Online Software Inter-process Communication

Brett Viren

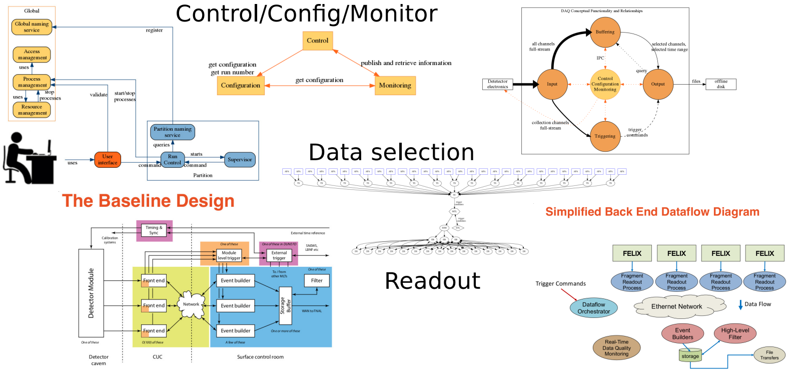
Physics Department

BROOKHAVEN NATIONAL LABORATORY

DUNE DAQ WS – 4 Feb 2019

Outline

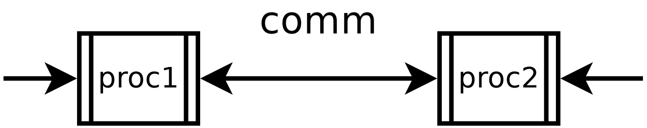
Current DAQ Subsystem Concepts



- Plus: CCM; Online Tools; Infrastructure and Integration

- Many "boxes" (applications) needed, but also many "arrows" (IPC).
- Implementing arrows takes comparable effort boxes.
- Many shared requirements for the arrows.
- We should avoid creating a "babel" of IPC implementations.

Inter-Process Communication (IPC)



processes	tightly coupled functionality bound into an application, operating "locally" on input message, producing output messages.
communication	asynchronous exchange of units of data obeying well defined message schema and protocol contracts.

- Bulk of a process is the **application logic**, but a sliver of each must deal with the communication.
- Two independent applications must share a common communication, they can't each invent something.

Established and Missing IPC

+ artDAQ has a well developed IPC
+ whatever RDBMS we use (eg, PostgreSQL) has its IPC
+ "OS-level" Apache/Ganglia/Superivord/Puppet/etc

? Internal comms between Run Control / CCM applications?
? RC to basically everything else?
? Trigger / Data Selection internals?
? Inter-module trigger exchange?
? Trigger command hand-off to Back-End?
? Front-end buffer data query/response?
? Monitoring, logging from essentially everything.
? extra-DAQ interfacing (eg CISC, det h/w)
? what do I miss?

Proposal

Develop a single, common IPC covering the missing IPC.

Require that DAQ applications:

- Use it for all **inter**-subsystem IPC.
- Use it for any **intra**-subystem IPC **where it's needed**.

Some Requirements on common IPC system

- Satisfy latency and throughput needs (inter– and intra–subsys).
- Support formally defined message schema and protocols with versioning.
- Provide mechanisms for redundancy, discovery and presence.
- API for application development.
- Portable and with minimal and stable dependencies.

Scope and implementation of the Common IPC

? artDAQ is one player in this space and we must leverage it, not replace it. Does it make sense for artDAQ to become the basis for all subsys?

? NetIO is used at protoDUNE/FELIX for transfer of full data from FE to the two BE computers. DUNE FD won't do this but should NetIO also be considered as a basis?

- I don't know these answers with 100% certainty, but I feel the common IPC should be built from a more general base.
- I'll describe what I think is right for implementing a common IPC system but think of it as a **proposal of work**.
- If more or less accepted, I'd like to lead the effort to develop it and I'll need some help.

! Discussion and buy-in by all subsystems are needed to determine a proper scope and design for a common IPC.

