art: A Framework for New, Small Experiments at Fermilab

Robert K. Kutschke
Fermi National Accelerator Laboratory, Batavia, Illinois, USA
E-mail: `kutschke@fnal.gov`

Abstract. Fermilab is preparing to mount a variety of new experiments at the Intensity Frontier, all of which require infrastructure software including a framework, an event data model, persistency, run-time configuration, management of singleton-like entities such as the geometry and conditions data, integration with Geant4 (G4), build and release management, and integration with GRID based work-flow management systems. In order to maximize the return on both past and future effort invested in supporting CMS, the Fermilab Computing Division (CD) has extracted the core of the CMS framework plus many parts of its associated infrastructure software; CD is supporting this infrastructure for use by the new Intensity Frontier experiments. This talk will present the plans for and status of this infrastructure software including points of view from both the developers and the physicist-clients working on the Mu2e experiment.

1. Introduction to Mu2e

The Mu2e collaboration has proposed an experiment to search for the coherent, neutrino-less conversion of a muon into an electron in the Coulomb field of a nucleus. With the envisioned design, the experiment will have a sensitivity of about 10^4 times that of the previous best experiment, and, for couplings of order unity, will have sensitivities to mass scales of up to O(10^4 TeV). Contingent upon DOE approval, Fermilab has scheduled Mu2e construction start in 2013 and the first physics run in 2018. Additional information about Mu2e is available at the collaboration's web site[1].

In order to allow members of the collaboration to focus on physics issues, the collaboration decided, as much as possible, to acquire, rather than develop, its infrastructure software. These proceedings discuss the results of that effort.

2. Frameworks and Infrastructure Software

As used in this proceedings, the term infrastructure software refers to the full body of the non-physics software used by an experiment, including a framework, a persistency mechanism, data presentation tools, build management, release management, workflow management, file catalogs, and databases for geometry, conditions data and metadata. It does not include the physics content of the databases, the internals of the event data classes, Geant4, or any of the algorithms for calibration, reconstruction or analysis. In this usage the framework has a very limited set of tasks; it provides the machinery that drives the event loop, a few services that are integral to the operation of the framework, the transient representation of the Event-Data

Model (EDM) and run-time configuration. Although these functions are few, they form the glue that holds the infrastructure software as a coherent whole.

3. The Origins of art

In the fall of 2008 Mu2e began a discussion with CD about what infrastructure software CD would be willing to support and at what level of effort. During these discussions, the use cases included analysis, reconstruction, calibration, simulation and use in the non-real-time parts of the Trigger/DAQ/Monitoring system. The collaboration wanted a highly flexible, lightweight framework in which the application specific codes can see a uniform interface to the the rest of the infrastructure software. In the list of use cases, the analysis case was given high priority because this is the single use case that the largest numbers of Mu2e physicists will see; a key requirement was that the infrastructure software, as viewed by an analyzer, be highly teachable.

During the discussions between Mu2e and CD, several other Fermilab Intensity Frontier experiments made similar requests for support from CD; these experiments included MicroBoone, NOvA and muon g-2. The result of these discussions was that CD would recommend infrastructure software and provide support ramping up to approximately two FTEs. The planned level of support was reached by mid 2010 and has continued since that time.

The list of candidate frameworks included the five frameworks supported at that time by CD (CDF, D0, CMS, MiniBoone and MINOS), plus several third party frameworks FMWK[2], the ALIROOT/ILCROOT family, GAUDI and JAS. It was quickly realized that, in the long run, much of the the work required of CD would be to support the interactions among the framework and the other components of the infrastructure software. Because the envisaged level of support was quite small, about 2 FTEs, Mu2e and CD decided that a third party framework was not a credible alternative; to achieve the goals it would be necessary to leverage the experience with one of the frameworks already supported by CD.

