Workshop Summary: DUNE DAQ Application Framework

Fatih Bay, Eric Flumerfelt, Wesley Ketchum, Roland Sipos, and input from many others

Introduction

Why a DUNE DAQ Application Framework?

Want a common structure for DUNE DAQ applications to ease and open up software development across the entire DAQ effort

A common framework will help provide common services, utilities, and interfaces to create *configurable* and *controllable* applications

> Standardizing interfaces to configuration and control will allow for easier and scalable deployment of DAQ applications

A common framework will also allow for modular development that can maximize reuse of good software components while allowing for development and testing of new components

Fermilab

How is this different from what we already have?

Desire a more comprehensive framework that specifically meets additional needs of DUNE DAQ development

- Ability to extend common configurability and control to not-traditionally-*artdaq* applications
- Ability to more easily develop/modify smaller functional pieces of data selection, data flow, and monitoring software
- Ability to extend more easily beyond *art* framework paradigms

Among other things, both *artdaq* and non-*artdaq* pieces of protoDUNE DAQ are key inputs

- An evolution of them to meet needs on protoDUNE-II and DUNE would be natural

“Requirements”

(in quotes because I don’t want to imply these are formalized requirements…)

A DUNE DAQ Application Framework should:

- Allow for modular development of functional blocks that can be written by DAQ developers and applied in a variety of contexts
    - E.g. data collection, selection, monitoring, and flow
- Provide standard interfaces for configuration, command/control, and monitoring utilities (including logging and “metrics”)
- Provide an application construction and configuration model that can both “scale up” (to work at the size of DUNE) and “scale down” (to work at the size of teststands)

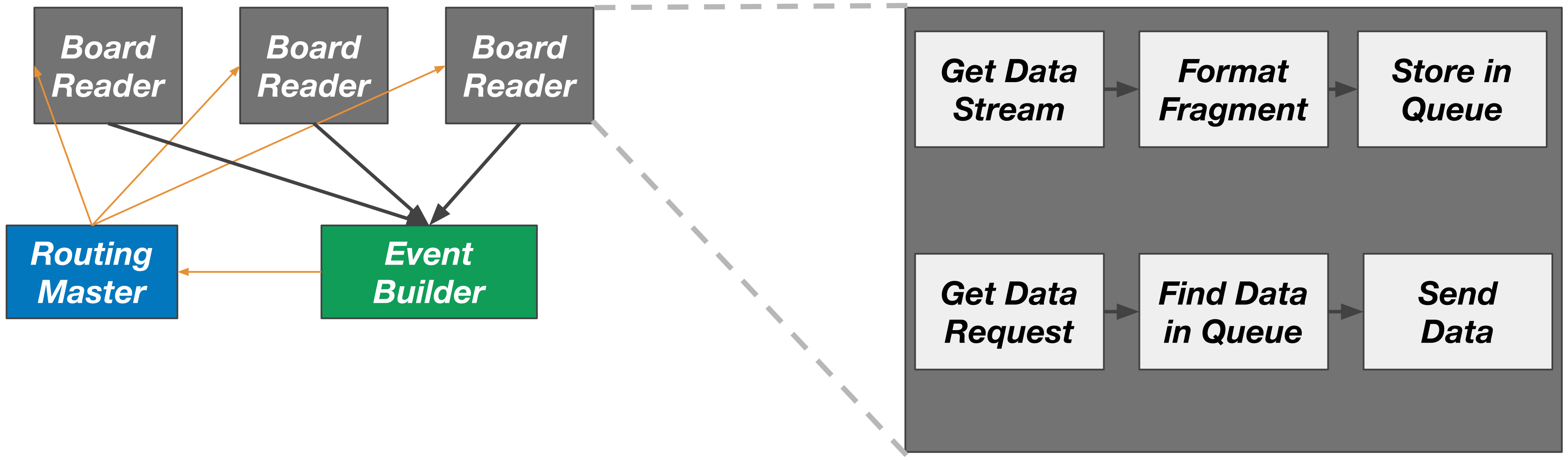
DAQ Application vs. DAQ System

A “DAQ System” is the total of DAQ processes running, usually across many nodes that is inter-connecting many data sources, data sinks, and selection/relay/routing information

A “DAQ Application” is the software for one of those DAQ processes, generally doing a limited set of functionality

