DUNE DAQ Software Suite
application framework overview

Marco Roda
Data Selection meeting
04 August 2020

UNIVERSITY OF LIVERPOOL

Introduction

- There are already a number of talks and documentations about the AF
  - I’m trying to not be repetitive and give an implementation oriented prospective
- Talk overview
  - Recap of the status of the AF
    - interfaces
    - future plans
    - a look at the big picture
  - I suggest some operative ideas on how to implement your ideas
    - Note that I **don’t** know your ideas

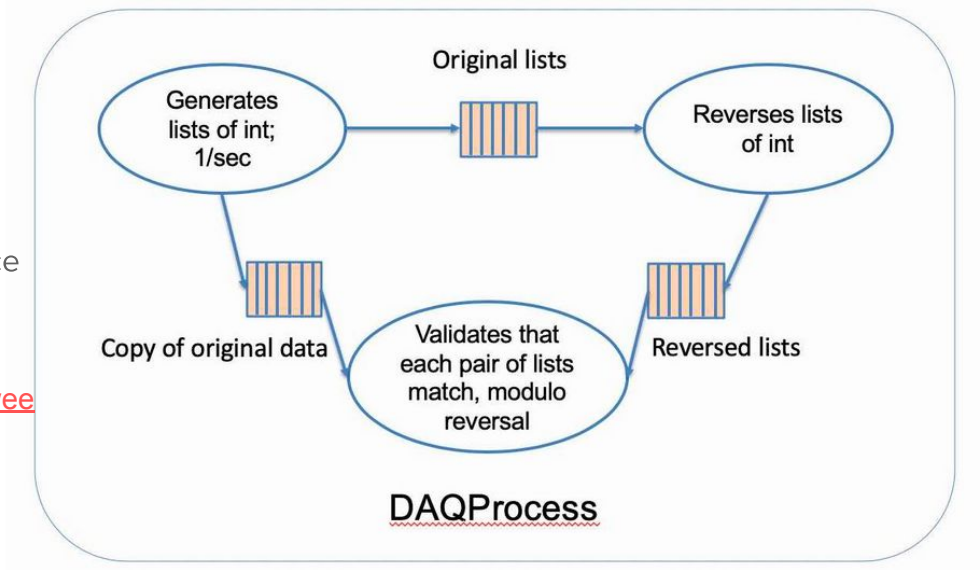
General Introduction

DAQ Software Suite Intro - The big picture

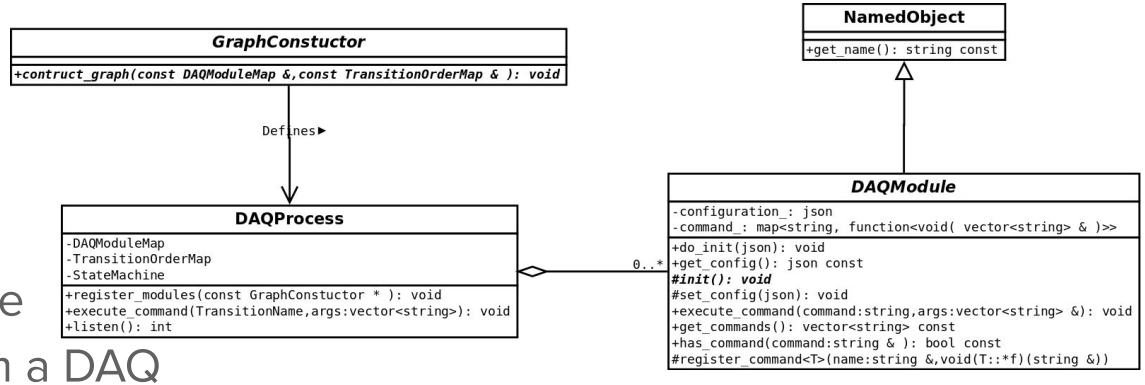
- A number of lessons learned from ProtoDUNE
  - https://docs.dunescience.org/cgi-bin/private/ShowDocument?docid=11737
    - Not complete nor definitive as the document was written at the end of the test beam
  - A number of items were related to software
- Different working groups will be creating packages compatible with the AF
  - The DAQ Suite will include these repositories
  - plus a number of tools
  - for the time being the main reposotories are https://github.com/DUNE-DAQ