Common IPC System Scope

- Methods to formally define message schema and code generation for functions to operate messages
  - eg, to form/parse/`send()`/`recv()`
- A set of C++ shared libraries providing IPC functionality for DAQ application development.
- A C++ API which supports layered entry points to match the diversity of application development
  - eg layers: function primitives → toolkit → application framework
- Proper factorization of concerns
  - general purpose core libraries.
  - DUNE DAQ semantics as extension libraries.
- Python API bindings.

Next I give an outline of the Common IPC System based on prototype software I've been developing at

`https://github.com/brettviren/digrex`

The most recent and promising line is in `dexnet/`

I take `dexnet` as working title for now.

dexnet Foundational Dependencies

ZeroMQ	provides high-performance networking abstractions.
Protobuf	candidate support for structured message data.
nlohmann/json	modern C++ JSON support, another candidate for handling structured message data.
Jsonnet	powerful, simple human-oriented configuration language.
Boost	for various C++ supports, including Boost.SML state machine.
Waf	for portable build system.

ZeroMQ

ZeroMQ	high performance, high level `socket(2)` abstractions, variety of communication patterns, reliable, robust against connectivity loss and congestion, support multiple types of transport.
CZMQ	High-level C bindings to ZeroMQ, behavior patterns: **actor**, poller, looper, auth/auth.
Zyre	A CZMQ actor providing network **discovery** and **presence**.

Comments:

- All are C libraries and usable in C++, minimal dependencies, very portable,
- Free Software (LGPL), long lived, strong and helpful development community.

ZeroMQ Transports

Three types message transports supported:

1. `inproc://thread-label` between **threads** in the same process
2. `ipc:///path/to/pipe` between **processes** on the same computer
3. `tcp://hostname.dune.daq:1234` over the **network**

Application is **independent of the transport**.

- Transport is **run-time configuration** with strings like the above.
- We can develop DAQ applications which **scale** from threads to processes to computers with no code changes. Clear benefit for DUNE DAQ.

Sample of ZeroMQ socket patterns

PUB/SUB	subscribe to topic, receive published messages, topic filtering on PUB side (don't send unwanted messages). After subscribe, async, one-way streaming. (drops)
PUSH/PULL	unidirectional, "FIFO". (blocks)
REQ/REP	classic client/server, synchronous. (blocks)
ROUTER/DEALER	advanced extension to REP/REQ. (blocks)
PAIR	Exclusive peer connection, two-way async "pipe". (blocks)

- Except for PAIR, M-to-N connectivity pattern may be used as good redundancy mechanism.
- The "(blocks)/(drops)" indicates what happens if the socket maintained internal buffer reaches its adjustable high-water mark.
- There are additional socket types available, these are most common.

CZMQ Actor

A **function** running in a **thread** communicating
via a thread-safe, bidirectional **pipe** (PAIR socket).

`dexnet` uses the *actor* pattern as a basic building block for its mid-level application API. More on this in a bit.

Zyre Actor

An actor which implements network **discovery** and **presenece**

(as well as a simple form of many-to-many "group chat" like communication).

Discovery:

- Send UDP broadcast **beacon** containing:
  - → **identifier** (a name), an **endpoint** string and set of **key/value** pairs.
- Listen on endpoint, respond to any peer beacons by sending above info to peer's endpoint.
- The time scale for discovery is 1-2 seconds.

Presence:

- Peer heartbeat messages at configurable interval.
  - heartbeats use both UDP and TCP (efficient vs reliable)
- Timeout interpreted as a loss of presence, Zyre actor notifies the rest of the application so that it may react.

API Layers of `dexnet`

An application may be developed leveraging as little or as much of `dexnet` as desired. In order of increasing abstraction:

1. No `dexnet`, application uses raw `socket(2)` and fully handles responsible for proper message forming/parsing.
2. Replace raw `socket(2)` with ZeroMQ sockets.
3. Use `dexnet` message schema codegen to provide form/parse code.
4. Use `dexnet` graph abstractions (described next).
5. Bundles those abstractions in form of *actor* but manages its lifetime.
6. Provides one or more *actors* in a `dexnet` plugin library and use `dexnet-agent` command line application for fully configuration-driven application aggregation.