Among the five frameworks supported by CD, the CMS framework is the most modern and most robust. Moreover it is the only one of the five that is expected to have a support-life that covers the projected lifetime of Mu2e, at least until the early 2020's. Therefore the CD team recommended using the CMS framework[3] but with a few evolutionary changes. The Mu2e collaboration agreed with this recommendation. Features of the CMS framework that will be retained include: the state machine that drives the event loop; the concepts of Modules and Services and their base classes; the five types of modules, Analyzer, Producer, Filter, Source and Output; the concept of a module label; the three part event ID; the full EDM, including both the transient and persistent representations; the three trees in the event-data files, Event, Run and SubRun (known as LuminosityBlock in the CMS framework); the four part data product ID; data product provenance; the exception handling strategy; TFileService, the class that maintains separate root working directories for each module label; the message logger; and, reconstruction on demand. At present, it is planned that scheduled reconstruction will be eliminated, leaving only reconstruction on demand.

It is also planned to replace the run-time configuration language, currently python. This has been a controversial decision. Part of the reasoning is that Mu2e currently has a variety of run time configuration files that hold information that will one day live in databases: the description of the nominal geometry, lists of physical properties not found in G4, and the very simplistic first implementation of conditions data. Mu2e wanted a single language for use in all of these run time configuration applications.

The philosophy behind the evolutionary changes is to identify features that are of little benefit to experiments that are of much less complexity than CMS; any of those features that are either hard to use or hard to support are candidates to be replaced or to be removed outright. One example is that EventSetup will be removed; this is the code that manages conditions data for CMS, allowing arbitrary intervals of validity for conditions entities and allowing event mixing across intervals of validity. All of the intensity frontier client-experiments expect that a traditional Services based conditions system will meet their needs.[1] A second example is the coupling between the build system and the dynamic loader that allows a user to ask for a module by class name. Mu2e expects to have an order of magnitude fewer module classes and proposes a convention that will automatically detect module name collisions at run time. While it would be preferable to have a system that detects collisions at build time, the proposed solution is adequate and has the advantage of reducing build-time coupling between components.

Another sort of evolution is to remove support for features that are required for backwards compatibility with older CMS practices or data formats. An example of the former is that Mu2e will perform event merging and overlays in Producer modules, not source modules; this is also current CMS practice but the CMS code base includes support for their original implementation, doing merges/overlays in Source modules. The choice of using Producer modules separates the job into two steps, an IO component that is best addressed by a computing professional, and a physics component that is best addressed by a physicist.

Finally, the Mu2e collaboration evaluated the CMS system for making product to product references and decided that they wanted a more full featured system, one like the CLEO III Lattice product, which also allows users to view a set of many data products as if it were a single class. An example is the relationship between tracks in the tracking system and clusters in a calorimeter; the producer of tracks should not know anything about the calorimeter, and vice versa. A third class understands which tracks point at which cluster. In this model all producers can act without requiring extra knowledge but the user can see the three collections as, apparently, a single entity.

The above changes are big enough to require a fork from the CMS code base, which will make it more difficult to incorporate improvements from CMS into Mu2e code and vice versa. So one must ask if the additional features are worth the cost. After some discussion the groups decided that enough of the core architecture will remain the same that significant cross-development benefits will remain.

4. Timeline

In January 2009 a team of 4 people from CD worked for one week to fork a first release from CMSSW; this included the framework, as defined in the previous section, plus the ROOT I/O based persistency system. All necessary classes were included in the fork, including some that will later be removed; EventSetup, for example, was replaced with a stripped down class that provides empty versions of only those methods needed for the code to link. The release was certified on both SLF4 and SLF5 for both 32 and 64 bit hardware; a port to MAC OS is likely but, at this time, a port to Windows is not foreseen.

This release was used as an opportunity to get real-life experience with a new build system, scons, and to evaluate it as a candidate for the long term build system. At about the same, other groups in CD similarly evaluated cmake as a candidate build system. When the two groups compared notes, the winner was cmake, mostly because of its much better support for running suites of unit tests; otherwise scons performed very well. In the next major release of the framework, cmake will replace scons.

