DUNE Software Framework Requirements
HSF Review

Review panelists: Marco Clemencic[a], Giulio Eulisse[a], Christian Haack[b], Matti Kortelainen[c], Charles Leggett[d], Marc Paterno[c], Simon Patton[d]
HSF frameworks WG conveners: Chris Jones[c], Kyle Knoepfel[c], Attila Krasznahorkay[a]
HSF coordinators: Benedikt Hegner[a], Liz Sexton-Kennedy[c], Graeme Stewart[a]

Introduction

In the spring of 2021, the DUNE experiment contacted the HEP Software Foundation (HSF) to review a set of requirements drawn up to help the experiment determine how to meet its framework needs. In response, the HSF coordinators and framework working-group conveners assembled a panel of framework experts from across the international HEP community.

A public workshop[1] was held on 2-3 June 2021 between members of the DUNE experiment and the HSF panelists, conveners, and coordinators. The DUNE framework requirements document (see footnote) was discussed in detail, along with questions that arose prior to the workshop.

This document presents the findings of the HSF review panel, including the coordinators and framework working-group conveners, who served as *ex officio* members of the review process. The panel's reactions to the requirements stated in that document are categorized into the following sections:

- General comments
- Missing requirements
- Requirements that need clarification
- Overly constraining requirements
- Novel requirements
- Further items for DUNE's consideration
- Appendix (sent separately): Prior experience

Instead of proposing a framework solution for DUNE, this document is intended to provide DUNE with information it should consider as they determine the best way forward for their framework needs. The comments herein should therefore not be construed as criticisms or endorsements, *per se*, of any particular framework option.

[1] https://indico.cern.ch/event/1038551/

a European Organization for Nuclear Research (CERN)
b Technical University of Munich
c Fermi National Accelerator Laboratory (FNAL)
d Lawrence Berkeley National Laboratory (LBNL)

Review findings

General comments

Use cases should be clear

Our understanding is that DUNE solicited physics use cases to form the list of requirements. References to these use cases should be included for each of the requirements so that a developer or maintainer may understand the context of the requirement and whether it is appropriate for the actual physics need.

Define terms intended to convey framework concepts

It is important to define any terms that convey a framework concept. The following list contains such terms that are used yet undefined in the requirements document

- Algorithm
- Service
- Data store

Please provide definitions for them.

Requirements should focus on specific behavior

A common problem among software users is to specify a solution as the requirement instead of stating the specific behavior that is desired. An example of this is requirement 3, which states that the framework should provide a Turing complete language for configuring a framework program. Based on the annotated requirements document, it would be better to specify that the framework should support a configuration language with (e.g.) control-flow constructs (list comprehensions, conditionals, etc.) and arithmetic capabilities. Such a description maps closer to a configuration language you may have in mind (such as Python or Jsonnet) than something that's generally Turing complete (such as C).

Requirements should be verifiable

The following statements were made in the requirements document:

- *Requirement 28: "Data products should not occupy memory beyond their* <u>*useful*</u> *lifetimes"*
- *Requirement 31: "Labelling of data objects...should be* <u>*intuitive*</u>*"*

The underlined words above are subjective and are not generally verifiable. In order for a developer to ascertain whether a requirement has been met, a specific explanation should be used instead of subjective terminology. For example, requirement 28 would be better phrased as "a given data product should not remain in memory past the execution of the last module or algorithm that requires it."

Missing requirements

There are some requirements that are either missing or implicitly assumed. This section enumerates such requirements.

Timescale

We understand that DUNE's developers are engaged in other, smaller scale LAr TPC experiments that would benefit from a timely DUNE framework solution. However, given that DUNE's far-detector data-taking is projected to occur in roughly 10 years, it is not immediately clear what factors impel DUNE to settle on a framework solution right now.

DUNE should therefore specify a timescale for when a framework solution must be in place, mentioning which aspects of the timescale are informed by experiments that will come online sooner than DUNE does. As technologies can evolve rapidly, we encourage you to reserve any detailed projections for only the next 5 years, allowing some flexibility for what may need to happen longer term.

Framework migration and backwards compatibility

DUNE currently develops offline functionality in the context of the *art* framework. It also has produced files in art/ROOT format, including files for detector simulation and also those that contain ProtoDUNE data. A requirement should be specified as to what extent backwards compatibility is needed. Specifically, to what degree (if any) must the framework (e.g.):

- Support DUNE's existing algorithms and code structures
- Read and properly interpret existing simulation and (ProtoDUNE) data files

A cost/benefit analysis will help determine how much bridging from one framework to another is reasonable.