Application framework idea

- Modular structure
  - DAQ Module
  - “Intra-Module Connections done with Queues
- All wrapped up inside a DAQProcess
  - It will propagate the commands from an interface (CCM) to the modules
- All details here
  https://github.com/DUNE-DAQ/appfwk/wiki/Interfaces-between-DAQ-objects



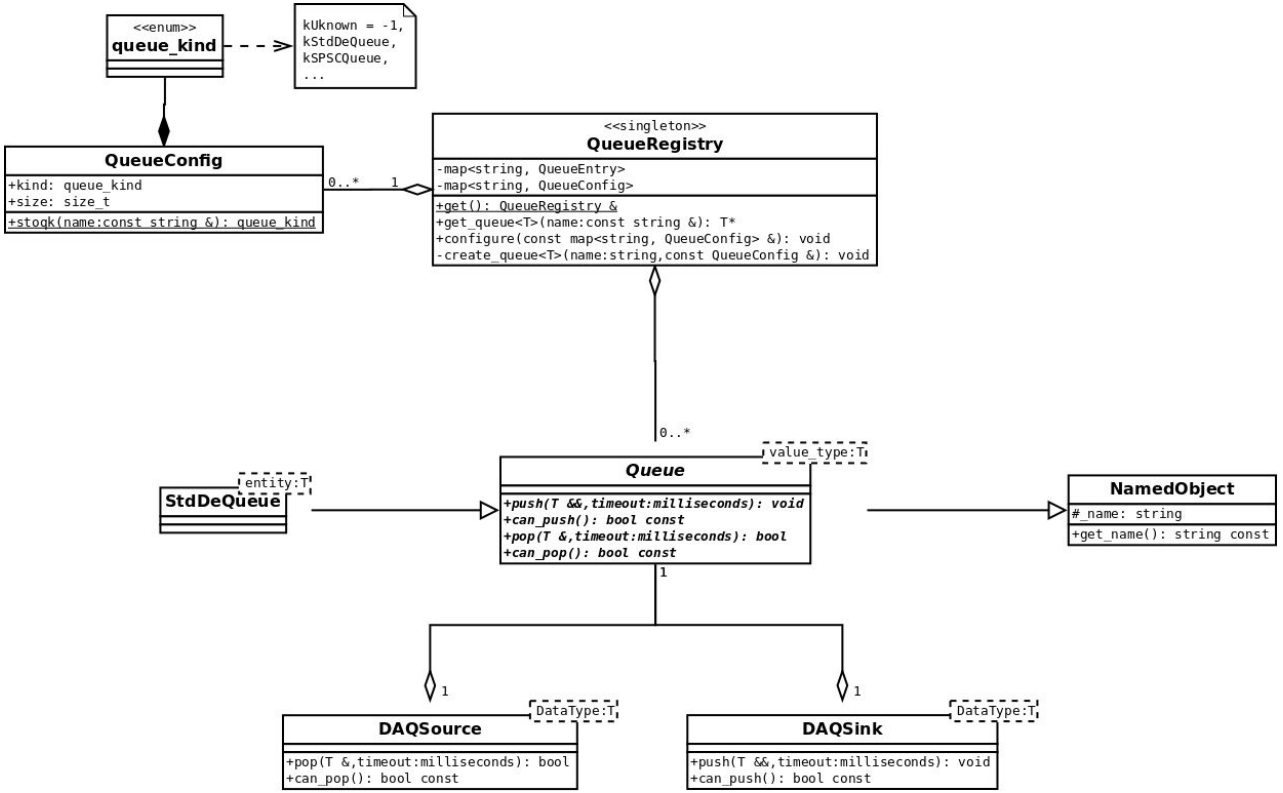
Key Components - DAQModule



- The main insertion point for user-written code
- One or more DAQModules are connected by Queues to form a DAQ Application
  - Each DAQModule should be self-contained and focused on a single task
    - “Self-contained” means that it should not rely on the presence of any other DAQModules in the same application
    - “Single task” can be as complex as necessary to achieve desired functionality

