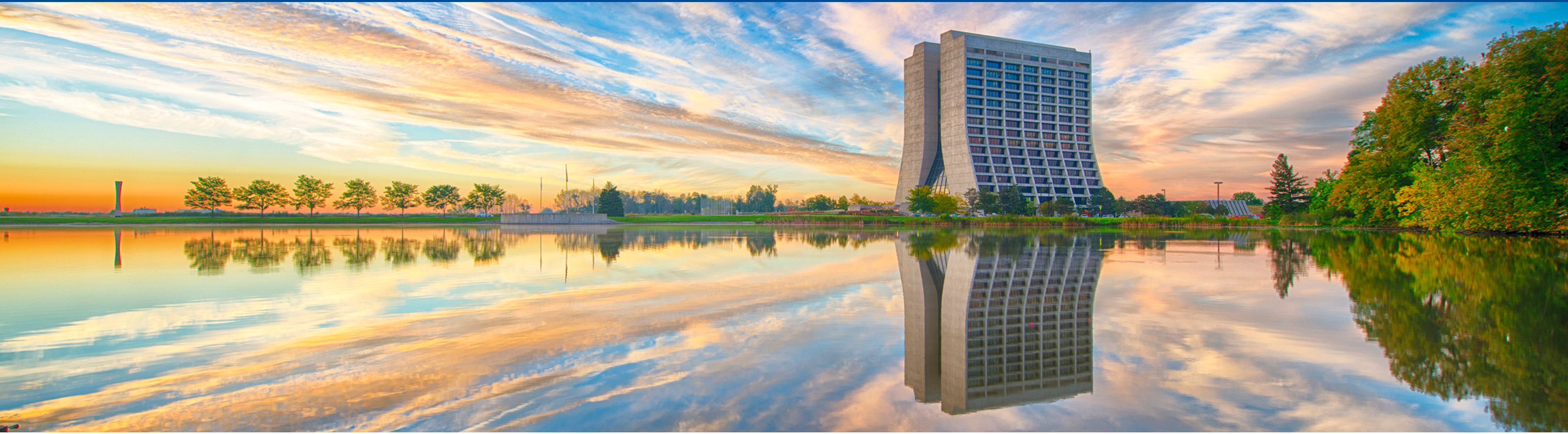

The *art* Framework: What It Is and Why

Kyle J. Knoepfel

Software & Computing Round Table

10 March 2020

https://art.fnal.gov

What it is…

art concepts

- Hierarchical data processing ($run \supset subrun \supset event$)
- **Experiments decide** how to define the processing levels (e.g. event)
- All processing elements are plugins, loaded at run-time via user configuration
  - Input source
  - Data-processing modules
  - Output modules
  - Other utilities that facilitate data-processing
- *art* provides various input sources and output modules, but all processing elements can be user-defined
- Workflows are assembled by a configuration file loaded at run-time
  - Adjustments to workflows do not require recompilation of C++ source code

Highlighted features

- **Core framework behavior**
  - Concurrent processing of events supported within a subrun (inspired by CMS)
  - Data-product management is thread-, type-, and `const`-safe
  - Core framework functionality does not depend on ROOT
    - We support a separate package (art-root-io) that provides a ROOT I/O layer
  - Output file rollover based on user-defined criteria (e.g. max. events processed)
  - Implicit data-product aggregation for non-event products
  - Secondary input (backing) files

Highlighted features

- **Core framework behavior**
  - Concurrent processing of events supported within a subrun (inspired by CMS)
  - Data-product management is thread-, type-, and `const`-safe
  - Core framework functionality does not depend on ROOT
    - We support a separate package (art-root-io) that provides a ROOT I/O layer
  - Output file rollover based on user-defined criteria (e.g. max. events processed)
  - Implicit data-product aggregation for non-event products
  - Secondary input (backing) files

- **Usability features**
  - Configuration description and validation suite
  - Module time- and memory-tracking facilities
  - Graph of data dependencies between modules

art software products

- The *art* project distributes several software products.

Full-feature framework and its
underlying packages.

art software products

- The *art* project distributes several software products.

