MXCuBE refactoring – the new Beamline Object

Rasmus Fogh

MXCuBE / ISPyB joint meeting
Berlin, October 2019

GΦL

Refactoring goals

- Less code, less complex code
  - Less to maintain
  - Less risk of misunderstandings or mistakes
- More standardisation, homogeneity
  - Easier to share
  - Easier to write generally useful code
- Long term advantage, but more work up front
- A **lot** of rewriting at the current stage

GΦL

Configuration

- Beamline-specific hardware
  - Beamline-specific parameters
  - Beamline-specific code, classes

- Pre-refactoring :
  - Configuration in .xml files
  - Identified and addressed by file name
  - Many links to the same object from different objects (e.g. Collect and Beamline_setup)

GΦL

Beamline-setup

SOLEIL PX2

```
<object class="BeamlineSetup">
   <object href="/diffractometer/diffractometer"
role="diffractometer"/>
   <object href="/diffractometer/omega" role="omega_axis"/>
   <object href="/diffractometer/kappa" role="kappa_axis"/>
<object href="/singleton_motors/photon_energy"  role="energy"/>
...
```

EMBL-HH P14

```
<object class="BeamlineSetup"  role="BeamlineSetup">
  <object href="/mini-diff" role="diffractometer"/>
  <object href="/eh1/diff-omega" role="omega_axis"/>
  <object href="/eh1/diff-kappa" role="kappa_axis"/>
  <object href="/energy" role="energy"/>
...
```

GΦL

Problems

- Role names are (mostly) standardised
  - But the same object may be defined in several different places – where should you look?
  - Each must be configured and maintained separately
  - You need one of the container objects to hand
  - E.g. get_wavelength
    in both Energy, Collect and/or Resolution

- HardwareRepository.getHardwareRepository()
  .getHardwareObject(filename)
  works from anywhere
  - But file names vary between beamlines

GΦL

Beamline Object proposal

- First proposal by Ivars Karpics (‘api/’)

- Current proposal by me
  - Merged into master
  - Beamline-specific configuration still TBD
  - Some aspects not finally settled

Beamline Object

- One central location to get hold of all hardware objects
- All access via standardised role names
- Protect against local naming variants (and typos)
- Specify name and type of common objects
  - Specify abstract class
  - Determines what you can code to
- Specify in code
  - So linters can check which attributes exist
  - Limits free configurability

GΦL

Beamline Object - contents

- Contained objects defined as properties

```
@property
def energy(self):
    """Energy Hardware object
    Returns:
        Optional[AbstractEnergy]:
    """
    return self._objects.get("energy")

__content_roles.append("energy")
```

  - Visible to linters and type checkers
  - Defined – with their type – in code
  - Remain visible in subclasses

GΦL

Beamline Object – attributes

- Configured attributes must be defined in __init__

```
def __init__(self, name):

    # List[str] of advanced method names
    self.advanced_methods = []

    # List[str] of available methods
    self.available_methods = []

    # int number of clicks used for click centring
    self.click_centring_num_clicks = 3

    # bool Is wavelength tunable
    self.tunable_wavelength = False
...
```

  - Visible to linters and type checkers
  - Defined – with their type – in code
  - Remain visible in subclasses

GΦL

ConfiguredObject class

- Replacement for HardwareObjectNode
- Superclass for Beamline
- Supports yaml-configured HardwareObjects

GΦL

Yaml configuration

- JSON-like
- Accepts comments
- Designed for configuration files
- Yaml 1.2 (ruamel.yaml)
  - Removed most gotcha's.

beamline_configuration.yml

```
_objects:
   !!omap
    # Hardware (Ordered dictionary):
    - session: session.xml
    - machine_info: mach-info-mockup.xml
    - transmission: attenuators-mockup.xml
    - energy: energy-mockup.xml
    - beam: beam-info.xml
…

# Non-object attributes:
advanced_methods:
  - MeshScan
  - XrayCentering
tunable_wavelength: true
disable_num_passes: false
...
```

GΦL

Yaml configuration loader

- Loads ConfiguredObject subclasses
- Enforces container hierarchy rooted at Beamline
  - Adds container-content links
- Only loads pre-defined content
- Only loads pre-defined attributes
- Works for loading normal hardware objects
  - With their own (nested) content
  - Hardware objects nested in Diffractometer (motors), Detector (distance), ...

GΦL

Beamline-specific subclassing

- Simple to write
- Maintains superclass content roles and data types
- Maintains superclass attribute names
- Extensions must be defined in subclass code
- New attributes can **not** be added in config files

GΦL

Beamline-specific code

- People **will** write beamline-specific code
  - What is the best way of organising it?
- ConfiguredObject subclasses forces you to code extensions explicitly.
  - Code is (often) documented and shared
  - Code is lintable
  - Attributes added in config are neither
- Why tempt people to bypass the specification system?

GΦL

END

GΦL