DAQProcess/Module network layer changes

- a report on work-in-progress

Kurt Biery

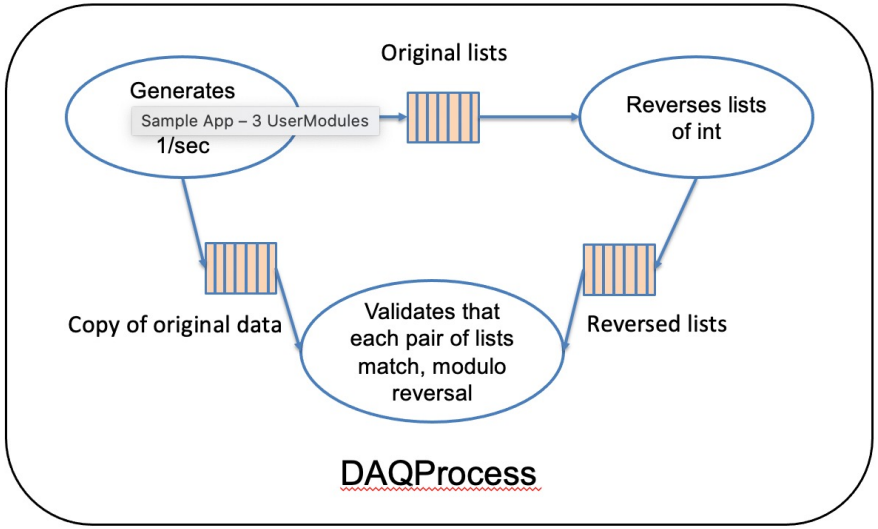
22 Sep 2021

DAQ Parallel Session, Collaboration Meeting

Introduction

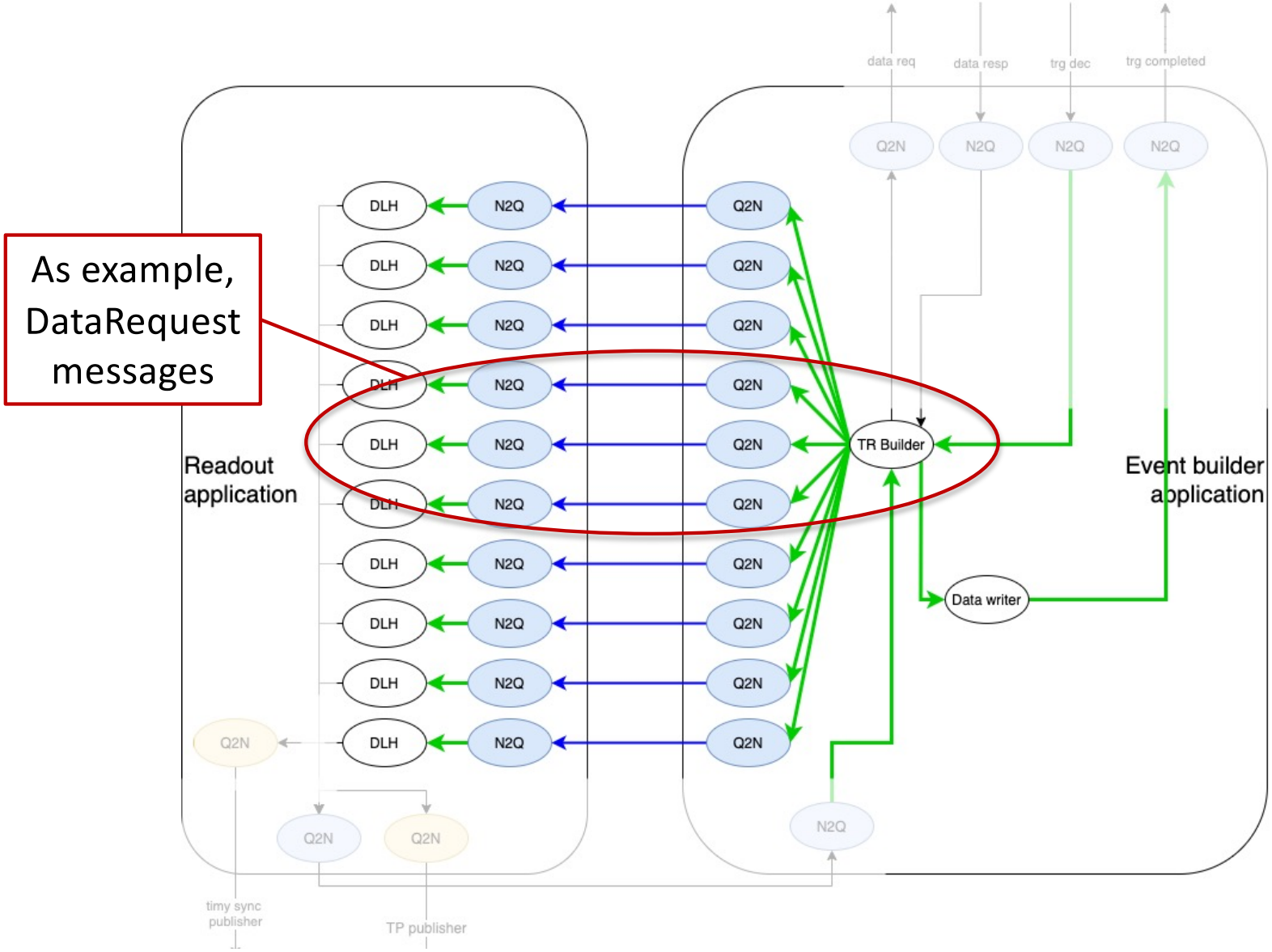
Changes are being investigated/planned in the communication between DAQModules (DAQProcesses).