Full-feature framework and its underlying packages.

Lightweight library that allows reading of *art*/ROOT files; does not create new data products.

Ported from CMSSW's FWLite by a CMS developer.

art software products

- The *art* project distributes several software products.

art

Full-feature framework and its underlying packages.

art GALLERY

Lightweight library that allows reading of *art*/ROOT files; does not create new data products.

Ported from CMSSW's FWLite by a CMS developer.

art STUDIO

Lightweight package that provides a development sandbox for testing new user-defined *art* modules.

art provides the framework needs for ~2k physicists

artdaq project

LArIAT experiment

Previous and potential users

Distinctive aspects of the *art* user community

- Many *art* users are doing development for experiments that are not yet running:
  - Reconstruction algorithms not yet finalized
  - Workflows are under development
- They are often involved in multiple experiments at the same time.
- They are involved in software development including event-generation, material simulation, processing raw data, reconstruction, to analyzing quantities of physical interest.
- They are defining experiment-specific data models.
- They are generally willing to (drastically) rethink any stage of the physics workflow:
  - Event representation, exploring other I/O libraries, etc.

Distinctive aspects of the *art* wrt other frameworks

- There is clear boundary between framework code and experiment code
  - Helpful conceptual distinction for users
  - Makes R&D easier

Distinctive aspects of the *art* wrt other frameworks

- There is clear boundary between framework code and experiment code
  - Helpful conceptual distinction for users
  - Makes R&D easier
- Potential for experiments to want conflicting framework behaviors/features
  - True in principle; but it has not happened in 10 years

Distinctive aspects of the *art* wrt other frameworks

- There is clear boundary between framework code and experiment code
  - Helpful conceptual distinction for users
  - Makes R&D easier

- Potential for experiments to want conflicting framework behaviors/features
  - True in principle; but it has not happened in 10 years
  - We strive hard for stakeholder consensus on any given feature.
  - We provide enough flexibility in the framework that each experiment’s needs can be met.
  - We take software engineering very seriously—any feature request that we think is unmaintainable down the road is not implemented.
  - We will not implement a feature that conflicts with the *art* processing model.

Current efforts

- *art* development has stalled until Fermilab's framework plans are more concrete
- We are working hard to help users benefit from *art*'s multi-threading support
  - Lot of effort toward upgrading LArSoft, a Fermilab-supported liquid Argon software toolkit
- We are moving toward a Spack model of delivering software
  - Designed for HPC systems
- We are working with some experiments/projects in using HPC systems
  - Includes use of alternative I/O systems (e.g. HDF5, object stores)
- Many more things, too…

Why it is the way it is…

Why it is the way it is…

Why it is the way it is…

- **Sociology –** how do users and developers interact with each other?
  - What is the support model for the framework?
  - How are new features implemented and bugs fixed?

Why it is the way it is…

- **Sociology –** how do users and developers interact with each other?
  - What is the support model for the framework?
  - How are new features implemented and bugs fixed?

- **Design –** what are the goals that need to be achieved?

Why it is the way it is…

- **Sociology –** how do users and developers interact with each other?
  - What is the support model for the framework?
  - How are new features implemented and bugs fixed?

- **Design –** what are the goals that need to be achieved?

- **Constraints** – what external factors must be accommodated?
  - Which computing languages?
  - Which data serialization technologies?
  - What type of hardware?
  - What are the memory/CPU constraints?

Why it is the way it is…

- **Sociology –** how do users and developers interact with each other?
  - What is the support model for the framework?
  - How are new features implemented and bugs fixed?

- **Design –** what are the goals that need to be achieved?

- **Constraints** – what external factors must be accommodated?
  - Which computing languages?
  - Which data serialization technologies?
  - What type of hardware?
  - What are the memory/CPU constraints?

Each of these change over time.

Why it is the way it is…

- **Developer experience**
  - Is the problem clearly expressed before a solution is attempted?
  - Are the limitations of the language, technology, etc. understood?

Why it is the way it is…

- **Developer experience**
  - Is the problem clearly expressed before a solution is attempted?
  - Are the limitations of the language, technology, etc. understood?