Within days of the first release of the framework, the first release of the Mu2e software appeared. This featured cartoon implementations of the detector and data products that were used to understand how to use the framework and to document that knowledge through example suites. The constantly evolving documentation for the Mu2e Offline software, including Mu2e

[1] Following the oral presentation of this talk, a member of the TOTEM collaboration mentioned that they also developed their software from CMSSW and that they invested a lot of effort learning to use EventSetup before giving up in favour of a simpler approach.

written documentation about the framework, is available online[4].[2]

Next, Mu2e learned how to use Geant4 within the framework. While both the framework and Geant4 want to own the event loop, the required behaviour is that the framework own the event loop and that Geant4 process one event at a time, with its output copied into the Event as one or more data products. Mu2e adopted a solution suggested by the Geant4 team and similar to that used in CMSSW: create a new class, Mu2eG4RunManager, that inherits from G4RunManager and overrides the BeamOn method, breaking it into several pieces, beginRun, a method to produce one event, and endRun. All other behaviour of the G4RunManager class is retained in the subclass. Mu2e created a framework module that, in its beginRun method, initializes Geant4, and, in its produce method, tells Geant4 to simulate one event.

Much of the G4 API requires that the user create objects on the heap and then pass a pointer to the object to G4. In some cases G4 takes ownership of the object and deletes it at the end of run or end of job. In most cases, however, G4 does not delete the object and the user code is responsible for the delete. Often it is inconvenient to code the delete and the missing delete has no impact on code performance; so it is tempting to skip the delete. The downside of this choice is that it creates a poor signal to noise ratio when using leak checkers. To make the delete almost automatic, the user code can pass the pointer to G4 and then add it to a primitive garbage collector that cleans up everything at the end of the job. The garbage collector is deployed within a framework Service named G4Helper; as a framework Service G4Helper is available from anywhere within the executing program.

Beginning in the spring of 2010, and using the Mu2e code as a model, a member of the MicroBoone collaboration began a port of their code to art. This work was done at very low duty cycle and took about one FTE-month, spread over about six months. By October 2010 MicroBoone had frozen its previous software suite and was accepting new code only in its art based system.

Over the summer of 2010, a few members of the NOvA collaboration began a port of their software to art, using the MicroBoone code as a model. Together MicroBoone and NOvA requested that several new features be added to the framework and the CD support team agreed to add them. One requested feature was the ability to run a few events, reconfigure some modules and then replay the same events; this feature was provided by their previous framework and was heavily used in development and debugging. Another requested feature was support for polymorphism in the EDM. Suppose that an experiment has a base class, B, and a derived class, D, that extends B; there may be data products that are a collection of B objects and there may be other data products that are a collection of D objects. The requested feature is that a user be able to view a collection D objects either as a collection of D objects or as a collection of B objects, at the discretion of the user, and that one be able to have both views within one module. Mu2e is planning a different method for getting similar functionality; the plan is to use a method similar to the CLEO-III Lattice to join the extended information to the base information.

By January 2011, the major features requested by NOvA were implemented and, at their January 2011 collaboration meeting, NOvA voted to adopt art as their framework.

At present Mu2e maintains Mu2eG4RunManager but would like for this class to become part of the CD maintained infrastructure. In this way it can be used by other experiments. This new product would not be part of framework proper because, by design, the framework must not have a build-time dependence on Geant4.

[2] Initially this documentation relied heavily on links to the CMSSW wiki-based documentation. In late 2009 those pages were made private. The author would like to thank the CMS collaboration for once again making their software documentation public.

5. Promoting Best Practices

After gaining experience with the framework, Mu2e has started to define a set of best practices for using the framework; some examples are given here. Data products that are intended to be persisted must not contain pointers; instead they should contain a pair of a data product ID and an index into that data product. At present, the resolution of this pair into a pointer is left to the user but tools to automatically turn this into a pointer will be provided soon; at present the plan is to model those tools on the CLEO III Lattice software. The MicroBoone and NOvA collaborations have chosen a different solution; they use a framework supplied class that behaves similarly to ROOT's TRef.

When code extracts information from ParameterSet objects, Mu2e encourages default values in the code; this avoids ParameterSets becoming so large that they are unreadable. Mu2e encourages the use of the appropriate sort of safe pointer in preference to a bare pointer; the principal exception to this recommendation is for interactions with ROOT and Geant4, which often require bare pointers. Event generators should be written as EDProducers, not as EDSources. Mu2e intends to remove the GetByType methods from the Event interface, preferring the methods that use InputTags; this will smooth the transition to reconstruction on demand.

Mu2e encourages code to test rigorously all assumptions on which it depends; if a condition is not met the code should throw an exception. Similarly, Mu2e prefers the use of at(i) over [i] for random access into a std::vector. When code stabilizes and execution speed becomes more important, these last two safety features can be easily identified and removed; for the present environment of frequent disruptive changes, they are critical.

