FriDAQ software plans for 2021

Antonín Květoň

DAQFEET 2021
10 March 2021, Remote

Contents

Our libraries

- DIALOG lib – IPC library
- Database lib – both common and application-specific SQL code
- Fsm lib (new) – state machine library, currently only used by the common client core
- Log lib (new) – interface to msglogger/browser from other applications. Currently not used anywhere

Our (major) libraries

- Registry lib (new) – used for representation of DAQ module registries. Replaced the old version of the library in our packages.
- Structure lib (new) – used for representation of the structure of the DAQ (modules and connections between them). Replaced the old code in some, but not all packages.

Our (major) packages

- Master – Single point for database access, management of process lifetimes and states, interpretation of control commands, error handling, aggregation and forwarding of slave state data...
- Slave control – One per DAQ module. Control and monitoring of modules using IPbus.
- Slave readout – One per readout machine. Spillbuffer readout, partial data consistency verification, transformation to DATE format, storage on disk.

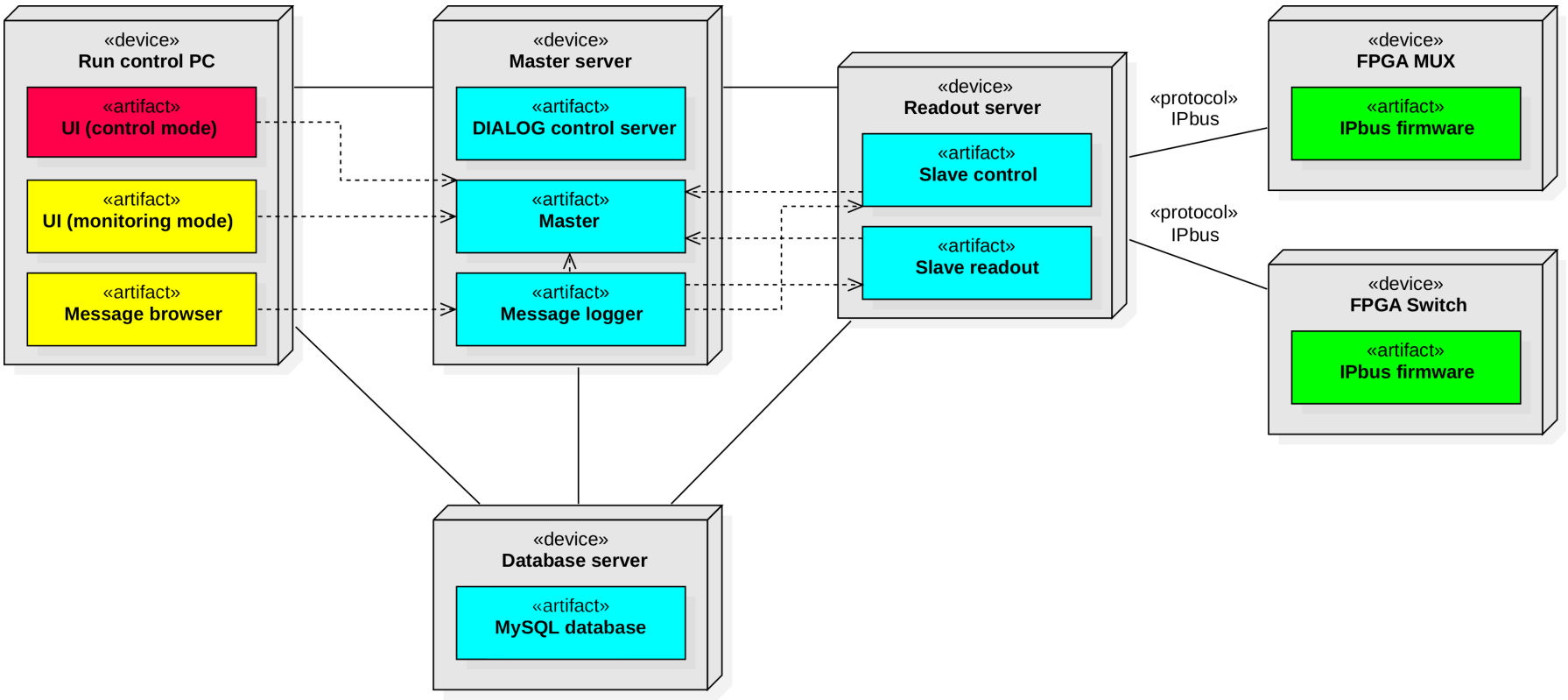
Our (major) packages

- Common client core – Complete representation of DAQ state using aforementioned libraries. Subscribes to Master's dialog service. Also allows for complete control of the DAQ.
- GUI – Control of the DAQ and visualization of its state. Based on the common client core.
- MessageLogger – Aggregation and storage of log messages from all processes.
- MessageBrowser – Browsing and filtering of log messages from all processes.

Our (major) packages

- Runlogger – Start of run functionality: logbook entries (run, beamline info, pt info...), bcoool, errorDumpAll
- Runmonitor – End of run functionality

DAQ deployment diagram



Version control split

- Our Gitlab repository is set up as a collection of submodules
- Each submodule (package) has its individual version control
- However, the packages often depend on one another, which could potentially make the transition very messy
- Solution: we branch-off at the repository level – two-level version control, aka a parent repository tracking the changes of its subrepositories

Slave Readout

- One of the oldest and least-touched packages in the whole system
- Core principles the same for AMBER, but most of the functionality wildly different (new data format, potentially calibrations)
- Easier to rewrite from scratch (using the new libraries) rather than adapt the existing one
- Work in progress

Master

- Functionality more or less same for both COMPASS and AMBER
