Meeting Minutes - MXCuBE Developers code camp 9th - 10th of December 2019

Introduction

The meeting was held at ESRF between the 9th and 10th of December 2019. For the first time a video conference option was made available for participants with limited possibilities to travel due to long distance or for other reasons. The code camp was in general successfully but the current setup makes it far less efficient than a “traditional” meeting, particularly since the audio kept dropping out.. The internet connection seemed to be the limiting factor, however the entire setup needs to be looked at to identify bottlenecks. As it stands, it would not be possible to run remote meetings routinely, but the video conferencing option should be kept primarily to make it possible for participants with long distances to travel (i.e. LNLS or NSRRC)

Participants

- Rasmus (GPhL) (At ESRF)
- Martin (Soleil) (At ESRF)
- Lais (LNLS) (Remote)
- Ivars (EMBL-HH) (Remote)
- Jordi (Alba) (Remote)
- Marcus, Antonia and Daniele (ESRF) (At ESRF)

Status Reports

The participants were asked for a status report and general news of what has happened since the last MXCuBE meeting the 15h of October.

Martin (Soleil):

- Extending characterisation, continuing previous work
- Working on ISByB, discussion on how to improve maintenance and development with their limited resources. There are currently three beamlines using ISPyB.

Ramus (GPhL):

- Strategy calculation software and simulation,
- Will soon get into MXCuBE3

Ivars (EMBL-HH):

- Tried current master with MXCuBE Qt, it took one to two days to have something running.
- Will soon create some PRs with the fixes he need to make in order to get it to work. Full deployment planned for next year.

Jordi (Alba):

- Working on b-zoom, calibration of the b-zoom and installation of the tango device server, created DigitalZoom hardware object.

Lais (LNLS):

- Worked mostly with MXCuBE, created a new docker image for MXCuBE3 based on Debian 10. The MX team is beginning to program the integration of their sample changer, a Stäubli robot, doing the integration from scratch.

Antonia

- Worked on improving the AbstractMotor and adapted Exporter and BlissMotor objects to use the new abstract class.
- Created a PR with the work done.

Marcus:

- Some work on the AbstractCollect and investigation of technical solutions for the DataObject concept that works for both Python 2.7 and 3
- Otherwise working on BSX3 since last meeting.

Organisation:

The organisers suggested that the code camp would be divided into discussion sessions with a practical session in the afternoon of the last day and asked the participants for their input. It was decided that the discussion session would be split into pieces and interleaved with practical sessions. Resulting in the following agenda:

December 9th:

8.30h - 11h	● General discussions on guidelines and considerations for AbstractClasses ● Review of all AbstractClasses
11h - 19h	Implementation of results from review, Done in smaller groups or individually

December 10h:

8h - 15h	Discussion on milestone 3 topics, DataObject and AbstractCollect
15h - 16h	Continued implementation of results from review
16h-17h	AOB

Review of Abstract classes

The review of the abstract classes started with a general discussion on the base HardwareObject and general considerations concerning abstract classes.

Decision: Changing from xml to yaml configuration, and consequently from HardwareObjectNode to ConfiguredObject superclass, would require a complete rewrite of everybody's configuration files. It is therefore decided to avoid large scale or wholesale change on this point, and the issue will be addressed piecemeal later.

Decision: It is a general rule that contained objects are properties; attributes are handled with get_ and set_functions.

AbstractActuator

The meeting discussed how to name the generic setter and getter methods for a value attribute for AbstractMotor and in general for HardwareObjects that could be considered to have a value. One proposal was to use 'value', 'get_value', and 'set_value' in all cases. Alternatively, we could make an exception at least for some HardwareObjects, to get a

clearer API by naming the accessor after the action it performs i.e move for motors or get_energy for retrieving the current energy. It was agreed to go with the first alternative as it is easier to remember and easier to use and apply generally.

Decision: Introduction of a AbstractActuator that will be the base class for everything that moves and can be considered to have a value. AbstractActuator will have the methods get_/set_value for setting its value.

Decision: AbstractMotor should inherit AbstractActuator and will thus have get_ and set_value.

Decision: The name of the HardwareObject is used to indicate which attribute is being referenced with get_ or set_value in the cases when it could be unclear. For example set_value would set the energy (and not the wavelength) in the case of the Energy HardwareObject.

Decision: AbstractActuators do need to store the last value that was set, in order to handle e.g. GUI updating correctly. The attribute storing this value should be called 'nominal_value' (and certainly not 'value').

Action: Antonia and Rasmus are to continue their work on AbstractMotor and create a AbstractActuator (to be used as a base class for AbstractMotor)

Action: Once the AbstractActuator and AbstractMotor are agreed and merged, Rasmus will go through the entire code base and change all relevant classes (also beamline-specific) to use the new inheritance.

