The DUNE-DAQ Application Framework

Eric L. Flumerfelt for the DUNE Collaboration

Abstract—The Deep Underground Neutrino Experiment (DUNE) is a next-generation neutrino experiment that will probe the properties of these elusive particles with unparalleled precision. It will also act as an observatory for neutrino bursts caused by nearby supernovae, in the event that one occurs while the experiment is in operation. Given these goals, the DUNE trigger and DAQ system must be able to maintain extremely high uptime and provide a path for full readout of the detectors for very long times (up to 100 s). To achieve these ends, we have designed the DUNE DAQ system around a flexible "application framework", which provides a modular interface for specific tasks while handling the interconnections between them. The application framework collects modules into applications which can then be interacted with as units by the control, configuration and monitoring systems. One of the key features of the framework is its communications abstraction layer, which allows for modules to interact with both internal queues and external network connections with a single transport-agnostic interface. We will report on the architecture and features of the framework.

Index Terms—Data acquisition, Physics computing, Software packages

I. INTRODUCTION

THE Deep Underground Neutrino Experiment (DUNE) [1] is a next-generation neutrino experiment that will probe the properties of these elusive particles with unparalleled precision. It is based in Lead, SD and receives neutrino beam from Fermilab, a 1300 km baseline, as seen in Fig. 1.

Up to four far detector modules will be constructed, with the first two being liquid-Argon time-projection chambers, Fig. 2. It will also act as an observatory for neutrino bursts caused by nearby supernovae, in the event that one occurs while the experiment is in operation.

Given these goals, the DUNE trigger and DAQ system [2] must be able to maintain extremely high up-time and provide a path for full readout of the detectors for very long times (up to 100 s). To achieve these ends, we have designed the DUNE DAQ system around a flexible "application framework", which provides a modular interface for specific tasks while handling the interconnections between them.

During the first DUNE 35 T and initial ProtoDUNE test runs, the *artdaq* DAQ framework from Fermilab was used

Eric L. Flumerfelt is with the Fermi National Accelerator Laboratory, Batavia, IL 60510 USA.
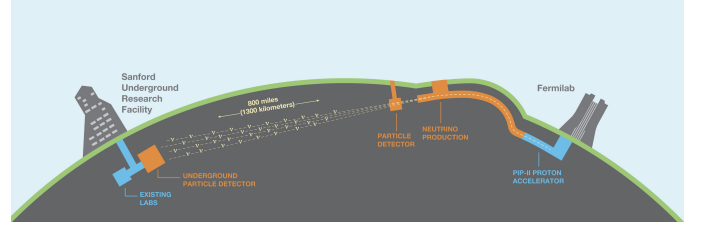

Fig. 1. This cutaway illustration shows the path of neutrinos in the Deep Underground Neutrino Experiment. A proton beam is produced in Fermilab's accelerator complex (improved by the PIP-II project). The beam hits a target, producing a neutrino beam that travels through a particle detector at Fermilab, then through 800 miles (1,300 km) of earth, and finally reaches the far detectors at Sanford Underground Research Facility.
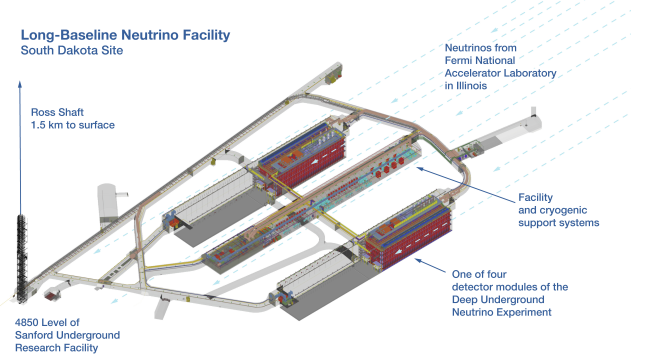

Fig. 2. Long-Baseline Neutrino Facility South Dakota Site

[3]. The up-time requirement for DUNE DAQ is 99.5%, leading to a requirement that the DAQ be able to handle run-time changes of the components selected for readout. It was found, however, that *artdaq* lacked sufficient flexibility to provide this functionality. Additionally, the *artdaq* system layout did not provide hooks for some of the data processing steps needed at the readout level, resulting in less-maintainable plugin implementations.