- **History –** how do past decisions affect future development?
  - Sloppy implementations make development more difficult (technical debt)
  - Breaking changes for users

Why it is the way it is…

- **Developer experience**
  - Is the problem clearly expressed before a solution is attempted?
  - Are the limitations of the language, technology, etc. understood?

- **History –** how do past decisions affect future development?
  - Sloppy implementations make development more difficult (technical debt)
  - Breaking changes for users

- **Human error**
  - Is the software tested? How well?
  - Are the *irreversible* parts of development understood?

Know your users

Not only do you need to know what the framework is intended to accomplish, but *who* are the individuals that will be using it.

In our experience, framework users…

Know your users

Not only do you need to know what the framework is intended to accomplish, but *who* are the individuals that will be using it.

In our experience, framework users…

- Are often willing to learn
- Are anxious to try new language features as soon as possible
- Prefer lightweight, simple-to-use systems

Know your users

Not only do you need to know what the framework is intended to accomplish, but *who* are the individuals that will be using it.

In our experience, framework users…

- Are often willing to learn
- Are anxious to try new language features as soon as possible
- Prefer lightweight, simple-to-use systems
- Rarely read documentation (which is sometimes a blessing)
- Often work under supervisors who have little regard for robust coding practices
- Often suggest solution X when they need to solve problem Y

Expect to redesign/throw away code that you write

Individuals commit the sunk cost fallacy when they continue a behavior or endeavor as a result of previously invested resources (time, money or effort) (Arkes & Blumer, 1985).

- Developing a good design is iterative.
  - Requirements change
  - Technology changes
  - Skills change, etc.
- Healthy programming includes revisiting design decisions and implementations.
  - Previous work done is rarely a waste.

Expect to redesign/throw away code that you write

> Individuals commit the sunk cost fallacy when they continue a behavior or endeavor as a result of previously invested resources (time, money or effort) (Arkes & Blumer, 1985).

- Developing a good design is iterative.
  - Requirements change
  - Technology changes
  - Skills change, etc.
- Healthy programming includes revisiting design decisions and implementations.
  - Previous work done is rarely a waste.

- *art* example
  - Before *art* 2: Data-processing loop represented by complicated finite state machine
  - *art* 2-*art* 3: Significantly simplified state machine that enabled more features
  - After *art* 3: State machine removed to facilitate multi-threaded processing

Some learned lessons: framework-specific

- *art*'s rigid processing hierarchy has been an awkward fit for neutrino experiments
  - Would probably pursue something more flexible next time
- *art* supports a class a plugins that can be accessed from anywhere.
  - This has led to many thread-safety issues that experiments must deal with.
- *art* users can access metadata and provenance about data products
  - Many users do not look at this information
  - Not clear that it has been worth it; might think of something else next time.
- Tying the CMake-based *art*-supported build system to Fermilab's custom package delivery system was arguably not the best choice.
- Framework limitations are not bad! Know what they are.

Some learned lessons: general

- Code maintenance is important
  - C++ standards/technology change
  - Beware of *obsolete* optimization
- Inheritance is overused
  - Most problems are solved without using object orientation
- Just because something can be represented in C++ doesn't mean it should be
  - Much of *art*'s metadata has been C++-based
  - Each time we have moved to relational database model, the metadata handling has become cleaner.

Conclusions

- The chance to start afresh is an opportunity to learn:
  - What has been done well
  - What could have been done better
- Know the problem you're trying to solve before you pursue a solution.
- Know the types of people who will be using the framework.
  - What are their skill levels?
  - What constraints do they have to deal with?
- Expect the code to change during its lifecycle.
- You're not in a vacuum: consult other projects to learn what they've done.

Conclusions

- The chance to start afresh is an opportunity to learn:
  - What has been done well
  - What could have been done better
- Know the problem you're trying to solve before you pursue a solution.
- Know the types of people who will be using the framework.
  - What are their skill levels?
  - What constraints do they have to deal with?
- Expect the code to change during its lifecycle.
- You're not in a vacuum: consult other projects to learn what they've done.

Thank you