This work was started during the first practical session and resulted in PR #440

AbstractNPosition and HardwareObject

A discussion on HardwareObjects that have discrete states was also conducted. The consensus was that a finite number of global states would be defined on HardwareObject or in another appropriate place. The 5 states mentioned during the discussion were READY, BUSY, WARNING, FAULT and UNKNOWN. The AbstarctNState object was discussed in conjunction with this and a question about its name was discussed. This as the name State refers to a predefined set of positions the object can have and this is easily confused with state definition mentioned above. It was therefore suggested to rename AbstractNstate to AbstractNPosition or similar. It was discussed how to map more detailed, hardware-specific states to the 5 basic states. Jordi raised the point that we should distinguish between the states that determine what actions can be performed on an object (e.g. the five basic states above), and the states that represent possible values or positions, which is a different concept.

Decision: A set of common states is needed to increase consistency.

Decision: Rename AbstractNState to AbstractNPostion or other suitable name, the discussion of suitable name will be conducted in: https://github.com/mxcube/HardwareRepository/issues/451

Decision: The methods currently implemented in AbstractMotor and AbstractActuator that are common to AbstractNPosition should be moved to a higher-level abstract class. The exact class structure will be decided once Jordi, Antonia, and Rasmus have produced a detailed PR.

Action: Everybody is to continue the discussion on #451 and Jordi and Antonia have volunteered to look closer at the actual implementation

See issue #451 created during the meeting

AbstractBeam

An AbstractBeam object will be introduced that hides the complexity of how the size and shape of the beam is determined. The AbstractBeam object will replace AbstractAperture, AbstractSlits and objects handling undulators.

Action: Create an AbstractBeam object (assigned to Ivars)
(https://github.com/mxcube/HardwareRepository/issues/442)

AbstractAttenuators

Action: Rename AbstractAttenutors to AbstractTransmission (Assigned to Martin issue #443)

Worked on during the first practical and resulted in PR #446

AbstractDataAnlysis

Action: Rename to AbstractCharacterision (Assigned to Marcus issue #444)

Worked on during the first practical and resulted in PR #447

AbstractDetector

Action

- **Add horizonal_position and vertical_position (motors)**
- **Humidity and temperature (as floats), width, height and other descriptive information as attributes when appropriate and as “dict” when used as metadata.**
- **Add get_beam_position() returns beam position in pixels (float, float)**

AbstractEnergy

Action: to be derived from AbstractActuator

AbstractMicroscope

The name Microscope is discussed, and the class is difficult to decide on, as the relevant area is organised differently in the two versions of MXCuBE (Qt, Web). The following is agreed:

Action: Rename AbstractMicroscope to AbstractSampleView; it should contain:

- **Shapes (Geometrical information about ROI's and POI's)**
- **zoom, front_light and back_light motors**
- **camera, video camera object**
- **get_snapshot method**

(Assigned to Ivars issue #449)

AbstractResolution

It is agreed that we do need this class, and that it should inherit from AbstractActuator.

AbstractMCA

It is agreed that to keep this class, and to move the fluorescence detector to here..

AbstractMachine (MachineInfo)

Action:

- **Create AbstractMachinefrom MachineInfo**
- **Rename MachineInfo Machine and inherit AbstractMachine**
    - **get_undulator_gaps()**
    - **get_machine_current()**
    - **get_machine_state_text()**

(Assigned to Lais issue #448)

AbstractProcessing

There have previously been two objects related to data processing called GenericParallelProcessing and autoprocessing.py. These could be generalised and inherit a AbstractProcessing object, which in a longer perspective also could inherit AbstractProcedure. The division in OnlineProcessing (result fed back to MXCuBE) and OfflineProcessing (procedure merely launched) is discussed, with some liking it, others proposing a single class with an online/offline switch.

Action: Create AbstractProcessing (Assigned to Ivars issue #452)

AbstractProcedure

It was briefly mentioned that it would be beneficial if all procedure like objects could, in the long term, inherit AbstractProcedure. That would currently be AbstractXRFSpectrum, AbstractCollect and AbstractEnergyScan.

AbstractSampleChanger

Accepted as OK for now.

DataObject and Collect

The afternoon of the second day was dedicated to the DataObject and structure of AbstractCollect. The current way of passing data within the application is done through dictionaries which makes it easy to add keys dynamically. This makes it difficult to know what is passed and is sensitive to typing mistakes or accidental misuse. The DataObject would provide a clear definition of the data passed around in the application by allowing for means to explicitly specify an attribute's type and its valid values. This makes it possible for linters and easier for humans to know which attributes actually exist. The DataObject could also be made immutable which means that risk for accidental misuse would be reduced.