The DUNE DAQ application framework was designed to meet the requirements of the DUNE. It provides a common set of APIs and utilities that allow for implementation of the DAQ logic in a consistent and maintainable way, while also providing upgradability through the use of dynamically-loaded plugin implementations for well-defined application interfaces.

II. DUNE DAQ OVERVIEW

The DUNE DAQ consists of a number of application components which communicate with one another as in Fig. 3. Each application handles a specific set of DAQ functionality using a number of "DAQ Modules" specific to that application.

1) Readout applications receive and buffer data from detector electronics, and send processed waveform data
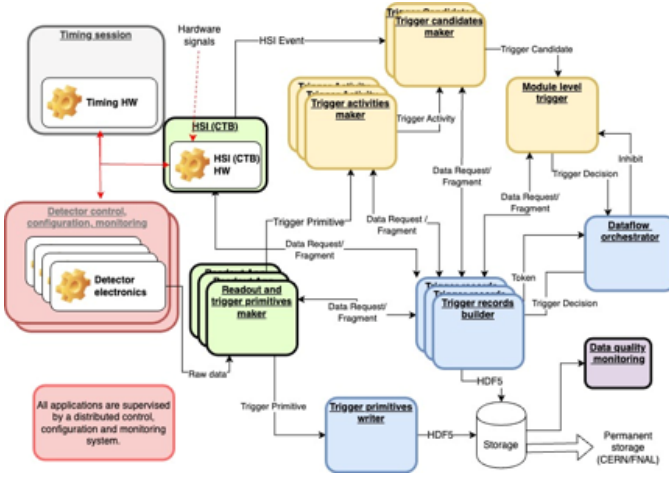
Fig. 3. Current organization of the DUNE DAQ software applications. Messaging connections between applications are shown along with their data types, though it is important to note that DAQ modules within the applications are sending and receiving messages, not the applications themselves.
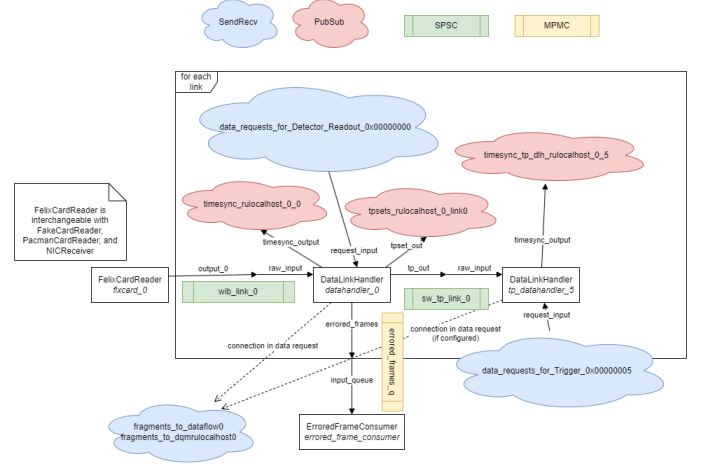

Fig. 4. Diagram of the readout application used in the DUNE DAQ. Modules are represented by white boxes, and network connections and queues are also shown. Queues are always strictly internal to the application, whereas network connections are always single-sided from the application's perspective.

(called Trigger Primitives) to the trigger applications. They listen for incoming data requests from the builder application. The internal structure of a readout application is depicted in Fig. 4.

2) The Hardware Signals Interface (HSI) application interfaces with the timing hardware and the Central Trigger Board (CTB) to produce HSI Event messages for periodic and beam event triggers.

3) The trigger applications create trigger decisions based on configurable algorithms using the trigger primitives sent from the readout. The trigger primitives are broadcast to all applications which have registered themselves as receivers, so the number of trigger applications is flexible and dependent on the number of desired trigger algorithms.

4) The trigger record builder applications receive trigger decisions, request data from the readout and trigger applications, and build Trigger Records which are saved to disk.

5) The Dataflow Orchestrator (DFO) application arbitrates the distribution of trigger decisions to builder applications based on current run-time conditions, and protects the builder applications from becoming overwhelmed by large trigger records, inhibiting the production of new trigger decisions if necessary.

