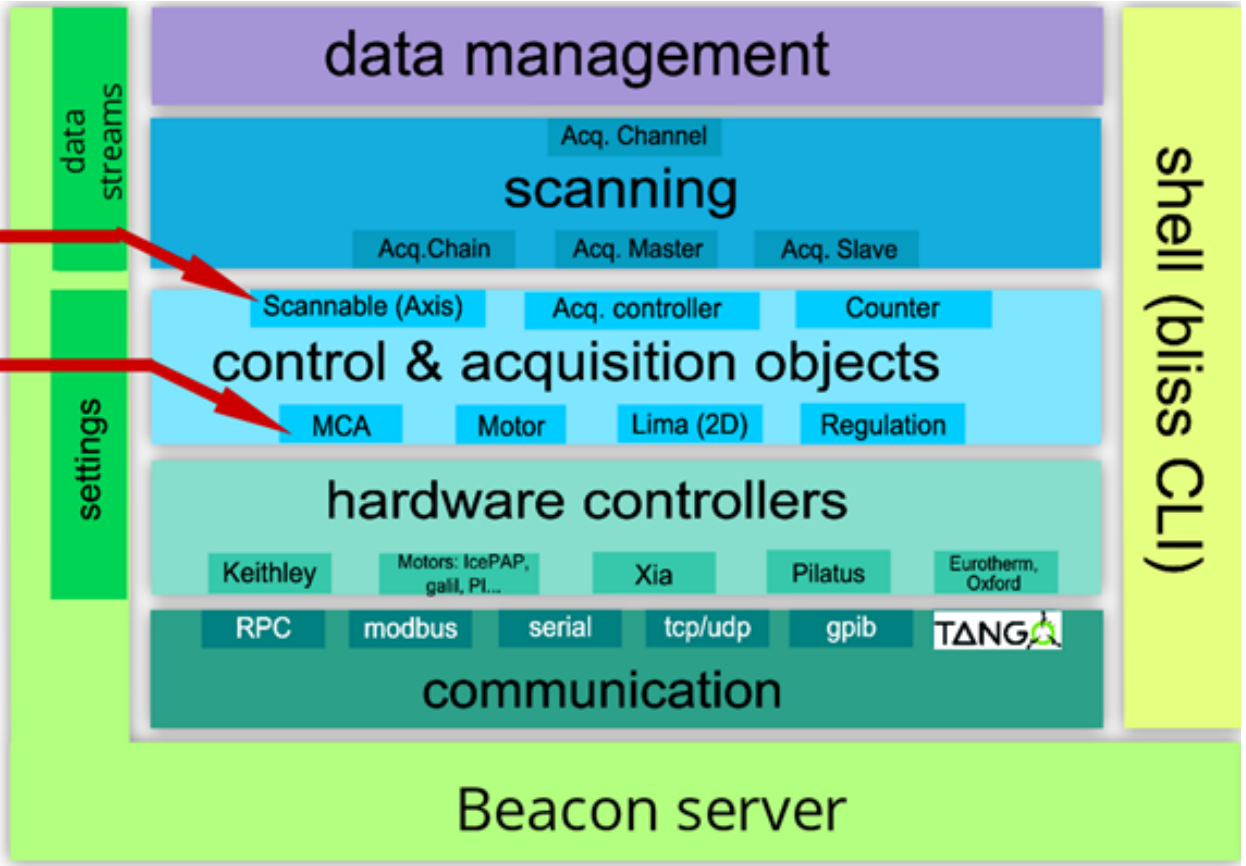
Data publishing,

acquisition made independent of storage

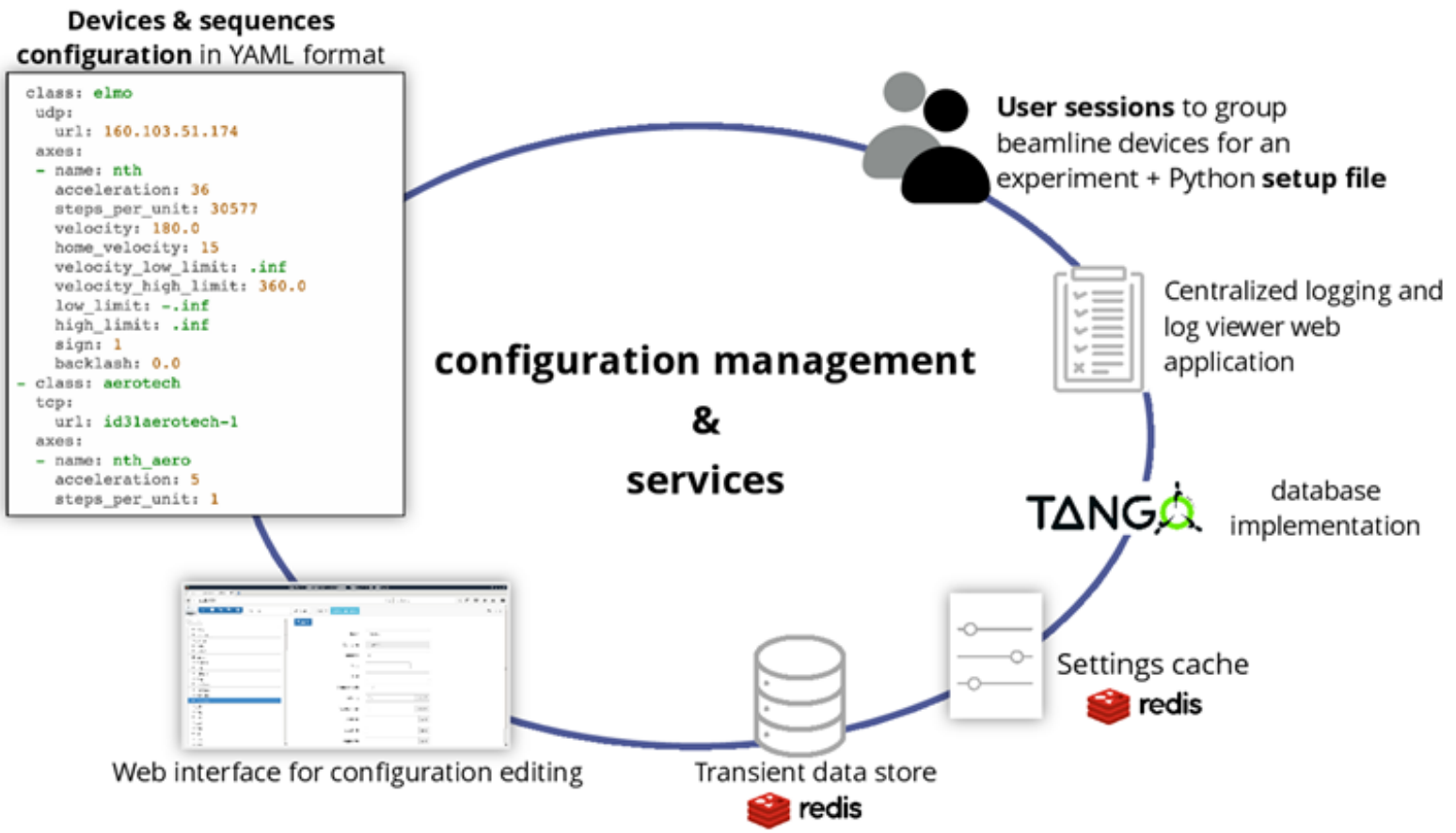
BLISS – Architecture Overview

Adaptation layer, to make hardware controllers behaving properly in scans, depending on the scan