Examples: Reading data from a Felix card, Sending data to another DAQ Application, Performing Trigger Candidate selection

Key Components - Queue Class Diagrams



Key Components - Queue

- Simple templated connectors between DAQModules, exposing a pop/push interface through the DAQSource and DAQSink helper classes
- Implementations are provided by the DAQ Framework
- DAQModules should not rely on a specific Queue implementation
- Initially, provided implementations is a std::deque-based and a Folly-library-based one
- DAQModules use DAQSources and DAQSinks for input and output to other DAQModules
    - Any number of DAQSources and DAQSinks may be constructed by any given DAQModule, the target Queue is identified by name in the DAQSource and DAQSink constructor

Application framework scope

- Provide common structure for DAQ Applications
  - Graph model for data flow within application
  - CCM-defined interfaces for control, logging, and metrics
    - DAQ Processs (== 1 application) is atomic unit from CCM perspective
  - User-facing interfaces are kept minimal for flexibility
    - DAQModule is primary user interface and it requires only one method to be implemented
- Support dynamic creation of modular applications
  - Add and remove application components via configuration
- Encourage the creation of toolkit repositories that contain utility DAQModules and libraries for performing common tasks

Status - version 1 release

- https://github.com/DUNE-DAQ/appfwk
- appfwk v1.0 has DAQModules and Queues
  - Users can write their own DAQModules and combine them together
  - configuration files to specify the graph
- appfwk v1.0 does not have inter-process communication
  - Goal is to have this available in the next release of the DAQ SW Suite
  - Command-line control is sufficient for configure/start/stop ‘transitions’
  - Command handling and state, re-configuration, and other aspects are either rudimentary or not yet available
- Instructions and examples on how to create new packages are available
  - https://github.com/DUNE-DAQ/appfwk/wiki
  - We expect that folks will start with DAQModules in their own Github repo(s) and move that code to the DUNE-DAQ project if/when they’re ready for that.
  - Feedback collection

Short term plan for the application framework

- Developer(s) from all WGs work through the examples and create their own DAQModule(s)
  - Check if the present design has limitations that limit operations
- Working groups identify volunteers to gather local feedback
- Data Flow WG hosts a discussion – answer questions, gather overall feedback
  - Targeting second week of August
- Submitting feedbacks directly to GitHub
  - opening an Issue

Medium-term DFWG plan for the DAQ Suite

- Some planned DFWG milestones:
  - (end of) September 2020
    - First availability of inter-process messaging infrastructure
    - Sample app with teststand-like readout & local write capability
  - (end of) December 2020
    - Simplistic DAQ vertical slice system

- Of course, plans can change and schedules can slip…

Design ideas

How to get to an effective design?

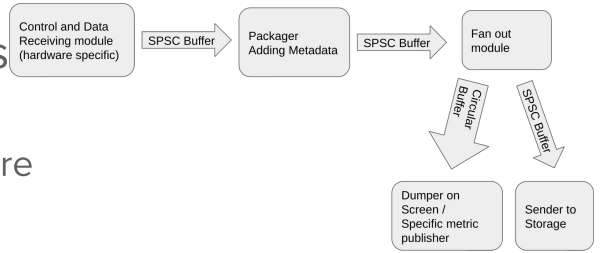
- The technicality of the implementations are described in the wiki
  - Description of the interfaces
  - listrev example with step-by-step instructions
- … But clearly your tasks are much more complicated than *listrev*
  - I'm going to give some ideas on how to produce a good design
  - Meaning of "good"
    - quickly identify problems with infrastructure design that requires feedbacks to the AF
    - maximise code reusability
    - good "sync" with the rest of the DAQ Suite
  - Note that I don't know your ideas
    - These are general suggestions you might find in any software engineering textbook
- General rule
  - write code only once the design is complete