6) An additional Trigger primitive writer application may be used to save the Trigger Primitive stream to disk; these files can then be used for future trigger algorithm development.

III. APPLICATION FRAMEWORK STRUCTURE

The DUNE DAQ Application Framework consists of several software packages which implement various sub-components and expose interfaces to the central framework functionality, as shown in Fig. 5, and the *appfwk* [4] package which provides the main interfaces used by the implementation modules. As previously stated, applications are composed of a number of

DAQ modules which implement the functionality for each application type, Fig. 6. Services used by the application framework abstract their implementations behind consistent interfaces so that those implementations can be changed at-will, increasing the longevity and maintainability of the framework. Fig. 7 shows the structure of the application framework packages, as well as including one of the packages which uses the interfaces provided by the application framework (*listrev* [5] is the application framework test package).

1) The *iomanager* [6] package provides the messaging interface used by the modules, whereas the DAQ module management facilities and API are implemented in the *appfwk* package itself.

2) The *ipm* [7] and *serialization* [8] packages are used for transmitting data between applications.

3) *utilities* [9] contains a number of tools that are used throughout the framework (including DNS resolution, a generic base class for all application framework objects, and threading tools).

4) *rcif* [10] provides the interface to the run control system, implemented in *cmdlib* [11].

5) *appdal* [12] and *coredal* [13] provide the configuration interface and schema, as well as recipes for generating the most commonly-used applications and their messaging connections at run-time, using module configurations stored in the configuration database.

6) The *opmonlib* [14] package contains the interface and implementation of the DAQ monitoring and metric collection code.

7) *logging* [15] takes care of log message archiving and provides a fast, run-time configurable interface for debug messages.

A. Service Interfaces

The Application Framework provides a plug-in interface for these external interfaces, allowing for both custom and commercial off-the-shelf implementations to be used.
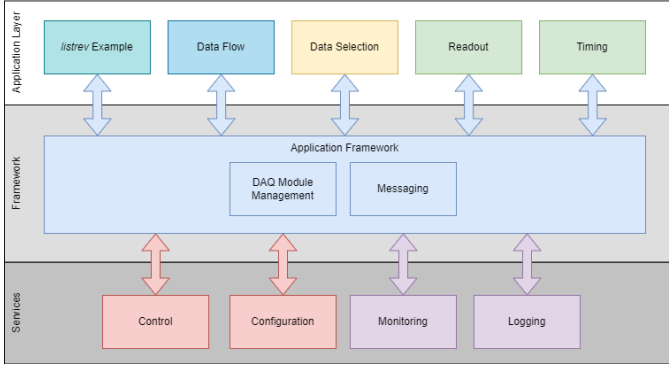
Fig. 5. Logical stack diagram of the application framework. Core services are exposed to the framework through interfaces, with plugin-based implementations to allow for upgrading functionality as technologies evolve during the life-time of the experiment. The framework's core functionality includes the definition of applications, the management of DAQ modules, and the messaging API used by the modules. Applications of various types are implemented on top of the framework to perform the data acquisition logic required by the experiment.
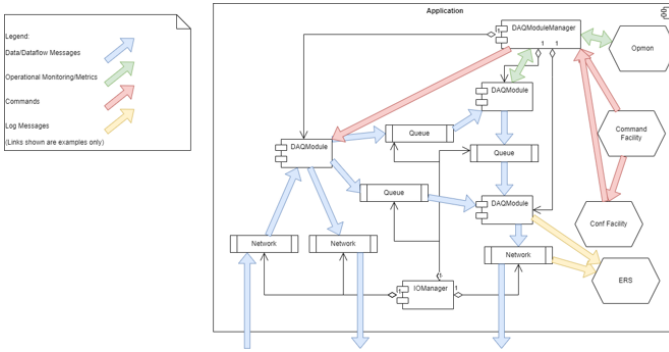

