Application Framework Thoughts/Ideas

Fatih Bay, Eric Flumerfelt, *Wes Ketchum*, Roland Sipos

The Basics

We can agree on a set of basic needs

- Data interface layers: queues, buffers, etc.
  - Should use standard types/patterns underneath, with well-defined
  - We want to encourage these to be common/centralized as much as possible, but of course allowing for development of new ones as needed
- User-written Modules
  - Should be reasonable contained
  - We want to encourage use of common/good ones, but of course the goal is to encourage development of new ones as needed
- Service Utilities
  - Required: CommandFacility, ConfigurationFacility
  - Optional: Logging, Metrics, Dance Music, etc.
- System construction
  - Want to be able to easily build a DAQProcess as a graph, minimizing the number of ways to get things wrong while maintaining flexibility

Data Interfaces

- Think queues and buffers (we'll may use the name queue throughout)
- Are the "edges" in a graph
- Have some state (start initialized, but later get configured), but maybe have state limited to that
- Configuration
  - Configuration like size, eviction policy, etc.
  - Get configured before modules
    - They are independent of any modules
  - Order of configuration of queues is not guaranteed
    - All should be independent of each other
- Get names to be registered in the DAQProcess
  - Will match naming with configuration

Modules

- Code blocks that do a particular well-defined function
- Think the “nodes” in our graph
- Callbacks to commands
  - Modules will have the right methods for the FSM
    - Could be configurable in DAQProcess to make calls more asynchronous
  - Modules can also have custom commands executed (more later)
- Configuration
  - Modules will get configured after queues
  - Modules also get names in their registration with DAQProcess
    - Again, will be matched to configuration
- Transition orders should be specified where needed
  - Definitely(?) want for start and stop
  - Probably would want config to be able to be separate
  - Can have some rules/API for placing one before/after all others, etc.
  - Default is that there is no guarantee on ordering

Plugins

- Plugins: exchangeable implementations for defined interfaces (e.g. transport mechanisms, output interface, metric publishing, etc.)
- Plugins could be used in user modules and service utilities
- Plugins require fixed interfaces, but can then be configured at runtime
  - Will need to be careful then on defining those interfaces
- Plugin configuration would come with the user module/utility configuration

- Plugins don't have to be used: an alternative design would be specific modules:
  - E.g. DataRequestReceiverTCP and DataRequestReceiverZMQ and …
- But, for many modules, may benefit from simple plugins for flexibility
  - E.g. DataRequestReceiver with a plugin for where requests come from

Constructing a DAQProcess

- There will be a common DAQProcess
  - Takes basic config (on command-line) to initialize basic logging, command facility, and configuration facility
- Registers modules by loading a *ModuleList* class

- Initial implementation idea: ModuleList classes are user-written C++ classes implementing a *ConstructGraph* method that does the following
  - Defines queues to be used and registers their names with the DAQProcess
  - Defines user modules(which take queues as arguments) to be used and registers with the DAQProcess
  - Defines transition orders for modules

Examples

We've been jotting some psuedo-code here:

https://docs.google.com/document/d/1EgtEsrolwTxys1qENWwM9v7PYuA-QlMReO0Py9aUEG8/edit?usp=sharing

Can walk through some of that now

Possible errors

- At compile time, user-written ConstructGraph methods will automatically check types of queues are appropriate for modules
  - We want this
- There isn't at compile time any checking that all queues/modules are added to the maps to be properly registered, or that the transition orders are correct
  - Maybe we want this? But it's not super clear how to do it right now in a simple/clean way, so leave as an open item
  - DAQProcess should scream if it has a configuration for something it's not using → hopefully user will notice
  -

Transitions and Commands

CommandFacility does a few basic things

- Gets/parses incoming commands from some transport (plugin?)
- Registers the callbacks for state transition commands (to be executed by DAQProcess)
- Registers callbacks for custom commands to be executed by User Modules
  - User modules can configure callbacks with command facility in configuration
  - Custom commands would not ordered between user modules
    - Would prefer that this is either handled by who is issuing the command, or be part of the FSM
  - CommandFacility will have to figure out how to deal with user module commands of the same name/if that's allowed/etc.

I have now run out of time

So, more discussion!