Online constraints

The panel understands that the online system was not in the scope of the review. It is important, however, to understand the interface between the online and offline contexts. Therefore, any constraints imposed on the offline framework by the online system should be explicitly stated.

Programming languages

In DUNE's annotated requirements document, requirement 38's annotation states:

> *"We specifically did not specify a programming language or packaging system in the requirements. We are looking for a framework that is portable between architectures and environments."*

It is likely that DUNE's framework solution will support C++, making it a reasonable assumption. However, what about interoperability with other languages (Python, Julia, Rust, etc.)? Such interoperability may suggest a data model that is more flexible (e.g. Apache

Arrow) than what current frameworks support. In this case, providing a reasonable subset of languages the framework must accommodate will be more helpful for a developer than not specifying any language.

HPC usage

It is emphasized several times in the requirements document (and its annotations) that use of the framework should be supported on HPC systems (e.g. requirement 38):

> *"The same code developed and tested on local resources must scale to large resources. This should include HPC resources as far as possible."*

The requirements document should reference use cases and workflows where such HPC utilization is important. The framework solution may look very different depending on the HPC capabilities that are required (multi-node processing, internode communication and use of network interconnects during processing, offloading to accelerators, etc.).

Requirements that need clarification

Processing contexts

There are different data-processing contexts required by DUNE (beam data, calibration data, streaming data for debugging, supernovae, simulation, physics analysis, etc.) and it is unclear how the processing requirements vary from one context to another. Please enumerate:

- Which contexts will require a framework,
- For those that require a framework, the relevant groupings of data that need to be processed (e.g. trigger record *vs.* slice/trigger primitive *vs.* APA), and
- Which processing contexts should be scalable from laptop environments to large-scale HPC jobs.

Memory usage

Throughout the workshop, prudent use of memory was stated by DUNE as a crucial consideration in processing data. However, the requirements document is relatively silent on memory estimates (with some exceptions in the annotations), and any estimates that have been provided are typically limited to only data use cases (e.g. trigger-record readout for beam data and supernovae) and do not seem to take into account other contexts such as simulation or machine-learning algorithms used in reconstruction.

In addition, the memory estimates for the data-only use cases cover a wide range: 40 MB of raw 12-bit data per APA readout for a single drift, 6 GB for a full trigger-record drift readout from all 150 APAs, to over 100 TB for a supernova event. These estimates do not account for inflating 12-bit data to 32-bit native C++ types, nor do they allow for multiple copies of data in flight at a given time, which is often necessary during file I/O operations.

Although it is likely possible to process portions of a trigger record, thus reducing the memory usage, not all the cases that the framework will need to handle were presented. If memory is of paramount concern, then the document should enumerate each of the processing steps that will be the responsibility of the framework (e.g. simulation, reconstruction, analysis) and their putative memory estimates. More firmly established estimates will have a significant impact in determining how the framework needs to adapt the running of jobs to the environment.

Overly constraining requirements

Reproducibility

Requirements 5, 6, 7, and 14-16 emphasize the need to be able to reproduce various types of results, ranging from a final framework configuration to physics results themselves. Reproducibility is a notoriously difficult subject and, except in specific contexts, generally unattainable. The document should specify more consistently what *types* of reproducibility are required.

For example, creating a precisely predictable final configuration (requirement 6) is sensible and achievable depending on what configuration language/features are used. Exact reproducibility of random-number generators can also be achieved if a suitable technology and seeding technique are chosen. Requirement 5, however, states that a robust persistency and versioning system must exist to enable reproducing “previous results,” which is non-specific.

In general, bit-wise reproducibility cannot be guaranteed on even the most predictable systems (e.g. CPU) for floating-point operations. The problem is compounded when multi-threading is adopted or when offloading to accelerators is required. For many situations, it may be more practical to require statistical consistency of results than bit-wise exactness.

We therefore encourage DUNE to be explicit (which it is at times) about what reproducibility is required.

One framework only

Although it is commendable to pursue common software to lessen maintenance and usage burdens, forcing all data-processing steps into a single framework is likely to hinder rather than enable agile development, production, and analysis. It may instead be preferable to share *components* (e.g. algorithms or modules) among several frameworks and to adopt a configuration language and coding syntax that is common among them.

We offer some further considerations for the reconstruction and analysis use cases.

Reconstruction

The physics contexts are disparate enough for DUNE’s physics use cases that accommodating all of them by one framework may be difficult. For example, it seems unlikely that supernova data will be processed on a laptop. DUNE could consider having multiple frameworks, each one tailored for a specific use case (e.g. beam data) or platform (e.g. HPC). Each framework might execute the same algorithms, just orchestrated by a dedicated framework in a way that fits the use case or platform.