Fig. 6. Diagram of a generic application based on the framework, showing queues and external network connections. Several facilities are available to the application and exposed to DAQ modules through framework interfaces.
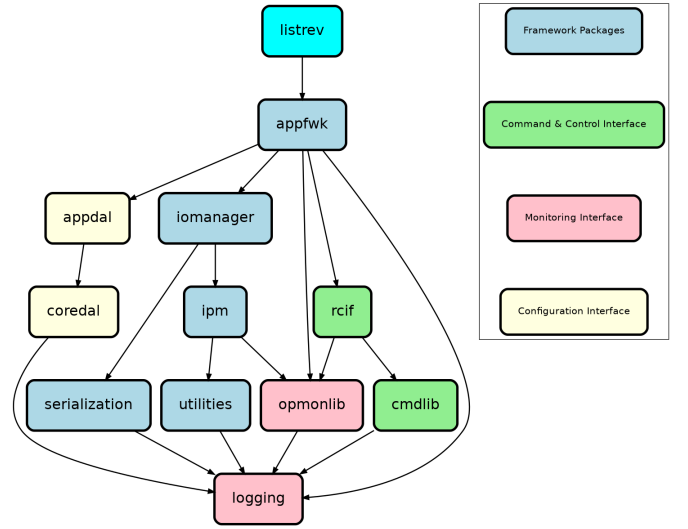

Fig. 7. Software packages comprising the application framework. The listrev package is shown as an example user of the framework, while several of the packages containing service interfaces and implementations are also shown as dependencies of the framework.

The Control interface keeps track of application state and validates incoming commands. Command transmission between applications can use a REST or gRPC-based [16] communication implementation. The Run Control system is a separate set of applications which are outside of the application framework. When a valid command is received, the application distributes it to its DAQ modules via an "action plan", a pre-configured series of parallel steps. Each step of an action plan is an "action", an aliased method within the DAQ module. A single module may register multiple actions to be executed within a single action plan, in order to define "pre" and "post" command steps.

Configuration information is provided to the application at startup using the OKS system adapted from ATLAS [17]. The configuration system uses an object-based model and retrieves configuration data based on object class and unique names. Along with the logging interface, the configuration interface is one of the basic components of the system which must be configured via the application command-line; all other service interfaces and application behavior are defined through the configuration.

Each "monitorable" element of the system is given a unique name which it can use to push metrics to the monitoring interface, which in turn publishes them using a plugin-based output facility. Additionally, the monitoring interface periodically polls for metrics in a top-down manner, ensuring that all monitored elements regularly report their current values. Current output facility implementations include JSON files and a protobuf [18] format via a Kafka [19] broker. If distributed via Kafka, the data are stored in an InfluxDB [20] database and plots are automatically construted via DUNE-developed Grafana [21] dashboards.

Log messages are handled using the ERS package [22], also adapted from ATLAS [23]. ERS provides for well-structured messages which can be collected in a back-end database (currently implemented in Kafka) for further processing. Additionally, Fermilab's TRACE [24] package provides high-speed low-level debug printouts which are assigned a name and a level, and these can be enabled or disabled at run-time. The performance of TRACE has been measured at <1 µs for enabled messages and O(1 ns) for disabled messages.

B. DAQ Module Interface

The goal of the Application Framework is to simplify as much as possible the interactions with other systems, so that DAQ modules can be written in a self-contained, task-oriented manner. It consolidates interfaces for control, configuration, monitoring, and logging; these systems interact with applications, allowing them to ignore the internal complexity of the modules and their interconnections. Users are provided with access to data received from the configuration system and the interface used to aggregate monitoring data. The framework marshals and delivers commands to modules when they are received by the applications.

The modules register their interest in certain commands, so when a command is received, the application can intelligently deliver it to just the modules that have a corresponding

action. The framework also provides an abstraction layer for connections, allowing modules to send data to an endpoint identified by a name without regard as to whether the module owning the endpoint is local to the application or running elsewhere.

Apart from a single required `init` method, the developer is free to implement whatever methods and internal threading is needed to accomplish the task of the DAQ module. The encapsulation provided by the messaging system means that the module is completely independent of other modules, though it is recommended for modules to use the callback system provided by the messaging API to schedule work on message arrival.

