EDNA Kernel Code Camp
November 18th - 20th 2010

Participants

- Olof (ESRF)
- Jerome (ESRF)
- Mark (DLS)
- Graeme (DLS)

Pre-discussion on Thursday November 18th p.m. The following points were discussed in the code camp:

- (1) Improvement of logging
- (2) XIA2CORE / pipelining
- (3) Define Methods and Interfaces
- (4) EA → Topcased
- (5) Units
- (6) Plugin generator
- (7) Tango server / client
- (8) Executive summary
- (9) Import thread safe
- (10) Create a sandbox in the edna subversion

The following points were discussed only in the pre-discussion and will be dealt with in forth-coming EDNA kernel VC and code camps:

- We agreed that we should resume EDNA video conferences, at least for the EDNA kernel. We suggest Friday morning kernel meeting 8:30 UK / 9:30 CET, agenda on wiki
- Image + Array + HDF5&Path
- We agreed that we should be careful to make static methods thread safe
- Templates → error in UnitTestControl:
- Control plugin → feed with controlled plugin names
- Automatic unpacking of the input object (maybe solved by pyxb)
- different execute methods, execution verying with type
- Verbosity of assertion

(1) Improvement of logging:

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=612

We agreed that logging in EDNA should be improved. We implemented a new EDLogging class based on the standard python logging facility and interfaced it to EDVerbose so that it could be used for logging for all EDNA classes without any modifications. By deriving all EDNA classes needing logging from EDLogging, and by replacing "EDVerbose.xxx" to "self.xxx", one enables the possibility of distinguishing logs from different classes and even different instances of the same class.

However, the first tests of this new logging scheme revealed that it's considerable slower that the old EDVerbose for verbose output like the unit test suites (EDTestSuitePluginUnitMXAll: ~40s with python logging versus ~30 s with EDVerbose logging). Thinks could be an issue for EDNA applications that must run fast. We agreed that a solution to this problem would be to for normal execution of applications that must run as fast as possible it should be possible to configure the EDNA logging facility to use the old EDVerbose scheme. For applications not under tough time constraints, and for debugging purposes, the new scheme using python logging should provide an much enhanced logging facility for EDNA.

Olof is following up the actions concerning the improvements of logging.

(2) xia2core was implemented in a xia2 core exec plugin

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=541

This essentially now works modulo a couple of issues:

- For working on Windows we need to fix up the way it verifies the existence of executables. This is in the configure, making sure that the file exists. N.B. this is already available in xia2core - is it exported? well it is a static method in $XIA2CORE_ROOT/Python/Driver/DriverHelper.py:executable_exists. Tidy.
- We also need to fix up the test cases e.g. to not write output to /tmp/insulin.mtz (degeorgification)
- Quoting needs to be handled more cleanly - currently the quotes are already applied by the xia2core - if additional quotes are supplied this confuses things - however is standard in edna land. Currently bodged, should use shlex.split http://docs.python.org/library/shlex.html#shlex.split however this does not appear to work corerctly (something odd - looks more to me like a fundamental error in Python?!)

Timeouts could be done with threading thus:

```
timer = threading.Timer(float(self.getTimeOut()), self.kill)
timer.start()
        self.__subprocess = subprocess.Popen ( … etc … )
        self.__iPID = self.__subprocess.pid
```

```
self.__strExecutionStatus = str(self.__subprocess.wait())
timer.cancel()
```

N.B. will also need to ensure that kill() method is correctly implemented in xia2core exec process doohicky.

Two issues within the xia2core itself:

- Separate pipes for standard output standard error:
- Timeouts on things.

http://sourceforge.net/tracker/?func=detail&aid=3113314&group_id=211879&atid=1019527

Both of these relate to using select() in the file reads. Does select() work on windows? No. Perhaps better to have threads reading from the standard out and standard error and closing when the pipes are closed. This is a recommended solution from stackoverflow.com. Muh. N.B. timeouts mechanism given above *via* timer.

http://docs.python.org/library/select.html

In terms of test cases it will be useful to write some test cases for the new xia2 core edna plugin exec. This should use the uberportable test programs which are already bundled in the xia2core. To do.