The biggest concern is that a few sites are still using Python 2 while the native support for data classes were introduced in Python 3. The libraries compatible with Python 2 that allow for a similar technical solution have also announced that the Python 2 compatibility is not guaranteed (but the aim is to support Python 2 for as long as reasonably possible).

The options investigated were:

- Python 3 Data Classes, and pydantic: (https://pydantic-docs.helpmanual.io/)
- Marshmallow-objects module: (https://github.com/SVilgelm/marshmallow-objects) marshmallow (2.x)

- .attr module: (https://www.attrs.org/en/stable/) (https://www.attrs.org/en/stable/python-2.html)

The developers are reluctant to adapt a Python 2 compatible solution since the development represent some work that would most likely be thrown away in one or two years as the end of life of Python 2 is January 2020.

It's therefore suggested that DataObjects using the Python 3 Data classes could be used for Python 3 users if the client code still uses dictionaries internally. This would mean that a thin wrapper would be used for Python 3 that handles the conversion from DataObject to dictionary while the Python 2 code would be untouched. This would further allow for a simpler transition the day when all sites are ready to use Python 3.

Decision: Python 3 Data classes to be used for Python 3 but the client code still uses dictionaries internally (during a transition period).

Decision: The work on defining and standardising the data being passed should proceed using dictionaries without waiting for the transition to Python 3. A file called DataModel.py would be a useful location. The implementation could for the time being be e.g. a {tag:type} dictionary.

The structure of the collect object is briefly discussed and its decided that it should inherit AbstractProcedure. CCD-specific code should be removed.

Action: AbstractCollect should inherit AbstractProcedure (Assigned to Marcus issue #456)

Next meeting

Preliminary in April

AOB

No other business

Appendix A

Meeting notes for each abstract class (taken collectively during the meeting)

Class name	Comment
AbstractActuator.py	Derive from AbstractNstate and create TwoState or InOut
AbstractAperture.py	Alternatively within AbstractBeam ? - Inherited from the AbstractNState.
AbstractBeam	Rename BeamInfo to AbstractBeam HWR.beamline.beam.slits HWR.beamline.beam.aperture Move beam_position (which is the position on the viewer) to the SampleViewer.
AbstractSlits.py	Alternatively within AbstractBeam ?
AbstractAttenuators.py	To be replaced with AbstractTransmission inherit from AbstractActuator
AbstractCollect.py	See above
AbstractDataAnalysis.py	Change name to AbstractCharacterisation, easy to mixup Analysis and Processing Rename DataAnalysis.py to EDNACharacterisation.py?
AbstractDetector.py	Add horizonal_position and vertical_position (as motors), humidity and temperature (as floats), width, height and other descriptive information as attributes when appropriate and as “dict” when used as metadata. NB Detector radius should be calculated on the fly, not set directly. Remove distance_limits method and Add beam position

	Remove detector_mode - it is for CCD detectors. HWR.beamline.detector.get_beam_position() returns beam position in pixels (float, float) <*RHFOGH I like it, but is everybody happy with using a tuple?*>
AbstractEnergy.py	Decision on where to keep Undulators (move to AbstractMachineInfo) Derive from motor (Method names (get/set_value ?)) inherit from AbstractActuator
AbstractMicroscope.py	Decision on shapes and graphics, and where to keep them Rename AbstractSampleView Derive from HardwareObject HWR.beamline.sample_view.shapes HWR.beamline.sample_view.zoom, front_light, back_light, camera HWR.beamline.sample_view.get_snapshot(overlays=boolean)
AbstractEnergyScan.py	Should this and XRF eventually become a Procedure ?
AbstractFlux.py	Contained within, AbstractBeam
AbstractMachineInfo	Rename MachineInfo to AbstractMachineInfo - get_undulator_gaps() - get_machine_current() - get_machine_state_text()
AbstractMCA.py	Keep this class, and move Fluorescence detector here.
AbstractMotor.py	Add read_only (here or at higher level?) Velocity belongs at the AbstractMotor level rather than the (higher) Actuator level.
AbstractMultiCollect.py	To be replaced by AbstractCollect
AbstractNState.py	See discussion above

AbstractProcedure.py	Use same model for Queue, Collect and Procedure, result from discussion.
AbstractSampleChanger.py	OK as it is now
AbstractResolution	inherit from AbstractActuator
AbstractVideoDevice.py	Split into AbstractVideoDevice and BaseVideoDevice ?
AbstractXRFSpectrum.py	Should this and XRF eventually become a Procedure ?
AbstractOnlineProcessing	Previously known as GenericParallelProcessing. Inherit from AbstractProcedure.
AbstractOfflineProcessing	Review autoprocessing.py. Inherit from AbstractProcedure.