C. Messaging API

One of the core functionalities of the Application Framework is its messaging API. Each message channel consists of a unidirectional, strongly-typed connection (i.e. each connection only handles one type of message data), with both network and queue-based implementations for inter- and intra-process messaging, respectively. Connections are indexed by name and data type (converted to a string by macros defined in the *serialization* package and called at data type definition), and can be retrieved from an external connectivity service using pattern matching on the name. Supported topologies include point-to-point, broadcast, and many-to-one connections. Endpoints are classified as being "bind"-type or "connect"-type, based on whether the endpoint should be at a well-known or ephemeral location for the given topology. For example, in the many-to-one case, the receiver is the "bind"-type endpoint, and all senders are "connect" type. Meanwhile, for the broadcast pattern, the sender is the "bind"-type and any receivers are "connect" type (this particular topology also allows for zero receivers).

The messaging API exposed to the DAQ modules is agnostic for whether messages are internal to an application or sent over the network, allowing for greater flexibility in how modules are organized into applications. This also allows for custom applications implementing large portions of the DAQ logic to be created for smaller test scenarios. Conversely, if a particular application has a large processing load, it can be split up to take advantage of available computing resources.

Network transport and serialization use plug-in interfaces, currently implemented via ZMQ [25] (in *ipm* and MsgPack [26]). ZeroMQ "push/pull" and "pub/sub" socket types are used to implement the required connection topologies. Several ZeroMQ socket options are used to guarantee that messages are only accepted if the remote endpoints are connected, including `sndtimeo` and `rcvtimeo` set to 0 and `immediate` set on the sender socket. These options allow for the application to quickly detect when remote endpoints are not functional and attempt to reconnect.

D. Threading and Utilities

Several utility classes are provided by the Application Framework, diagrammed in Fig. 8. Most notable are the WorkerThread and ReusableThread interfaces, which provide long-lived tasks and a thread pool implementation, respectively.
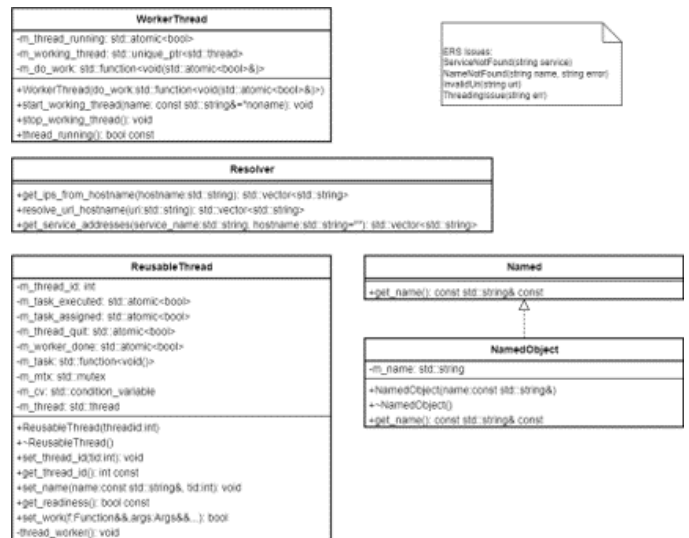

Fig. 8. UML Class diagrams of the application framework utilities.

These threading helper classes help DAQ module developers to implement their logic in a consistent way, improving system maintainability and allowing for the framework to manage thread lifetime and exception handling.

The framework provides a naming hierarchy which is used for monitoring and logging, and the *utilities* package provides the `NamedObject` interface which is the base class for all application framework classes.

DNS resolution utilities provided by the `Resolver` class are used in the network messaging implementation and have the ability to interface with the Kubernetes DNS [27] for service-based endpoint resolution.

IV. TESTING AND EXAMPLES

In order to ensure that the application framework meets the requirements of the DUNE DAQ, each component of the system is tested in both individual unit tests and through a series of integration tests of increasing complexity, cumulating in tests that exercise a significant portion of the DAQ itself, including real detector hardware.