There can be and we want some overlap of uses here

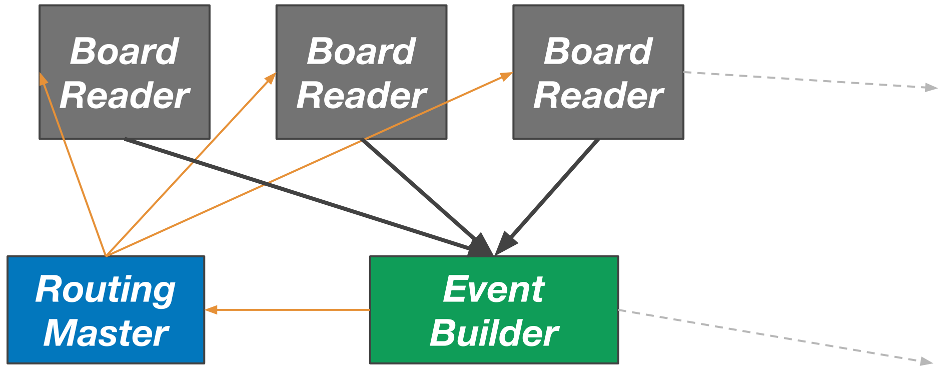
Application vs. System Example

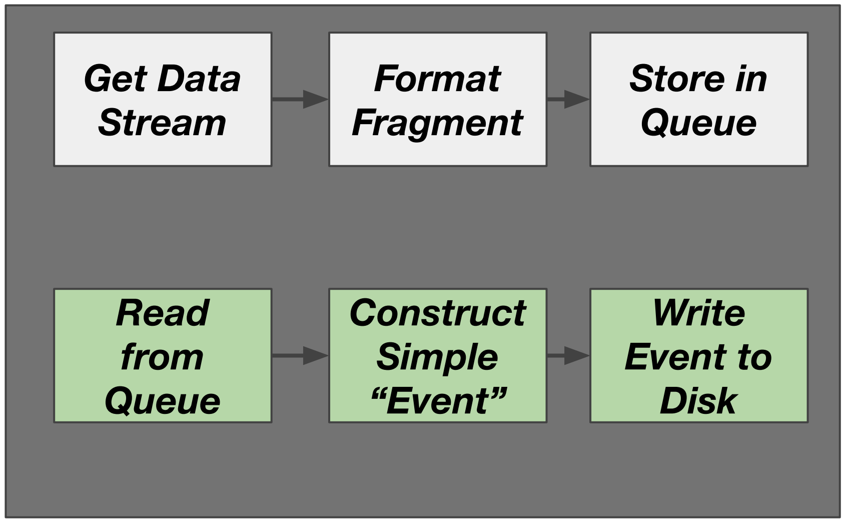


DAQ System

DAQ “BoardReader” Application

Or … another example...





DAQ System

DAQ "Teststand Driver" Application

List of applications

A list of possible applications was brainstormed at the workshop:

https://indico.fnal.gov/event/22384/session/17/material/minutes/minutes.html

A few hopefully obvious points

We need DAQ applications to work well in a variety of different online contexts

- DUNE application framework needs to have low overhead, be flexible, and coupled well with server infrastructure and process management

The effort to develop DUNE applications must be well beyond the scope of any one group

- Applications will exist/be developed across all groups
- Interfaces for data, configuration, management/control, and etc. will continue to be key points of common discussion and development

From the workshop

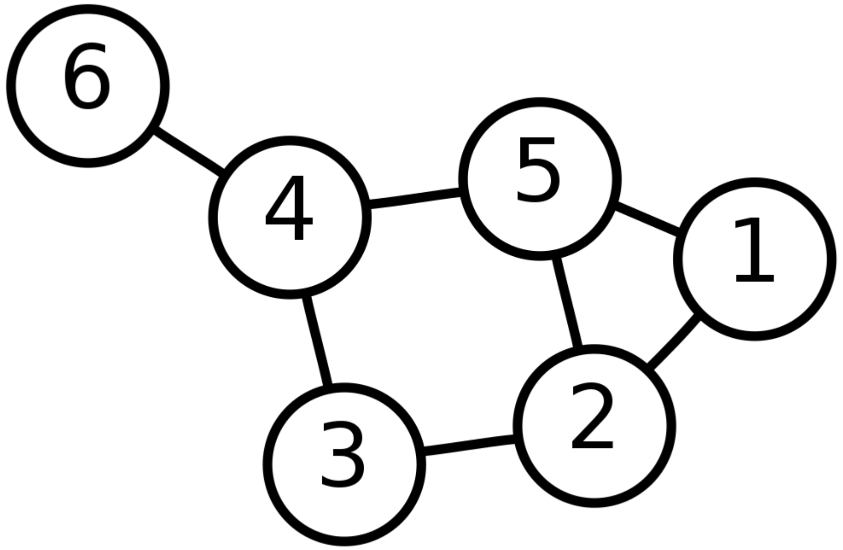
Key questions/discussions

- What are key components of DAQ applications?
  - *Modules*, queues/buffers, service utilities, plugins, etc.
- What properties will those components have?
  - FSM, configurability, etc.
- How will applications be constructed/configured from those components?
  - Module Manager?

