ISPyB Developer’s Meeting

ESRF, December 9, 2022

Meeting Minutes

DRAFT

Participants

Not recorded as meeting was half in person, half on zoom.

Agenda

- Multi-sweep data collection
- Time frame and resources of introduction of Py-ISPyB
- Uniform handling of auto-processing
- ‘Full software stack for structural biology’

Multi-sweep data collection

Time frame and resources of introduction of Py-ISPyB

Clemens Vonrhein (CV) notes that it is extremely useful that there is a development environment to run PyISPyB. This will allow Global Phasing to try things out, and to make sure what ends up in ISPyB is actually what was intended. The way forward should be to have a well-documented function to slurp data into ISPyB (an API?); if it works for external programs it will also work for internal ones - whereas the converse is not necessarily true.

Peter Keller (PK) suggests one could load JSON or XML directly into a database field as a way to store program-specific information/ Karl Levik (KL) confirms this is possible, but Rasmus Fogh (RF) worries that information that is common to several programs could become hard to extract if not stored uniformly.

Alejandro de Maria (AdM) agrees that it is important to add support for multi-sweep data, and that this has to happen. But migration over the next year could be very demanding because you must guarantee against the live code on the beamlines breaking. Any change in the storage tables has to be applied retroactively to 20+ years of legacy data, which will require a well-prepared migration plan. CV would love a code freeze, and a migration plan would require one, in order to have a fixed

target. There is sure to be a lot of very valuable but undocumented code available and a code freeze would allow you to find it. Eliot Hall (EH) cautions that one would “have to be very careful” as database migration is hard to do well. It would be easier to use a new system only for new data. It is noted that there are many software tools that depend on the current DB schema. CV proposes that one could leave the current storage for sweeps untouched, and move over to a system of multi-sweep data storage based on the new SSX system; SSX will have to support multiple tiny wedges as well as multiple still images. AdM notes that the ESRF has had a code freeze for two years now. PyISPyB is being applied to SSX, the question is how to make the move for MX and other fields.

Marcus Oscarsson is asked how hard it would be for MXCuBE to adapt to a new data structure, but does not expect this to be a big problem. Contact to ISPyB is handled via an ISPyBClient hardware object, and a new one can be developed in parallel and switched in, beamline by beamline. The introduction of PyISPyB would mean that access endpoints would change. Autoprocessing might be a harder problem, as there may be more direct ISPyB access in that area. The discussion ends on the suggestion that the steering committee will ultimately have to decide the time frames, but without having agreed on a specific proposal.

Uniform handling of auto-processing

There was general agreement on the problem. Each site supports several different processing programs and has a system to view the results, but the views and the data presented are all site-specific and the system used to gather them is opaque. Users are treated to different views with differently defined data, and particularly for an outside program developer it is extraordinarily hard to figure out which data flow to which display (or ISPyB) slots following which triggers. It would be extremely helpful if views and triggering were documented, and especially tif there was a documented ‘slurp’ function (an API) that ingested data in a predictable way. It is noted that ‘ISPyB’ strictly speaking does nothing – ISPyB is a data storage program – but to users ‘ISPyB’ is the entire system including viewers and applications.

It might be hard within some sites even to track exactly how the data flow - frequently data seem to flow to the views without passing through ISPyB – and clarification might almost require archaeology in some cases. However nobody knows the situation well enough to estimate how much reworking might be needed. The problem is tied up with the site-internal pipelines – some programs might be mandatory in the pipelines solely because important data are harvested by grepping through their log files.

According to MG making and implementing an API is easy enough. The hard part is identifying the data you want, find out where it came from, and refactoring. AdM points out that an API close to the database tables makes it **extremely hard** to make database changes, since it means the table structure is coupled to the presentation code. What is required would be a high-level API. One problem is that XML and JSON are a bit too rigid to work well for an API. Another problem is that ISPyB is mission critical and has confidentiality and security requirements. An open and accessible API would be a security risk. The role of EDNA is raised, but EDNA is explained as ‘just a

wrapper', that copies data around, starts processing runs, and sets up Slurm parameters. Some EXI sites uses an alternative SOAP framework, not EDNA. Martin Savko points out that the data upload should remain site specific. ISPyB should have a clear API for uploading data, and each site could then decide how best to call it.

One idea for how to move ahead would be to make an inventory of all endpoints that are used in executing and viewing auto-processing results.

Full software stack for structural biology

The question is whether the collaboration could share both data model and tables, back end, front end and user interface technology. A few years back cooperation within ISPyB was almost impossible, since none of these were shared apart from the data base tables. A common software stack does not preclude having different look and feel to user interfaces, or supporting different features. EH has some reservations; scientists at DLS have very strong views on the look and feel and features they want in their applications. At a minimum there would be a requirement for a lot of flexibility and the ability to add extra modules. When pressed whetherf a collaboration on a full software stack might be possible, the answer of EH was that he would have to check with his stakeholders. People from other sites think that a shared stack should be possible - even different beamlines within a site have different interfaces while using the same stack. Several people spoke in favour of allowing for different views, selectable by the user or driven by data stored by individual applications – the MXPress workflow current stores JSON data that define their own view.

On the question of how to proceed, some people proposed to start by agreeing on the data model, others to start by implementing PyISPyB. Mael Gaonach (MG) suggested that the hard part is agreeing on what to do; the implementation is more straightforward. CV suggested decreeing a data model freeze immediately to work on introducing PyISPyB. AdM (for ESRF) had no problems with this, since the ESRF has been in a freeze for two years. At the moment all sites seem to be fairly close on terms of the data model. AdM commented that the key step would be to have people at each collaborating site actively working on PyISPyB. EH (for DLS) thought this might not be impossible, but again could make no commitments at this stage. It was suggested that it might be easier to move the common software stack forward among sites that were all using MXCuBE, and that maybe a core group could start the work without waiting for universal buy-in.