Each package within the application framework has a suite of unit tests, and the *listrev* example was created to exercise the functionality of the framework as whole as part of the integration testing of the DUNE DAQ. Its operation is as follows:

1) A "ReversedListValidator" module broadcasts a request to create a new list of integers.
2) Each RandomDataListGenerator uses the information in the create message to product a list of integers.
3) The ReversedListValidator requests a list identified by an iteration index from the RandomDataListGenerators and the ListReverser.
4) The ListReverser, upon receiving a request for a new list, in turn requests the list from the RandomDataListGenerator, inverts the order of the list received, and sends it to the ReversedListValidator.
5) Once the ReversedListValidator has received the relevant pairs of data, it can check that the list was indeed
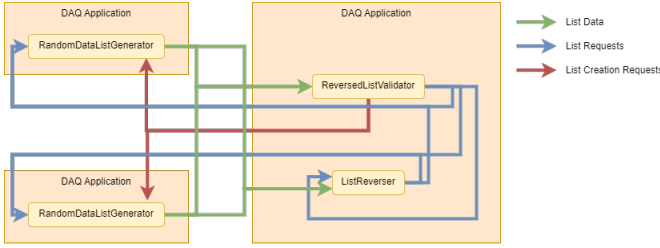
Fig. 9. Communication paths within the listrev example package. Several iterations of the example are run as part of the application framework integration test suite, to ensure that queues and network connections are properly working in the required connectivity patterns, and that the DAQ modules remain functional regardless of which application they are located in or what other DAQ modules are also in that application. The configuration shown has two generators in separate applications and one reverser and validator in a third application.

reversed and that it matches the pattern specified in the create message, seeded by the iteration index.

This example showcases many different aspects of the application framework's functionality, including DAQ module creation and managment, several messaging topologies and patterns (create messages use broadcasts, and the request of one data type resulting in a data response of a different data type), and it uses the control, monitoring, and configuration interfaces. This allows for a discrete test of the application framework without relying on detector hardware in well-controlled conditions.

V. IMPLEMENTATIONS OF THE FRAMEWORK

The DUNE DAQ Application Framework has been used as the basis for the DAQ system for ProtoDUNE horizontal and vertical drift, as well as the ICEBERG prototype detector at Fermilab and the TOAD Near Detector prototype. The system has evolved with each iteration, with different DAQ modules being used in different configurations. Readout has transitioned from Proto-WIBs to DUNE-WIBs and from FELIX-based readout [28] to WIBEth Ethernet-based readout [29]. This has shown the framework's ability to interface with different readout hardware with a reasonable development period for the DAQ modules specific to each readout type. The DUNE-DAQ Application Framework is being deployed at ProtoDUNE II for data-taking activities in 2024, with the horizontal-drift TPC currently being made ready for operations.

VI. CONCLUSIONS

The application framework has been deployed at the Proto-DUNE test detector at CERN, as well as other test detectors, and we have used the test runs to refine and improve the framework. The framework is in the final stages of development, with reviews being performed to finalize the functionality and close out development of new features. As it is a central part of the DUNE DAQ, any API changes results in a large amount of effort to reconcile existing module implementations with the change, so it is imperative to have the framework in a very stable state to allow development to reach a conclusion in other parts of the system.

We have demonstrated that the framework's plugin-based architecture allows for easy extension to additional readout types, an essential feature for the DUNE which will require readout from at least three and up to six different detector technologies simultaneously. We believe that the framework may prove useful as the basis for the DAQ suites of other high-energy physics experiments in the future.

REFERENCES

[1] *Dune homepage*. [Online]. Available: https://dunescience.org. Accessed on: May 7, 2024.

[2] A. Abed Abud, C. Batchelor, K. Biery et al., "Trigger and data acquisition (tdaq) system design," DUNE DAQ Project, Tech. Rep., Jan. 2023. [Online]. Available: https://edms.cern.ch/document/2812882. Accessed on: May 7, 2024.

[3] K. Biery, E. Flumerfelt, J. Freeman, W. Ketchum, G. Lukhanin, and R. Rechenmacher, "Artdaq: DAQ software development made simple," *Journal of Physics: Conference Series*, vol. 898, no. 3, pp. 032013, Oct. 2017, DOI. 10.1088/1742-6596/898/3/032013.