Top down approach

- In this case the **only** thing which is fixed is the high level structure
  - That is the application framework
- You don’t get to decide what are the interfaces you are supposed to use
  - So start from what is fixed and work your way through the development from there
  - It does not mean the interfaces are immutable
    - But first try and see if they are enough

1) Identifying the DAQModules you need to develop

- I’m assuming you know which application you need to develop
  - I’m suggesting how to develop those applications
- The first step is being able to produce these diagrams
  - Check common code between applications
  - immediately checks if AF is enough or you need something more
    - In that case Feedback to us is necessary
- A few ideas to break down your applications in modules
  - Start writing flowchart of the operations for each applications
    - operation Loops are good candidates for a DAQModule
    - Data transmission operations are good candidates
      - Defer the implementation of these modules as they might be developed as toolkits
      - Feedbacks on how you intend to communicate or transmit data are very valuable



2) Internal tools

- you are expected to develop an entire (git) repository: be **creative**
  - not only DAQModules
  - Identify tools that are specific for your operations and model them in classes and object
    - In a flow chart scenario class candidates are the single blocks
    - Even writing down a small text that describes the operation helps
      - classes ↔names
      - functions ↔verbs
  - Identifying the relationships between the object is also important
- Operatively the possible output of this stage is a UML diagram

Missing pieces

- Of course there will be missing pieces
  - I suspect that big problem for DS will be the Hardware related software
  - How to fill the gaps?
    - writing (short) proposals about what information should be passed and how
      - circulate with the relevant groups
    - Contact people directly
    - have liaisons between the working groups
- This is already happening at the level of the AF
  - In fact Kurt suggested Liaisons for the AF at a previous SC meeting

Conclusions

Conclusions

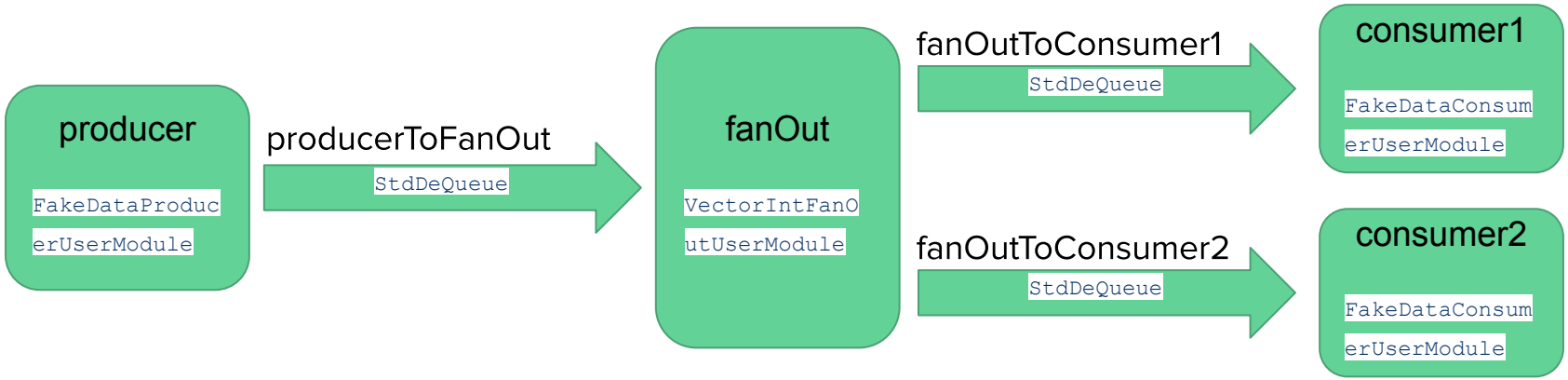
- The first brick of a DUNE DAQ Suite is complete and deployed
  - documentation and examples
- We are gathering feedbacks
  - From the WG that should try to develop their own DAQ Modules
- Next steps
  - Toolkit development
  - DAQ Suite release
- Point of contacts:
  - me and Eric Flumerfelt - AppFmwk issues
  - Phil and Kurt for more general issues and global feedback collection