Graeme is following up the work on implementing the xia2 core exec plugin

(3) interfaces ideas were dropped.

GW described with a toy example of how interfaces could work which was agreed to fit with the 4 pillars of edna. However OS said that while these could be sensible, and would indeed fit in, the complications involved in agreeing on the interface definitions within the wider edna collaboration would be prohibitive. GW agreed that this was the case, particularly recalling the dna data model discussions and the progress on MXv2.

Application level interfaces could work, however we would need the capacity for plugins to have multiple execute methods. GW will investigate this with the CCP4 folks. Would be particularly useful there for the standard CCP4 program apis.

(4) new methods for uml design seem effective

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=690

The new UML Editor Topcased seems very good, there are a couple of issues with null-pointer exceptions, but this is very minor and should be fixed in comming releases. In All the majority of enterprise architect functionality is there, and the uml to xml tool from edna-site works well. We agreed that we should try to move from EA to Topcased for the kernel data model this year.

(5) Units

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=692

Units were discussed, and some code was added and code reviewed during the meeting, the method that was decided upon was to keep the value in terms of the given unit, and not to convert to SI as standard. This means that the code is more backwards compatible.

(6) Plugin generator

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=699

It was decided that the Plugin Generator needed a good overhaul to sort out some problems which have been discovered in the last few EDNA training sessions. The conclusions were that the plugin generator should be a plugin itself, and the implementation of this was started. It was also decided that this would make it far easier to incorporate the generator as an eclipse plugin.

Mark is following up the work on the plugin generator.

(7) Tango EDNA server.

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=590

With the new EDJob functionality there is a minimal amount of code required to create the Tango EDNA Server side. It was commented that it may be useful to consider an EPICS implementation for this as well, which could make for another effective way to serve various Plugins. The EDJob also would make it easy to make another separate server if required.

Jérôme is following up the work on the EDNA Tango server.

(8) Executive Summary

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=698

Discussions were made about the Executive summary and it was decided that it should be approached in the same way as the EDLogging works, but not using the python logging methods. The key points for the summary should be

1. "__main__" style summary for the top level plugin
2. One line short summary per plugin
3. Few line brief summary per plugin
4. Full summary per plugin

It should also contain the tools to allow pieces of information to be added to a central "pile" which can be used for other summary information, for example

- log files by path
- Pieces of information not required by the pipeline but possibly useful for the User output
- etc.

(9) Import thread safe

http://www.edna-site.org/bugzilla3/show_bug.cgi?id=693

We added a new method "preImport" to the EDFactoryPluginStatic class for making imports thread safe. This has not yet been thoroughly tested.

(10) Create a sandbox in the edna subversion.

During this discussion it became apparent that an ".ednaignore" file could be contained in any directory which the automatic plugin finder should not bother to search, this would mean that the sandbox could be kept out of the plugin exploration unless required, but it also means that any directory which should not be walked by the explorer can be ignored.

EDNA Sandbox Readme

As agreed at the meeting at the ESRF 20th November 2010 we need a sandbox where quick developments may be made in the process of making proper new developments.

Intentions:

- this is a place where code for illustrating suggestions etc. can go, also things for discussion.
- when the discussion is complete the corresponding files should be removed.
- a directory should be created which helpfully indicates what is contained therein - e.g. codecamp/nov10/interface_discussion/

This should be ignored by the edna plugin finding code.

Appendix 1 : The Code Hammer

The term “code hammer” is now used for us for when we are programming with more enthusiasm than care - for example being trigger happy with search + replace when trying to change over something to use the xia2core plugin. Just for instance. :o) (It has to be said though that code hammers can sometimes be very efficient if combined with test-driven programming, enabled by the EDNA testing framework. If no testing framework is available, more delicate instruments than hammers should preferably be used when operating on the code...)

Appendix 2 : Google docs

This document was elaborated with Google docs:

- On the positive side it turned out to be a very fast way of creating a document with many authors since three of us literally edited the document at the same time. It’s also saving time to not have to copy the document between different computers.
- On the negative side not all kernel developers have sold their souls to google by having a google account so one of us (no name mentioned!) could not edit the document on-line.