[4] *DUNE-DAQ/appfwk*. [Online]. Available: https://github.com/DUNE-DAQ/appfwk. Accessed on: May 7, 2024

[5] *DUNE-DAQ/listrev*. [Online]. Available: https://github.com/DUNE-DAQ/listrev. Accessed on: May 7, 2024

[6] *DUNE-DAQ/iomanager*. [Online]. Available: https://github.com/DUNE-DAQ/iomanager. Accessed on: May 7, 2024

[7] *DUNE-DAQ/ipm*. [Online]. Available: https://github.com/DUNE-DAQ/ipm. Accessed on: May 7, 2024

[8] *DUNE-DAQ/serialization*. [Online]. Available: https://github.com/DUNE-DAQ/serialization. Accessed on: May 7, 2024

[9] *DUNE-DAQ/utilities*. [Online]. Available: https://github.com/DUNE-DAQ/utilities. Accessed on: May 7, 2024

[10] *DUNE-DAQ/rcif*. [Online]. Available: https://github.com/DUNE-DAQ/rcif. Accessed on: May 7, 2024

[11] *DUNE-DAQ/cmdlib*. [Online]. Available: https://github.com/DUNE-DAQ/cmdlib. Accessed on: May 7, 2024

[12] *DUNE-DAQ/appdal*. [Online]. Available: https://github.com/DUNE-DAQ/appdal. Accessed on: May 7, 2024

[13] *DUNE-DAQ/coredal*. [Online]. Available: https://github.com/DUNE-DAQ/coredal. Accessed on: May 7, 2024

[14] *DUNE-DAQ/opmonlib*. [Online]. Available: https://github.com/DUNE-DAQ/opmonlib. Accessed on: May 7, 2024

[15] *DUNE-DAQ/logging*. [Online]. Available: https://github.com/DUNE-DAQ/logging. Accessed on: May 7, 2024

[16] *gRPC*. [Online]. Available: https://grpc.io/. Accessed on: May 7, 2024

[17] R. Jones, L. Mapelli, Y. Ryabov, and Soloviev, I., "The OKS persistent in-memory object manager," *IEEE Trans. Nuclear Science*, vol. 45, no. 4, pp. 1958-1964, Aug. 1998, DOI. 10.1109/23.710971

[18] *Protocol Buffers Documentation*. [Online]. Available: https://protobuf.dev/. Accessed on: May 7, 2024

[19] *Apache Kafka*. [Online]. Available: https://kafka.apache.org/. Accessed on: May 7, 2024

[20] *InfluxDB Overview*. [Online]. Available: https://www.influxdata.com/products/influxdb-overview/. Accessed on: May 7, 2024

[21] *Grafana: The open observability platform*. [Online]. Available: https://grafana.com/. Accessed on: May 7, 2024

[22] *DUNE-DAQ/ers*. [Online]. Available: https://github.com/DUNE-DAQ/ers. Accessed on: May 7, 2024

[23] S. Kolos,A. Kazarov, and L. Papaevgeniou, "The Error Reporting in the ATLAS TDAQ System," *Journal of Physics: Conference Series*, vol. 608, no. 1, pp. 012004, Apr. 2015, DOI. 10.1088/1742-6596/608/1/012004

[24] S. Foulkes, and R. Rechenmacher, "TRACE - A System Wide Diagnostic Tool," *2007 IEEE-NPSS Real-Time Conference*, 2007, DOI. 10.1109/RTC.2007.4382742

[25] *ZeroMQ: An open-source universal messaging library*. [Online]. Available: https://zeromq.org/. Accessed on: May 7, 2024

[26] *MessagePack: It's like JSON, but fast and small*. [Online]. Available: https://msgpack.org/index.html. Accessed on: May 7, 2024

[27] *DNS for Services and Pods — Kubernetes*. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/. Accessed on: May 7, 2024

[28] Soo Ryu on behalf of the ATLAS TDAQ Collaboration, "FELIX: The new detector readout system for the ATLAS experiment," *Journal of Physics: Conference Series*, vol. 898, no. 3, pp. 032057, Oct. 2017, DOI. 10.1088/1742-6596/898/3/032057

[29] R. Sipos for the DUNE collaboration, "The Ethernet readout of the DUNE DAQ system," *2024 IEEE-NPSS Real-Time Conference*, Apr. 2024, to be published