General paradigm

In general, we recognize that this is type of graph

- *Nodes* composed of custom-written (C++) modules that execute some core functionality
- *Edges* link the two together, generally through communication of data payloads over some buffer



https://en.wikipedia.org/wiki/Graph_theory

Fermilab

DAQ process essentials

A common DAQ process class should:

- Setup required service facilities for configuration and command
  - (Also some basic minimum of logging)
- Setup any additional optional service utilities
  - E.g. metric monitoring, detailed logging, party lighting, etc.
- Register desired modules/buffers
- Manage transitions through a common FSM

User module essentials

User modules are code blocks that do a particular well-defined function

- Development
  - Expect some modules may be written once, written well, and simply maintained (e.g. searching/pulling data by timestamp off of a queue?)
  - Other modules will be modified or replaced often (e.g. data selection algorithms)
- All modules will have callbacks to commands for the common FSM
  - If not needed, choose to not use
  - Command ordering/asynchronicity should be configurable, though ideally largely each module is standalone
- All modules will have access to configuration
  - Will need some registry of module name/attributes to aid configuration utilities

Queues/Buffers essentials

Perhaps they are just special modules, but data interface layers play important role in DAQ applications

- Interface layers between more functional user modules
- Need to often be highly optimized for performance and reliability
  - Likely can be templated classes, but connections between user modules may demand they be strongly typed

Few important attributes

- Should be configurable for size, eviction policy, etc.
- Should be completely independent of all other modules
- Need to be properly configured and connected to user modules

Fermilab

Plugin essentials

- 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
- But, for many modules, may benefit from simple plugins for flexibility/better design

Transitions and Commands

A command facility should do 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.

To ModuleManage, or not to ModuleManage

We discussed *a lot* about the need for a specific “ModuleManager” class/utility, and what it would/should/could do

Basic things I think we can all agree on:

- There must be something that “constructs the graph” in a usable way:
  - (1) instantiates the modules/buffers/etc.
  - (2) handles the proper configuration of topology and both explicit and implicit interaction between modules
- The handling of ensuring “proper configuration” is not trivial, is tied to configuration tools, *and requires good documentation of module interfaces*

It will take time to develop higher-level configuration tools that meet these needs

First step forward

At the workshop, we tried to iron out a first path forward that allows for practical development

- Explicitly defined (C++ classes) of *ModuleLists* that would declare **all user module and buffer components of an application and their interfaces**
    - Compiled code → helps ensure proper typing/interfaces
- ModuleLists would currently be “developer written” and put load of proper config/declaration on application writer
    - If this is a heavy/unscalable load, we will definitely need a common *ModuleManager* tool
- Got a long way towards writing out some pseudo-code use cases
    - ***Which really helped lay out practically useful things*** (declaration of order of ‘start’ command) and perhaps some anti-patterns

Pseudo-code examples

I’ll likely regret this, but if you really want you can see some of the working notes/pseudo-code from the meeting here:

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

Next steps

(Caveat: maybe heavily Wes’s opinions)

Long-term elements that we should work on now

It's clear (at least to me) that many of the elements of a DAQ application framework are ready to be worked on and implemented in a first iteration

- Implementation of common/standard queues and buffers
  - Likely built off of existing standard C++ elements, but with added utilities to our use cases
- Evolution of some standard user modules from elements we already have
- A basic backbone DAQProcess class with basic configuration and command facilities, and the associated 'main' to go with it
- Standardized interfaces (and plugins?) for basic elements:
  - Config, command, logging, metrics, etc.

Investigations to pursue now (in parallel)

Higher-level configuration utilities and a ModuleManager would benefit from really trying to implement with a real case

- E.g. a graphical ApplicationBuilder interface that interacts with a ModuleManager
- Automated instantiation of buffers/other components?

A logical next step may be to have such a utility “stamp-out” a C++ *ModuleList* class, that could then be compiled in and used in a DAQ system

Some Important decisions to start making then

Proper interfaces can avoid tying ourselves to something for forever, but we do need to make some practical decisions

- Configuration language to use for now
  - I think it's FHiCL until we say it's not FHiCL?
- States in the FSM
  - I think it's something like Booted/Configured/(Started=?)Running/(Paused?)/Stopped(=Configured?)
- ***Documentation standards***

Some questions on organizing future effort

How should we organize this future effort?

- As said before, this cuts across all groups, so doesn't seem to naturally fit in one
- It critically will need the inputs from everyone
  - Framework developers
  - Application developers
    - Savvy coders, but also power and non-power users
  - CCM interfaces

What are the near- and long-term milestones?