`dexnet` Ported Graph Abstraction

node	a **procedure** operating asynchronously from others.
port	a **labeled point** of a node through which data flows.
edge	a **connection between two ports** which transmits **messages** following one or more **protocols**
message	a typed **unit of data** transmitted along an edge
protocol	a set of **message types** and **expected behavior** of their transmission.

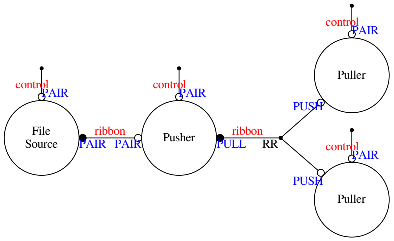
Note: any connected subgraph of a ported graph may be recast into a single ported node. This has strong implications on complexity management.

A *node* may be a fundamental *actor*, an *actor* which manages other *actors* (an "agent"), a logical subgraph, or an application that merely provides or emulates *ports*.

Actor Example

Four actors work together to process data

- The "actor pipes" (PAIR sockets) speak "control" protocol sending RC commands to the actors.
- Actors create additional sockets and `bind/connect` to endpoint addresses based on "control" protocol messages and initial configuration.
- The "ribbon" protocol is the exchange of working data.



- The "control" protocol connects to RC directly, or via intervening application code.
- Actors may be in individual applications communicating across network or aggregated into one and communicating across local thread-safe queues.

`dexnet` Development Effort

- A lot of prototyping exists already so many problems worked out.
- My intention (if the consortium is behind it) is:
  - Make `dexnet` a higher priority (it's mostly been a "weekend project" so far).
  - Complete remaining prototype issues.
  - Move the code into production quality code base.
  - Along the way, respond to input from DAQ application developers.
- I think I need a minimum of **two or three part-time individuals** to closely collaborate on **design, development and testing**.
- It would be best if these individuals were using their other part-time to develop the "business logic" of actual DAQ applications.

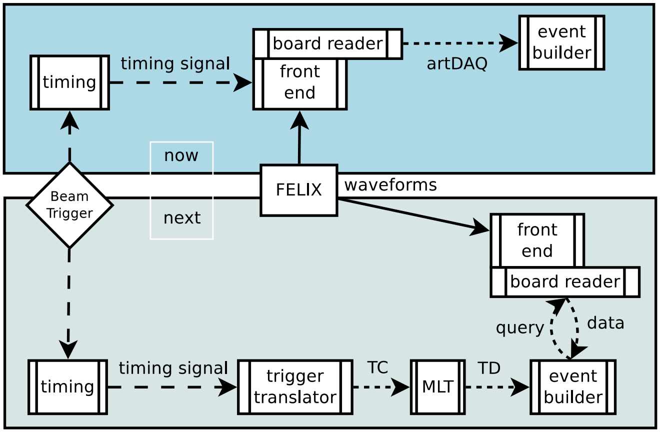
Timeline

- I am a bad judge of time.
- But, I feel the effort for `dexnet` + applications is comparable to what was needed for the Wire-Cell Toolkit
  - That is now in its ≈3rd year.
  - WCT had off-again/on-again effort, so "3 years" should be in quotes.
  - Much of the core development of WCT has been in place a while and it's mostly been new "application" ("algorithm") work lately.
- So, I think, given the help, the `dexnet` core might take 6 months to a year to be very usable.
- Full "business logic" applications need their own timeline estimation but it should progress in parallel with `dexnet` development.

ProtoDUNE Demonstration

Paraphrasing Dave Newbold:

> *Splice in a software trigger into current protoDUNE timing system based trigger to test stuff.*



dash: hard wired
dots: network
solid: full data

ProtoDUNE Demonstration Caveats

- We can (and will) test `dexnet` without protoDUNE.
- Writing protoDUNE "adapters" will take real work.
- We need to understand what collective/emergent features/problems such a demonstration would test in exchange for the effort.

$\mathcal{FIN}$