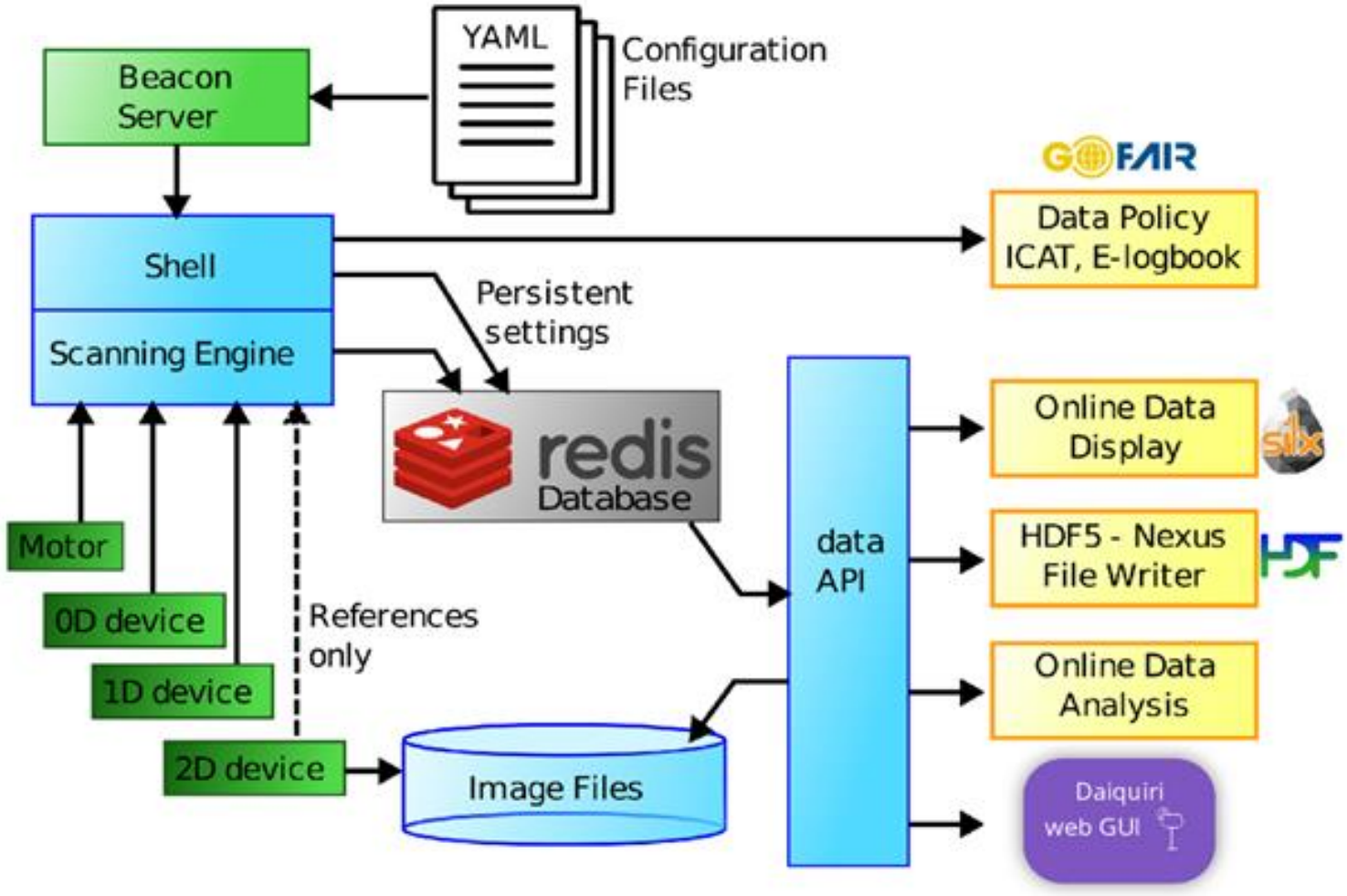
Mini frameworks within BLISS, to ease integration of same kind of hardware