Recall that in releases <= v2.8.0, most modules only knew about Queues; inter-process (socket) communication was handled by special-purpose modules that took care of serializing messages and sending them over the network.



For reference, a diagram that shows the *listrev* demo.

v2.8.0 and earlier…



Reasons for change…

Reasons for updating the model for inter-module communication include the following:

- Lots of QueueToNetwork/NetworkToQueue module pairs adds complication to the configuration (and uses resources at runtime)
  - the generation of those config snippets could be automated, but still…
- The Q2N/N2Q model doesn't have a way to alert the user modules of errors
- The intermediate Q2N/N2Q modules make clean Start and Stop transitions incrementally more difficult
- It's hard to imagine how graceful failover might happen if we need to destroy/construct Q2N/N2Q pairs as part of that

Desirable features for an updated model

As little as possible information needs to be known statically. For the request/reply part: the entity that receives a request only becomes aware of a requester when a request comes in. It doesn't need to know anything about who potential requesters are upfront. Very valuable for many recovery scenarios.

We limit the socket connections to a reasonable number, in some cases focusing on applications and not the underlying building blocks. One area where this may be valuable is within readout applications. We expect that it will be valuable for event builders to *not* know all of the details of the sub-structure within readout apps.

Logical naming of endpoints/ports

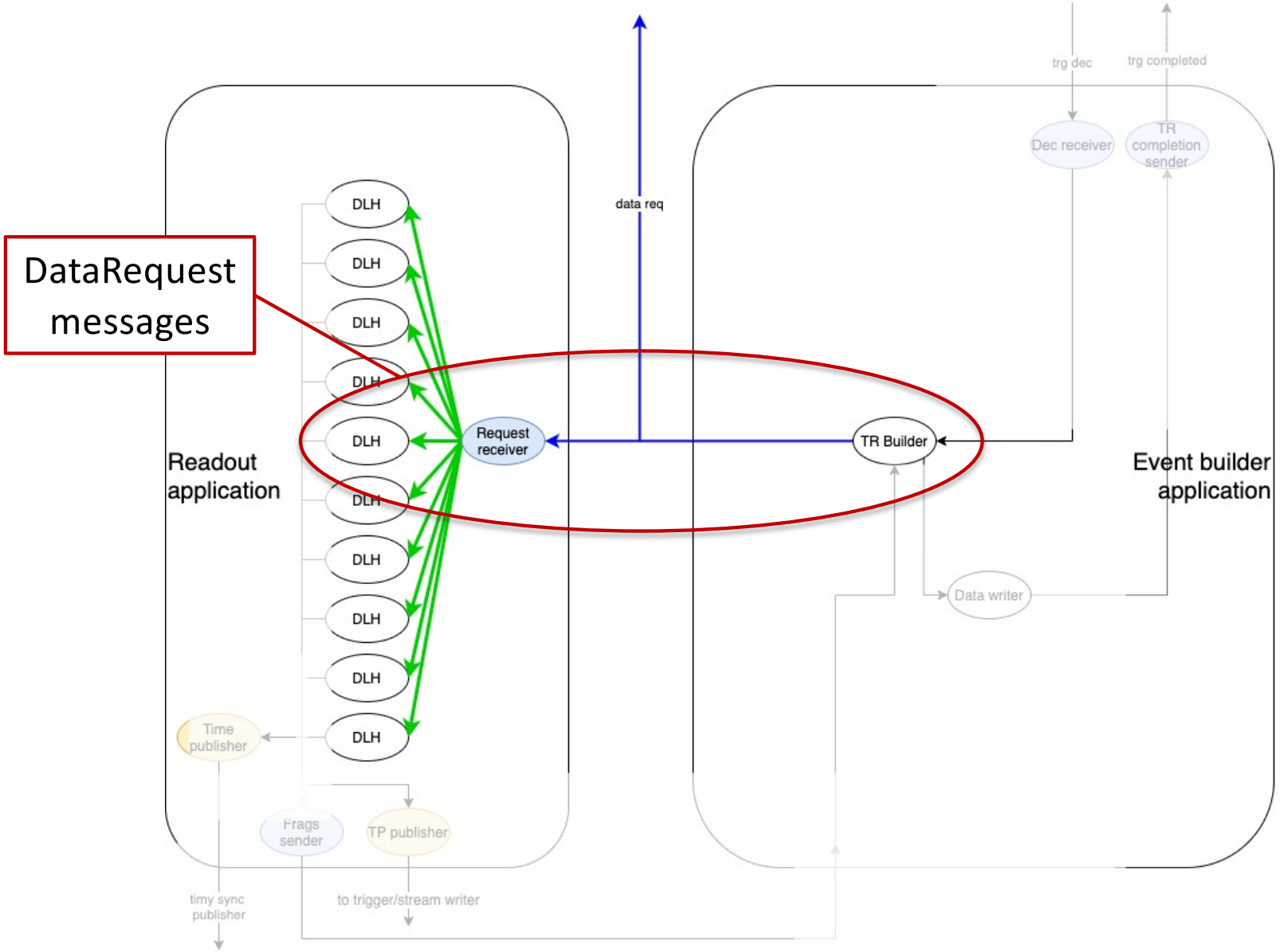
- A proposal: <partition identifier>:<communication channel type>:<destination identifier>, eg. Part3:DataRequest:RU-55