Appendix 3 : Thoughts prior to the code camp

Action 1? Visit some of these requirements.

Possible Improvements Identified for the EDNA Kernel

G Winter 25 March 2010
During the implementation of the DIMPLE pipeline using the EDNA framework I have noted the following changes which could be made to the EDNA kernel and principles to make the implementation of plugins more straightforward. I also believe that these changes will make the kernel better, and will make the best of the data modelling efforts which have been made.

EKP #1: Automatic unpacking of input objects

Given that we have defined the data types and objects for the input and output, it should be trivial to automatically unpack the input data objects into plugin member

variables. I will produce a demonstration implementation of this. As a side effect, it would be also good to define getter and setter functions for these dynamically.
The plugins could therefore symmetrically take input XML documents or use set / get methods to provide input and output information. In some cases this may make the use less cumbersome.
In terms of debugging, it is helpful to have the full input document available. If we could also automatically repack, the process() method call could roll up the values which are actually going to be used for execution.

EKP #2: Definition of methods

In some cases I implemented two plugins which perform almost exactly the same task. It would be nice to do this once, then have methods which allow one route or another to be decided based on the implementation. For reasons of state it is better to have methods than to customise through inheriting.

EKP #3: Allow use of state

Although most of the input and output is handled through documents, it would be helpful to be able to define the state internally to the plugin and allow this to be used to cache information. Example: XDS plugin, storing intermediate results.
N.B. implementation of state makes debugging more tricky, though automated rolling of XML documents would help.

EKP #4: Define plugin types in data model

Next Steps

Four Pillars of EDNA

Modular
Data Model
Workflow
Testing

Would like to have methods, but this could break some of these pillars. Possible solution? Build on the current preProcess, process, postProcess principles and extend them to multiple functions.

So what we had in mind was to define an interface (perhaps as a Python class in interfaces next to datamodel in the plugin directory) which does exactly one task, for example:

```
class indexer:

def _preIndex(self):
        pass # should be overloaded, perhaps raise exception?

def _index(self):
        pass # should be overloaded, perhaps raise exception?

def _postIndex(self):
        pass # should be overloaded, perhaps raise exception?

def index(self):
        # create a thread, call _preIndex, _index, _postIndex, join thread

# then getter and setter methods for the things (data items) which relate to
# indexing - the translation from standard to program specific language can
# be handled in the pre / post methods.

class Mosflm(indexer, integrater, strategiser) # say
```

So the plugin can expose interfaces for indexing &c. Caller calls *only* index() which works the same as execute() in current plugins calling the overloaded pre_process, process, post_process methods inside the thread. As the call to the index will be blocking, the plugin can't be doing anything else so this will be thread safe.

THEN we implement a test for this interface. This can call the index() method and make sure that the plugin behaves itself. This test will then work for *all* implementations of this interface.

So: is this modular: possibly more so that we currently have.
does it respect a data model: yes, only more so.
does it respect the work flow? yes!
does it support testing? yes, more so!

EDNA, now more than ever (said he mangling something from The Player)

Action 2? xia2core + EDNA (EDNA bugzilla # 541) many notes follow below.

Action 3? Reporting and output.

Version of __name__ == '__main__' in the summary output i.e. only write things out if I am the toplevel plugin.

Carrying a container of interesting things out of plugins to help with composing said report.

Notes

EKP #1 FIXME we could still do with looking at this.

```
def adopt_init_args(obj, args, exclude=(), hide=False):
  del args["self"]
  for param in exclude:
    del args[param]
  if (hide == False):
    for key in args.keys():
      assert not hasattr(obj.__dict__, key)
    obj.__dict__.update(args)
  else:
    for key in args.keys():
      _key = "_" + key
      assert not hasattr(obj.__dict__, _key)
      obj.__dict__[_key] = args[key]
```

Also need a ‘toy’ system to show the benefits of interface specification etc. Did not to this in the edna repository but will hopefully be able to share the code before travelling.