BLISS: experiments control of EBS beamlines

presented by Matias Guijarro
Beamline Control Unit, Software Group
ESRF, Grenoble, France
Extremely Brilliant Source (ESRF – EBS) project

- 150 M€ investment over the period 2015-2022
- 4th generation light source
- 100x improved brilliance and coherence of X-ray beams
- New state-of-the-art beamline portfolio

Full details in talk: FRAPL07
BLISS
BeamLine Instrumentation Support Software
Why BLISS ?
spec: 26 years driving experiments at ESRF

- Direct control of devices
  - easier to debug
  - restarting = reset
- Integrated tool
  - configuration
  - controllers for all kinds of devices
  - plotting
- Server mode to connect with external processes (GUI...)
- Commercial support
Poor macro language
No extensibility
Single task operation
Exclusive hardware control
Per-session configuration, no sharing
No built-in continuous scan framework
Limited data management
No code ownership, less freedom

spec: 26 years driving experiments at ESRF
The path to BLISS

- Python library + tools
- Technical choices
- Beacon: services for BLISS
- Hardware control
- Scanning & data acquisition
- Data management
- Sequences as genuine Python functions
BLISS Python library and tools
BLISS Python library and tools

Embed into any Python program

```python
>>> from bliss.common.axis import Axis
>>> from bliss.controllers.motors import IcePAP
>>> iceid2322 = IcePAP.IcePAP("iceid2322",
    {
        "host": "iceid2322",
        ["mbv4mot", Axis, { "address": 1,
                        "steps_per_unit": 80,
                        "velocity": 125,
                        "acceleration": 500
                        }
        ]
    }, [], []
>>> iceid2322.initialize()
>>> m = iceid2322.get_axis("mbv4mot")
>>> m.velocity()
125.0
>>> m.acceleration()
500.0
>>> m.position()
252.23750000000001
>>>
BLISS Python library and tools

Command Line Interface based on ptpython

matias@kashyyyk:~ % bliss -s test_session
test_session: Executing setup...
Initializing 'heater`
...
Initializing 's1hg`
Done.

>>> ascan(m1, 0, 10, 30, 0.1, diode, save=False)
Total 30 points, 3.0 seconds

Scan 4 Mon Sep 11 11:58:03 2017 <no file> test_session user = guijarro
ascan m1 0 10 30 0.1

<table>
<thead>
<tr>
<th>#</th>
<th>timestamp</th>
<th>m1</th>
<th>diode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.50512e+09</td>
<td>0</td>
<td>499.112</td>
</tr>
<tr>
<td>1</td>
<td>1.50512e+09</td>
<td>0.345</td>
<td>500.799</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1.50512e+09</td>
<td>9.655</td>
<td>505.622</td>
</tr>
<tr>
<td>29</td>
<td>1.50512e+09</td>
<td>10</td>
<td>499.883</td>
</tr>
</tbody>
</table>
BLISS Python library and tools

Configuration web application
BLISS Python library and tools

Graphical interface for users: interactive web shell
BLISS technical choices
BLISS key concepts

All I/O based on **gevent** cooperative multi-tasking

Direct hardware control

Distributed control ownership & shared state

Persistent settings cache

Scan acquisition chain, represented as a tree

Transient data store
BLISS modular architecture

- Online data analysis
- Data visualisation
- Data archiving

**Data Management**
- Acq.Channel
- Acq.Master
- Acq.Device

**Scanning**
- Counter
- Axis
- MCA
- Image

**Hardware Controllers**
- Keithley
- IcePAP, Galil, PI, ...
- Xia
- Lima (2D)
- Eurotherm, Oxford

**Communication**
- zerorpc
- modbus
- serial
- tcp/udp
- gpib
- PyTango

**Beacon Services**
Beacon: services for BLISS
Beacon static configuration service

Beacon server

Devices & sequences configuration in YAML format

Sessions to group objects
Python setup file
User scripts

Web interface for configuration editing

Can replace TANGO DB
Conversion script provided
Beacon: example configuration

```
sybil:~/local/beamline_configuration % tree
.
| ├── beacon.rdb
| ├── eh
| | ├── diode.yml
| | └── __init__.yml
| └── motors
|     ├── bv.yml
|     ├── DtoX.yml
|     └── __init__.yml
|     └── md2.yml
|     └── mirror1.yml
|     └── slits.yml
|     └── table.yml
| └── oh
|     ├── bpm.yml
|     └── __init__.yml
|     └── motors
|         ├── bv.yml
|         └── __init__.yml
|         └── mono.yml
|         └── slits.yml
|         └── transfocators.yml
|         └── wagos.yml
| sessions
|     └── id232_setup.py
|     └── id232.yml
|     └── __init__.yml

bv.yml:
  motor object
  - controller:
    class: IcePAP
    host: iceid2322
    axes:
      - name: mbv4mot
        address: 1
        steps_per_unit: 817
        velocity: 0.3
        acceleration: 3
```
Beacon dynamic services

- Beacon server
- Services built on top of Redis
- Message broker
  - State sharing
  - Distributed lock
- Persistent settings cache
- Transient data store
BLISS Hardware Control
Direct hardware control