Backup slides

Key Components - Style Guide

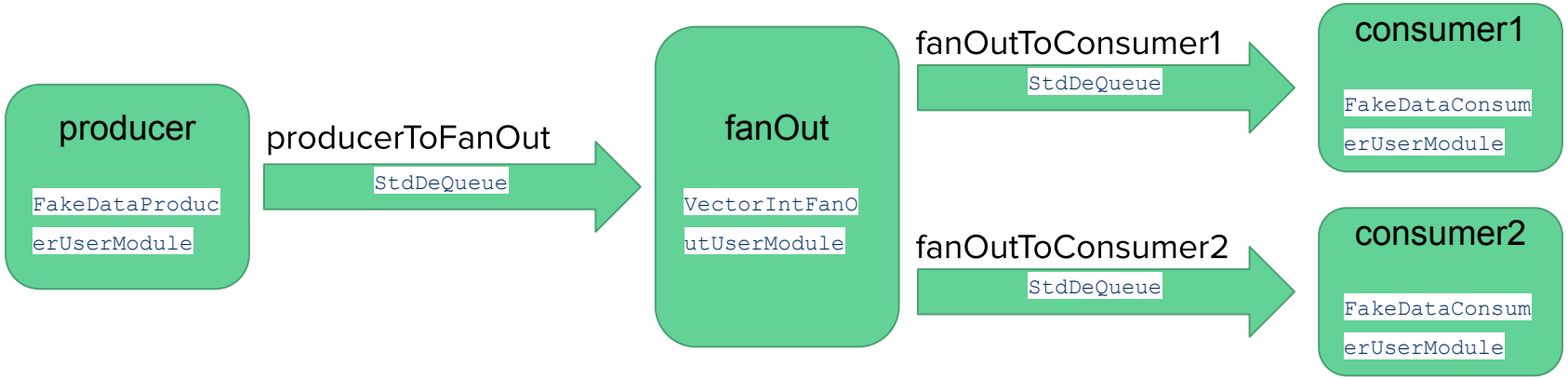
- As DUNE is a large and long-lived experiment, enforcing consistent coding style is important for maintainability
- The Software Coordination group has produced a style guide as well as linting tools
- Style should be checked for every Github pull request
- https://github.com/DUNE-DAQ/styleguide/blob/develop/dune-daq-cppguide.md
- User code is expected to conform to the DUNE style, please read through the rules and let the Software Coordination team know if there are any you can't live with

Example Application



- One configuration file for daq_application is provided in the initial appfwk release package
- producer_consumer_dynamic_test.json loads a “FakeDataProducerDAQModule”, a “VectorIntFanOutDAQModule”, and two “FakeDataConsumerDAQModules”

Example Application



- “FakeDataProducerDAQModule” creates fixed-length vectors of integers, with a specified length, starting integer, and ending integer (subsequent vectors start where the previous left off, wrapping around when they reach the end int)
- “FakeDataConsumerDAQModule” validates received vectors of ints, checking their length and whether they start with the expected integer

Example Application - Config

```
"queues": {
    "producerToFanOut":  { "size": 10, "kind": "StdDeQueue" },
    "fanOutToConsumer1": { "size": 5,  "kind": "StdDeQueue" },
    "fanOutToConsumer2": { "size": 5,  "kind": "StdDeQueue" }
  },
  "modules": {
    "producer":  { "user_module_type": "FakeDataProducerDAQModule", "output": "producerToFanOut" },
    "fanOut":    { "user_module_type": "VectorIntFanOutDAQModule", "input": “producerToFanOut",
                  "outputs": [ "fanOutToConsumer1", "fanOutToConsumer2" ],
                  "fanout_mode": "RoundRobin" },
    "consumer1": { "user_module_type": "FakeDataConsumerDAQModule", "input": "fanOutToConsumer1" },
    "consumer2": { "user_module_type": "FakeDataConsumerDAQModule", "input": "fanOutToConsumer2" }
  },
  "commands": {
    "start": [ "consumer1", "consumer2", "fanOut", "producer" ],
    "stop":  [ "producer" ]
  }
```

Note that command ordering should be used sparingly within an application; it is shown here as part of the example