Lists of questions to be answered

Examples from Phil:

- How do we handle errors? Which modules get notified?
- What to do if downstream is blocked?
- Can we drop messages, and if so, whether/how to signal that to the sender/receiver?
- How to get a clean stop?

Sample investigation – granularity



Sample investigation – C++ class

NetworkManager class (pseudo-code)

- start_listening(const std::string logical_addr)
- start_receiving(const std::string logical_addr, callback)
- send_to(const std::string logical_addr, const void* msg, size_t size)
- publish & subscribe with suitable arguments
- (where logical_addr ~= “Part3:DataRequest:RU-55”)

Some notes

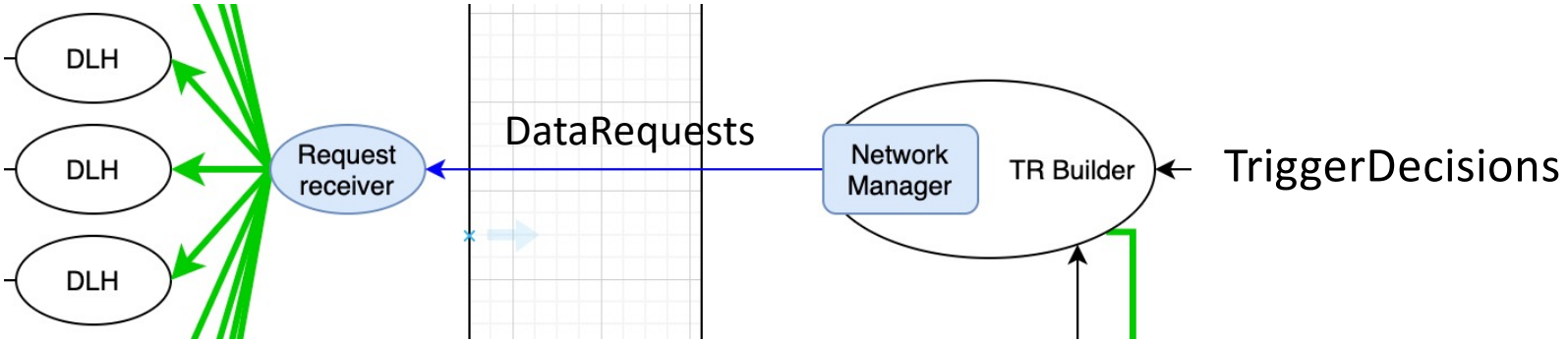
- This model implies a switch to using callbacks for receiving messages
- It also implies that user code will handle (de)serialization of messages (there will probably be interest in providing helper tools)

Other plans

Add ‘destination’ and ‘return address’ to DataRequest message

A little detail on routing

The proposal is to keep the existing list of components, by GeoID, in the TriggerDecision message unchanged



Several 'lookup tables' to get DataRequests to the right DLH

1. GeoID to RequestReceiver logical address (in TR Builder)
2. RequestReceiver logical address to hostname and port in the NetworkManager [naming service; something from K8S - TBD]
3. GeoID to Queue in the RequestReceiver(s)

Prototyping

There are ongoing discussions about many details of this work.

To help provide some focus for those discussions, and to gain experience with the issues with various use cases and the pros/cons of various approaches, we'll be looking into some demonstration modules and small systems.

Topic that came up during discussion

Phil reminded us that one of the advantages of the Q2N/N2Q model is that the user modules don't need to know whether they are running in the same DAQProcess as their collaborators or in different ones. And, there are situations in which we may want to keep that flexibility.

As such, the target for the changes described in this talk may be interactions that we expect to always be between different processes. For example, between TriggerRecordBuilder and Readout modules, since it is a reasonable choice to always have separate Dataflow/TRB and Readout processes.

The Trigger modules may be a different story. For those, it may be beneficial to keep the flexibility of Q2N/N2Q.