6. Refactoring

At the time of the conference, the CD team was in the middle of a push to refactor the framework and add new features; this will break backwards compatibility with CMSSW. The refactoring included removing EventSetup, removing features needed for compatibility with old CMS practices and data formats, and breaking couplings that are no longer necessary. An example of the last item is that the MessageLogger will be distributed as a standalone software product, not as a part of the framework. This will allow non-framework applications to use the MessageLogger without bringing in dependencies on the framework. Another part of this effort has been to replace python as the runtime configuration language. The replacement language, the Fermilab Hierarchical Configuration Language (FHICL), has been designed for use both in the new Intensity Frontier experiments and in a new Lattice QCD project. One of the key design goals for FHICL is to separate the physics description of a job from the details of workflow, leading to a workflow system with minimal awareness of the job configuration.

This work was done by a team of about four people who spend two or three uninterrupted days, every two weeks on the project. The effort started in late August 2010 and was substantially complete by the end of December 2011. Since then the team has worked on adding new features.

Both MicroBoone and NOvA have adopted the refactored art. Mu2e has an imminent CD-1 review, after which it will adopt the new art.

7. Multithreading

A common theme at this conference has been that efficient use of multiple core machines will be critical to the future of HEP computing. The framework proper is thread-safe and is ready to explore module-parallel execution. Initially this work will need to take place outside of the Mu2e, MicroBoone and NOvA codes, which depend on ROOT and Geant4, neither of which are thread-safe. Mu2e envisages that art could be used as the framework for the lowest, non-real-time layer of the online system. This system is a candidate for exploring the multi-threading capabilities of art.

8. New User Experience

Those who have used the art based framework fall into three broad categories. The first category includes those who are familiar with ROOT, have at least basic C++ skills and have seen a modern framework before. For this group there are few new ideas, only new syntax, and most are doing productive work in a few hours to one day.

Mu2e has a number of collaborators whose last experiment used Fortran, Geant3 and paw. Many of these people are experienced scientists who know exactly what they want to study but who have never seen C++ or ROOT; in many cases they have never used a language with pointers and structures. To date only one member of this group has become productive using art based software; he reports that it was close to one month of almost full time effort before he felt comfortable.

A third group of users includes students who are working under the guidance of people in the second group, who can provide the physics guidance but not the technical guidance. Among this group, those with strong scientific computing skills quickly become productive but those who have weak computing skills progress very slowly.

The core problem is that even the simplest example requires one to know a little bit about too many things; so there is simply no place to start. Consider a module that gets a handle to a container of hit straws and makes a few histograms, such as the pulse height of the hit and the z position of the hit wire. In addition to knowing about the Mu2e StrawHit class, the new user must know a little about inheritance, STL, handles, using templated methods of classes, CLHEP::Hep3Vector, the basics of root and the use of TFileService. Each of these requires a significant startup investment.

CMS reports that they had a similar experience. Their solution was write the CMS Offline WorkBook[5]. Mu2e envisages a similar solution; given the limited manpower on Mu2e, we hope to develop a tutorial that can link into the CMS documentation whenever possible; the author would like to thank the CMS collaboration for once again making their software documentation public.

9. Summary

The Fermilab Computing Division has agreed to support art, a framework derived from CMSSW; the first clients, Mu2e, MicroBoone and NOvA have already adopted art and the muon g-2 experiment at Fermilab is considering art for their framework. The Fermilab Computing Division is supporting art at a level of about 2 FTEs. A possible future research direction of the CD development and support team is to use art as a platform to study module-parallel multithreading.

Acknowledgments

The author wishes to acknowledge the assistance of the staffs of Fermilab and the collaborating institutions. The author would also like to acknowledge the strong and continuing support of the Fermilab Directorate and the United States Department of Energy.

References

[1] `http://mu2e.fnal.gov` .
[2] `https://cdcvs.fnal.gov/redmine/projects/nusoft/wiki/Documentation` .
[3] `https://twiki.cern.ch/twiki/bin/view/CMSPublic/WebHome` .
[4] `http://mu2e.fnal.gov/public/hep/computing/gettingstarted.shtml` .
[5] `https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBook` .