```
from bliss.config.static import get_config

cfg = get_config()
pz1 = cfg.get('psz1')
pz1.move(200)
```
Management of concurrent access

- Multiple BLISS processes means **concurrent access**
  - distributed control ownership
  - based on a protocol: ask Beacon for permission

- **State coherence**
  - hardware state is shared between all peers via **channels**
Management of concurrent access
Management of concurrent access

BLISS process A

IcePAP controller
psy1

acquire lock

BLISS process B

IcePAP controller
psy1
psz1
Management of concurrent access

BLISS process A

IcePAP controller

psy1

ok!

acquire lock

BLISS process B

IcePAP controller

psy1

psz1

.yml

.yml
Management of concurrent access

BLISS process A

IcePAP controller

psy1

move

ok!

BLISS process B

IcePAP controller

psy1

psz1

.acquire lock

.yml
Management of concurrent access

BLISS process A
IcePAP controller
psy1

move

state channels
update

psyl locked
to A

 BLISS process B
IcePAP controller
psy1
psz1

.yml

.to A
BLISS scans

- Acquisition chain
  - a tree with master & slave nodes
  - master triggers data acquisition
  - slave takes data

- AcquisitionMaster, AcquisitionDevice
  - wrappers around BLISS control objects

- Data writer
  - HDF5
Continuous scan example

- m0 position
- m0 speed
- Detector frame triggering
Continuous scan example

```python
sybil:~ % bliss
>>> from bliss.scanning.chain import AcquisitionChain
>>> from bliss.scanning.acquisition.motor import SoftwarePositionTriggerMaster
>>> from bliss.scanning.acquisition.lima import LimaAcquisitionDevice
>>> from PyTango.gevent import DeviceProxy

>>> m0 = config.get("m0")

>>> lima_dev = DeviceProxy("id30a3/limaccd/simulation")

>>> chain = AcquisitionChain()

>>> chain.add(SoftwarePositionTriggerMaster(m0, start=5, end=10, npoints=10, time=5), LimaAcquisitionDevice(lima_dev, acq_nb_frames=5, acq_expo_time=0.03, acq_trigger_mode="INTERNAL_TRIGGER_MULTI"))
```
Continuous scan example

```python
>>> SCAN_SAVING.template = '/data/id23eh2/inhouse/{date}/{sample}'

>>> SCAN_SAVING.sample = 'HAK1234'

>>> SCAN_SAVING.get_path()
'/data/id23eh2/inhouse/20170324/HAK1234'

>>> from bliss.scanning.scan import Scan

>>> my_continuous_scan = Scan(chain)

>>> my_continuous_scan.start()
```
Classic step-by-step scans

- Directly available as functions from 'bliss.common.standard'
  - Example: `ascan(axis, start, stop, npoints, count_time, *counters)`
- Default acquisition chain
- Use the same underlying framework as continuous scans
Model for organizing acquired data

- Mirroring of the Acquisition Chain tree
  - each device in the chain has a name
  - each device define 1 or more 'AcquisitionChannel' objects
- Acquisition channels
  - must have a name, a type and a shape
- Metadata
  - scan_info dictionary (\{ key: value, ... \}) associated with scans
Online data publishing

- While a scan is running, **data is published** to the redis database provided by Beacon.
  - Scalar values are stored directly.
  - Bigger data (images, spectra) is just referenced.
  - Configurable time to live (TTL).
- Any external process can access redis data to perform **online data analysis**, for example.
User Sequences
Sequences as Python functions

```python
from bliss import *  # imports generic scans, cleanup functions, etc
from bliss.setup_globals import *  # imports objects from session (setup)
import numpy  # I know you dreamt of it
import gevent

def set_detector_cover(in):
    wcidxx.set('detcover', in)
    # 5 seconds timeout waiting for detector cover to move
    with gevent.Timeout(5):
        while wcidxx.get('detcover_in') == in:
            time.sleep(0.1)

def my_super_experiment(name):
    safety_shutter.open()
    old_att = attenuators.get()

    def restore_beamline():
        set_detcover_open(False)
        attenuators.set(old_att)

    with cleanup(safety_shutter.close):  # cleanup is always called at the end
        with error_cleanup(restore_beamline):  # this will only be called in case of error
            attenuators.set(50)
            set_detcover_open(True)

    SCAN_SAVING.name = name
    MEASUREMENT_GROUP.enable('diode')

    data_node = dscan(m0, -5, 5, 10, 0.1)

    for data in data_node.walk_data():
        # do something useful with data...
```
Sequences as Python functions

from bliss import *  # imports generic scans, cleanup functions, etc
from bliss.setup_globals import *  # imports objects from session (setup)
import numpy  # I know you dreamt of it
import gevent

def set_detector_cover(in):
    wcidxx.set('detcover', in)
    # 5 seconds timeout waiting for detector cover to move
    with gevent.Timeout(5):
        while wcidxx.get('detcover_in') == in:
            time.sleep(0.1)

def my_super_experiment(name):
    safety_shutter.open()
    old_att = attenuators.get()

    def restore_beamline():
        set_detcover_open(False)
        attenuators.set(old_att)

    with cleanup(safety_shutter.close):
        with error_cleanup(restore_beamline):
            attenuators.set(50)
            set_detcover_open(True)
            SCAN_SAVING.name = name
            MEASUREMENT_GROUP.enable('diode')

    data_node = dscan(m0, -5, 5, 10, 0.1)
    for data in data_node.walk_data():
        # do something useful with data...

    with cleanup(safety_shutter.close):
        # cleanup is always called at the end
    with error_cleanup(restore_beamline):
        # only called in case of error

...
Conclusion
Project state

- Current state of deployment
  - MX beamlines are already running BLISS
  - 3 more beamlines (Materials Science) for the end of the year
  - Full deployment in 2020
- Project is in active development
  - Not ready for use outside ESRF yet
  - git repository
Conclusion

- **Long term project** for EBS beamlines
- Control paradigm: *keep what works, add new concepts*
- Python *scanning framework*
- Prepared for *current and future challenges*
  - scans with online feedback
  - data management
  - evolutive platform
Aknowledgements

BLISS core development team

+ ESRF BCU contributing members: A. Beteva, M.C.Dominguez, M. Perez, J. Meyer

ESRF Software Group: A. Goetz