BLISS – Beacon Server

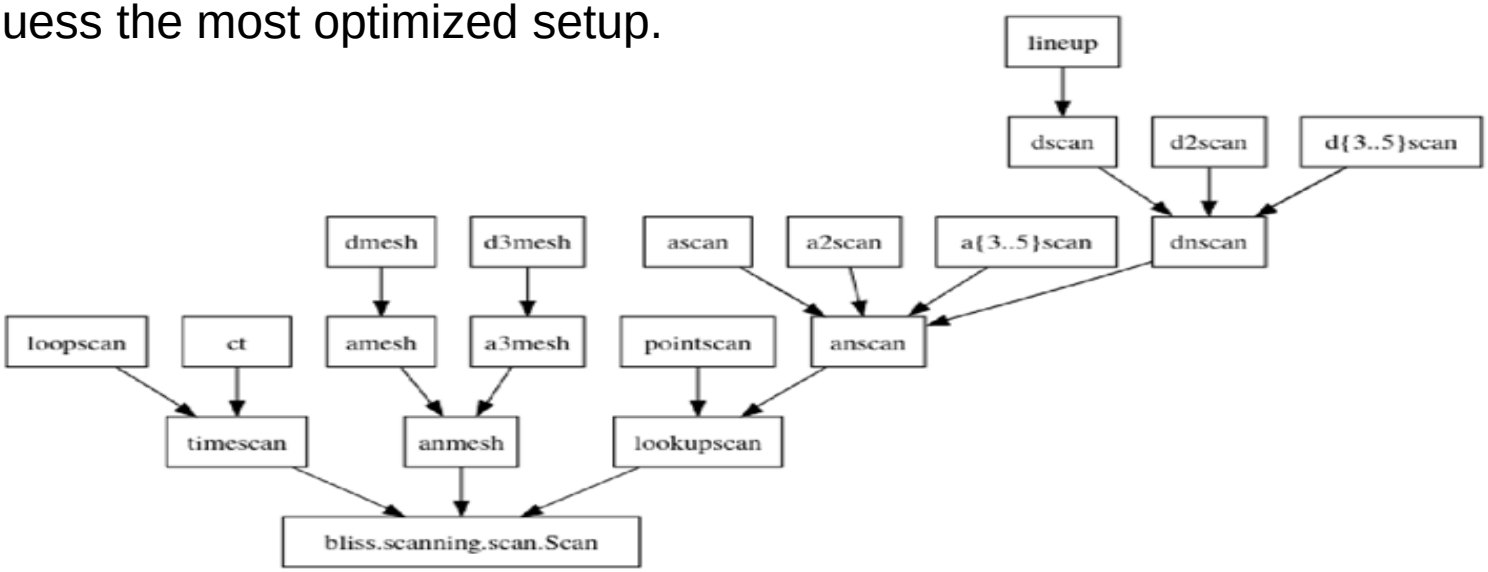


BLISS – Data Flow



BLISS – Scanning

- Powerful scanning engine for step-by-step and continuous scans.
- All scans are based on the same Scan object.
- The Scan object iterates through the Acquisition Chain that describes the triggering sequence (software or hardware).
- Data is sent to Redis and can be used for online display, online data analysis and saved.
- For “standard” scans the Acquisition Chain is built automatically, trying to guess the most optimized setup.



BLISS – What Users Like

- Python – scientist can take an advantage of a huge ecosystem for their own scripts
- Interactive data display tool – Flint
- Easy to switch between real and pseudo axes.
- Pseudo counters – any experimental parameter that can be measured during a scan.
- Software regulation loop – BLISS provides s Software Loop object that knows how to regulate with PID parameters.
- Data accessible immediately vis Redis for custom online display and data analysis.
- Large number of hardware controllers, ready for immediate use.
- Easy to change and save setup – useful when changing sample environment.

BLISS – mxcubecore Implementation

- BLISS is written in python, which makes it very convenient to embed, rather than defining commands and channels, which might end up with a huge number.
- Install BLISS in the same conda environment as mxcubecore. We can thus import the configuration and the "session", which contains all the objects and scripts we need.
- The Bliss class implements how a session is initialised and gives access to any of the session objects. The initialisation is done at the start of the mxcubeweb server, with the first hardware object which uses bliss.
- A BLISS session represents an experimental setup associated with experiment control sequences. A session has a list of objects from configuration and a setup file. The BLISS session is an object, so it has an yaml configuration file, which lists the bliss objects that the session will provide. The session setup file is python file where scripts like quick_realign, find_max_attenuation, centrebeam... are instantiated. These scripts can than be run as beamline actions or used in Hardware objects like XRF and EnergyScan.
- To update state and value and thus emit the standard valueChanged and stateChanged signals, the HardwareObject connect method to connect an object from a bliss session is used.

BLISS – mxcubecore Implementation

Bliss class

```
from mxcubecore.BaseHardwareObjects import
HardwareObject
from bliss.config import static
class Bliss(HardwareObject):
    """Bliss class"""
    def __init__(self, *args):
        super().__init__(*args)
    def init(self, *args):
        """Initialis the bliss session"""
        cfg = static.get_config()
        session = cfg.get(self.get_property("session"))
        session.setup(self.__dict__, verbose=True)
```

xml file

```
<object class="Bliss">
  <session>mxcubebliss</session>
</object>
```

BlissNState class

```
class BlissNState(AbstractNState):
"""Implementation of AbstartNState"""
                    …
  def init(self):
    self._bliss_obj = getattr(
                self.get_object_by_role("controller")
            self.actuator_name)
    self.connect(self._bliss_obj, "state", self.update_value)
```

Xml file

```
<object class="BlissNState">
    <username>Detector Cover</username>
    <actuator_name>detcover</actuator_name>
    <object href="/bliss" role="controller"/>
    <values>{"IN": "IN", "OUT": "OUT"}</values>
</object>
```