- Most of the new libraries are however not integrated, and their architecture is so different from the current codebase, that again, it would be better to rewrite the whole package
- However, there are some blockers: new version of DIALOG library not yet fully functional, new version of database library required to eliminate some antipatterns in the new version of Master

Master

- Going through with the rewrite would eliminate some technical debt, some long-standing non-critical, but annoying, bugs and prevent accumulation of more technical debt in the future
- It is therefore a kind of "optional, but highly desirable" project for this year
- Whether it will be done this year will depend on the workload in general as well as the resolution of the blockers
- However, it should definitely be done in the following year.

Runlogger and runmonitor

- Some small changes are expected (especially w.r.t. detector-specific functionality, beamline condition logging..)
- However, there is a big question when it comes to monitoring (bcoool, errorDumpAll). These are launched by Runlogger and any work required will depend on how much the interfaces to these tools change (which has not been decided yet...)

Common client core/GUI

- Only small changes really required to make these work with the new DAQ (to adjust for differences between the registries in the old and new firmware of the DAQ modules)
- A large open question is whether we want to implement a decoding module into the common client core

Common client core – Decoding module

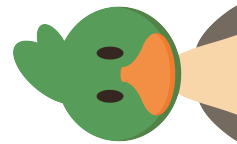
- This module would associate MurphyTV/coool-like decoding information with the structure of the DAQ in real time
- Usage primarily in detector expert programs, and would eventually potentially even allow of adding "MurphyTV functionality" into the main GUI
- Another use case: automatic safe stops/LOADs/restarts of run when a threshold is exceeded...
- Non-trivial amount of work, depends on how much we want this

Slave control

- Will work as-is
- However, a question arises if we decide to integrate the new frontends (controlled over IPbus) into the system. One process per frontend is not exactly viable. Would have to modify the package (and probably the database) such that one slave control process could control/monitor multiple modules.
- Also, the default registers are written every time the DAQ enters configured mode, which might not be necessarily desirable as far as the frontends are concerned.

Summary – open questions

- When to rewrite the Master process?
- Do we need a decoding module in the common client core?
- How do we control the new frontends?
- What will happen to coool/bcoool/errorDumpAll?



Thank you!