MXCuBE 3

M. Eguraun, F. Bolmsten
MAX IV MXCuBE Team
8th Feb. 2016
Outline

- MXCuBE 1, 2, ..., 3
- Backend
- Frontend
- Demo
MXCuBE

- Macromolecular Xtallography Customized Beamline Environment
  - Started in 2005 at ESRF
  - Beamline control and data acquisition platform for running MX experiments

- Supported by the following partners: ESRF, Soleil, MAX IV, HZB, EMBL, Global Phasing Ltd, DESY, ALBA.
  - Common solution for scientist
  - Already tested software & builtin experience
  - Quick setup

ALBA, Dec. 2015
MXCuBE - Main Features

- Customizable interface for each beamline/facility (PyQt bricks, web)
- Hide the complexity of the Hardware to the user (and to the developers...) thanks to the usage of the **HardwareObjects**
- Reuse of existing code for different beamlines
  - same or similar hardware
  - same or similar experimental procedures
- A huge builtin experience (many years + many people + many beamlines)
MXCuBE - HardwareObjects

- A HO is not only hardware! Procedures/sequences etc
- Link between devices and the graphical interface
  - Through the HardwareRepository...
- Configured through xml files
- emitting signals to others HOs, graphical elements
- supported protocols: Tango, Spec, Exporter, Sardana

```python
class MicrodiffMotor(Device):
    def init(self):
        self.position_attr = self.addChannel({'type': 'exporter', 'name': 'position'}, self.motor_name)
    def getPosition(self):
        return self.position_attr.getValue()
    def move(self, absolutePosition):
        self.position_attr.setValue(absolutePosition)
```

udiff_omega.xml  MicrodiffMotor.py
MXCuBE - HardwareObjects

<!-- Example beamline setup file -->

<object class="BeamlineSetup" role="BeamlineSetup">
  <!-- Objects directly associated with hardware -->
  <object href="/transmission-mockup" role="transmission"/>
  <object href="/minidiff" role="diffractometer"/>
  <object href="/cats" role="sample_changer"/>
  <object href="/spec_mxCuBE/res" role="resolution"/>

  <!-- Software (abstract) concepts -->
  <object href="/shape-history" role="shape_history"/>
  <object href="/session" role="session"/>
  <object href="/lims" role="lims_client"/>
  <object href="/edna_config" role="data_analysis"/>
  <!--<object href="/workflow-mockup" role="workflow"/> -->

  <!-- Procedures and routines -->
  <object href="/energyscan" role="energy"/>
  <object href="/mxcollect" role="collect"/>

  <!-- Is it possible to change the beam wavelength. Should perhaps be associated with the diffractometer -->
  <tunable_wavelength>True</tunable_wavelength>

  <!-- Disables or enables the number of passes input box, used for acquisitions. -->
  <disable_num_passes>False</disable_num_passes>

  <!-- Should be moved to a detector object in the future -->
  <detector>
    <manufacturer>MAR</manufacturer>
    <type>marccd</type>
    <model>marmosaic</model>
    <px>0.07324</px>
    <py>0.07324</py>
    <cansum>no</cansum>
    <has_shutterless>False</has_shutterless>
  </detector>

  <!-- Default values for an acquisition -->
  <default_acquisition_values>
    <exposure_time>10</exposure_time>
    <start_angle>0.0</start_angle>
    <range>1</range>
    <number_of_passes>1</number_of_passes>
    <start_image_number>1</start_image_number>
    <run_number>1</run_number>
    <overlap>0</overlap>
    <number_of_images>1</number_of_images>
    <detector_mode>1</detector_mode>
  </default_acquisition_values>

  <!-- Default values for a characterization -->
  <default_characterisation_values>
    <exposure_time>5</exposure_time>
    <start_angle>0.0</start_angle>
    <range>1</range>
    <number_of_passes>1</number_of_passes>
    <start_image_number>1</start_image_number>
    <run_number>1</run_number>
    <overlap>0</overlap>
    <number_of_images>1</number_of_images>
    <detector_mode>1</detector_mode>
  </default_characterisation_values>
</object>
MXCuBE 3