Analysis

Neutrino analyses come in different physics flavors (e.g. neutrino oscillations, proton decay, supernovae, multi-messenger studies), which may not be well suited for any one particular framework. It is also likely beneficial to decouple the offline and analysis frameworks if MPI functionality, which has been successfully used by neutrino analyses in the past, cannot be provided by the offline framework.

Lastly, in the panel's experience, attempts to meet both offline and analysis needs with a single framework have been largely unsuccessful. Notable exceptions are ALICE and Belle II, whose common reconstruction/analysis frameworks may be possible in part due to the fewer triggered data streams required by the experiments.[2]

Although an offline framework should be mindful of analysis needs, designing a framework to meet both offline and analysis needs is difficult to achieve, both technically and sociologically. Such an attempt would need to involve analysts from the earliest stages and see their use cases as paramount and continually tested against the developing software.

Novel requirements

The requirements listed here are a poor fit to existing frameworks and would require development.

Multi-node processing

Multi-node processing is not new to HEP (e.g. ALICE's O^2 framework or MPI-based analyses). However, a framework that can efficiently schedule jobs on such disparate scales as a laptop up to a many-node HPC system (see requirement 39) has not been demonstrated before. As this requirement could have far-reaching implications for framework design, changes in node capabilities should be monitored closely over the next few years--what might be multi-node today may not need to be 5 years from now. If the need for such dynamic and scalable functionality can be demonstrated, then development outside of current framework technologies would be required.

Overlapping processing atoms

Requirement 27 states that the framework

> *"must be able to operate on subsets of a trigger record. Specifically, it must be possible to break trigger records down into smaller chunks (e.g. one APA) and be able to stitch those chunks back together."*

[2] ALICE primarily triggers on minimum-bias events, and the decay products of the $\Upsilon(nS)$ resonances detected by Belle II are significantly cleaner than those of hadronic collisions.

This requirement, although listed under the heading "Memory management," presents more processing challenges than memory ones, and it is related to the processing contexts subsection above. A critical ingredient of most HEP frameworks is the ability to process framework events as independent from each other--i.e. the order in which events are processed is irrelevant. If DUNE's processing model requires correlations between its processing atoms, or if data from adjacent drift windows overlap each other or impose boundary conditions that must be respected, then this argues for a framework design that takes this into account upfront.

Additional complications include combining the results from different drift windows within the same trigger record, and (for supernovae contexts) finding ways to straddle multiple trigger records within the same job. Although extant frameworks may be able to accommodate such processing, there is no single framework that we know of that readily provides such processing behavior.

Meeting this requirement will not only require significant software development, but it will also likely necessitate involvement with DUNE's online developers. Consequently, DUNE may need to consider a simpler framework model, perhaps moving the (e.g.) APA-stitching further down the processing chain.

Fluid data-processing hierarchy

Most HEP frameworks have little flexibility in their data-processing hierarchy. The hierarchy used by DUNE's current framework *art* (*run* ⊃ *subrun* ⊃ *event*) has its origins in the collider community (viz. CMSSW), where the subrun represents a luminosity block. Depending on which processing contexts must be supported by the framework (see above), a more flexible and fluid data-processing paradigm may be required. If so, development would be necessary not only to determine an efficient computing model but to also develop a system that can sufficiently meet DUNE's provenance needs.

Further items for DUNE's consideration

Framework *vs.* production system

The concept of a framework job has evolved over the last 10 years--from one single-threaded process to a multi-threaded one, to a single program that can launch many processes with many threads potentially distributed across multiple nodes, communicating in coordinated ways.

If DUNE is considering a framework that supports such multi-node programs, it is important to understand the boundary between the framework itself and the production system--i.e. the entity responsible for managing the launching of framework jobs and collecting (and perhaps validating) their outputs. For example, any job that uses MPI effectively is responsible to some degree for scheduling the work among various ranks, which can reside on physically distinct machines. It will therefore take analysis to determine which of these details, for a given data-processing scenario, should be managed by the framework and which should be handled by the production system.

An extant example demonstrating the framework/production system boundary problem is ATLAS' event service.[3] The event service supports an external scheduler responsible for assigning event-processing to (potentially physically isolated) systems in a coordinated manner. An HPC-specific implementation of the service exists (see footnote 3) that relies on MPI for scheduling processing on NERSC. ATLAS' experience in this area may be of benefit to DUNE.

[3] D Benjamin *et al* 2017 J. Phys.: Conf. Ser. **898** 062002 [doi:10.1088/1742-6596/898/6/062002]