- Beamline control and data acquisition as web application
- Modern technologies
- Future easier integration and maintenance
- Remote access in a more natural way
- Reuse existing HardwareObjects

Challenges:
- Refactor existing code, remove dependencies
- New design for the user interface
- Decoupling logic and interface: any client possible

https://github.com/mxcube/mxcube3
MXCuBE 3
Backend

- Python **Flask** microwebframework:
  - web server made simple
  - extensions (database, login, ...)
  - easily adaptable to your needs while scalable
  - big community
- http request **API**: rest like (but probably not fully rest)
  - an url for each function
  - Simple to add new features without changing existing ones
- Flask **socketio** for sending HO messages
  - server-client bi-directional communication, websocket based
- Reuse the existing Hardware Repository
Http requests

- API for the calls from client to server (GET, PUT, POST, DELETE)
- Decoupling the server and the client
- Should be easy to understand by the client
  - (http://example.com/queue/4/12/execute)
- Url routings for sample centring/video/queue: already working

Sample Centring API

```python
PUT /mxcube/api/v0.1/samplecentring/centring/start3click
Start 3 click centring procedure
Args:
    None
Return:
    'True' if command issued successfully, otherwise 'False'

Note:
This does not mean if the centring is succesfull or not
```

```python
PUT /mxcube/api/v0.1/samplecentring/centring/startauto
Start automatic (lucid) centring procedure
Args:
    None
Return:
    'True' if command issued successfully, otherwise 'False'
```

```python
def centre3Click()
    
    # Start 3 click centring procedure
    logging.getLogger('HWR.MX3').info('[Centring] 3click method requested')
    try:
        currentCentringProcedure = mxcube.diffractometer.start3ClickCentring()
        return 'True' #this only means the call was succesfull
    except:
        return 'False'
```
Adding new devices

- Existing HO framework makes easy the addition of new devices
- Clear decoupling
- Steps (roughly):
  - Write your new Hardware Object
  - Configure it (xml file, specific address, range, etc.)
  - Does the current http api support the new HO?
    - if not: add new routes
  - Tell the client how to make use of the api
Frontend REACT

- **Javascript/React** library (Facebook)
- Only for the user interface (the V in MVC)
- Virtual html DOM kept as internal state
  - Different components programmed independently
- Reusing existing code when the layout changes
Frontend REDUX

- **Redux** application architecture/pattern
  - Predictable state container for JavaScript apps ...
  - Unidirectional data flow
  - Changes on the internal state in a single place
Frontend REDUX+REACT all together
MXCuBE 3


diagram showing the interaction between Hardware Objects, Server API, and Client.
Layout

● A main objective was identified
  ● Improve the user experience

● And for that it is useful to
  ● Have a clean interface
  ● Use modern web technologies
  ● Learn current usage and feedback
- Experiment configuration in a batch like mode
  - All available samples

- Experiment management for each sample
  - centring mechanism
  - should also be automatic and transparent for the user

Transitions between views to be defined
Layout - first real design

- Common parts

Area for different views

Hiddeable

View selector

Not the final design!
Layout - first real design

- Sample Grid
Layout - first real design

● Sample Video
Layout - Queue

- Queue:
  - Current sample always on top
  - Todo and History (collapsible)
  - Drag&Drop, search, reorder, skip ...

Not the final design!
MXCuBE 3 - demo MD2
MXCuBE 3 - demo Client
MXCuBE3 People

Main developers:
- **MAX IV**: M. Eguirauñ, A. Milan-Otero, J. Nan, F. Bolmsten
- **ESRF**: M. Guijarro

Supported by:
- **MXCuBE collaboration**
- **MAX IV MX and KITS teams**
- **ESRF BCU team**
Thanks for your attention!