SoLID Software Framework

Ole Hansen

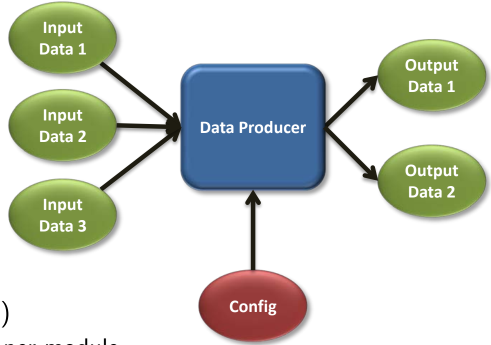
Jefferson Lab

SoLID Collaboration Meeting
January 13, 2016

General Considerations

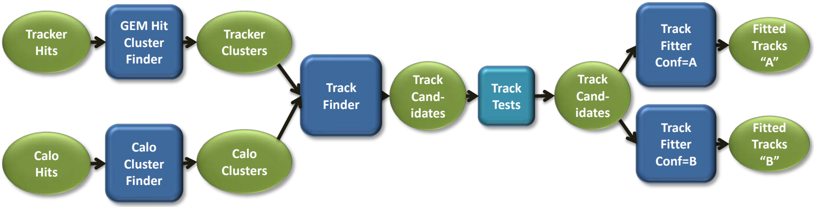
- Maximize consistency: Framework should support all of simulation, digitization, reconstruction and physics analysis
- Must support multi-pass processing: output → input for next pass
- Support multiple analysis chains per job, *e.g.*
  - Investigate different tracking or PID schemes
  - Run several physics analyses in parallel
- Interactive analysis must be possible with ROOT
- DSTs should contain extensive metadata, *e.g.*
  - Database parameters from previous stages (geometry etc.)
  - Data provenance

User-Written Components



- Data producers (algorithms)
  - Ideally, single algorithm per module
  - Run-time configurable
  - Must be reusable without recompilation → multiple instances allowed, differing in configuration
- Data objects (results)
  - transient or persistent
  - separate from producers
  - may reference other data objects
  - should hold metadata about their origin

Analysis Chains



- Modules communicate exclusively via data objects
- Module relationships configurable at run time by selecting from available compatible input data objects (by name, class, instance or similar)
- Support condition testing modules. Select subset of results and/or skip further processing if certain tests fail or succeed.
- Support multiple chains per job
- Output modules write user-configured subset of available data objects

Software Framework Comparison (preliminary)

Feature	art (FNAL)	FairRoot (GSI)	JANA (JLab)	Fun4All (PHENIX)
Origin	CMS	AliRoot (ALICE)	In-house	In-house
First release	2009	2004	2005	1998
Experiments using framework	~9	~10	1	1
Language	C++11/14	ROOT C++ (pre STL)	C++98	ROOT C++ (pre STL)
Base framework	self-contained	ROOT	self-contained	ROOT
Output, object persistency	ROOT	ROOT	HDDM	ROOT
ROOT 6 support	beta	no	n/a	no
Steering, configuration	FHiCL	ROOT macro	command line	ROOT macro
Reusable/multi-instance modules	yes	user	no	user
Multiple analysis chains	yes	yes	limited	yes
Automatic metadata, data provenance	partly	user	user	user
Test/filter modules	yes	user	user	user
Multithreading	no (planned)	no (unlikely)	yes (partial)	no (possible)
Installation dependencies	cet-is (3.5 GB)	FairSoft (2.8 GB)	Xerces XML	ROOT (500 MB)
Preferred installation	Binary via UPD	Source (GitHub)	Source (GitHub)	Source (GitHub)
Unit tests	425	39 (high-level)	0	0
User documentation	User Guide (500+ pages)	Examples, Wiki	Examples, Wiki, User Guide (old)	Examples
User code reusable for SoLID	little (DB, I/O)	much (Panda, EIC)	much (GlueX)	some (PHENIX)

Example FairRoot/EICRoot Script

From Alexander Kiselev's Sept 2015 EICRoot examples:

```
void reconstruction()
{
  // Load basic libraries;
  gROOT->Macro("$VMCWORKDIR/gconfig/rootlogon.C");

  // Create generic analysis run manager; configure it for track reconstruction;
  EicRunAna *fRun = new EicRunAna();
  fRun->SetInputFile ("simulation.root");
  fRun->AddFriend    ("digitization.root");
  fRun->SetOutputFile("reconstruction.root");

  // Call "ideal" hit-to-track associator routine;
  EicIdealTrackingCode* idealTracker = new EicIdealTrackingCode();
  idealTracker->AddDetectorGroup("FWDGT");
  // Add a bit of fairness to the reconstruction procedure; smear "ideal"
  // momenta by 10% relative before giving hit collection over to KF fitter;
  idealTracker->SetRelativeMomentumSmearing(0.1);
  // Also smear a bit "ideal" vertex;
  idealTracker->SetVertexSmearing(0.01, 0.01, 0.01);
  fRun->AddTask(idealTracker);

  // Invoke and configure PandaRoot Kalman filter code wrapper;
  fRun->AddTask(new EicRecoKalmanTask(idealTracker));

  // This call here just performs track backward propagation to the beam line;
  fRun->AddTask(new PndPidCorrelator());

  // Initialize and run the reconstruction; exit at the end;
  fRun->Run();
} // reconstruction()
```

Equivalent art FHiCL configuration file

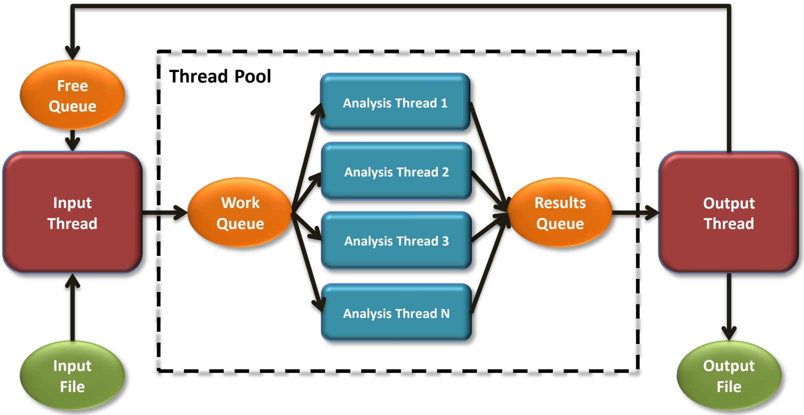
```
#include "fcl/minimalMessageService.fcl"

process_name : reconstruction

services : {
   message : @local::default_message
}

source : {
   module_type : FriendlyRootInput
   fileNames : [ "simulation.root" ]
   friendFileNames: [ "digitization.root" ]
}

outputs : {
   rootOut : {
      module_type : RootOutput
      fileName : "reconstruction.root"
   }
}

physics : {
   producers : {
      idealTracker : {
         module_type : IdealTrackingCode // Ideal hit-to-track association
         input : FWDGT                   // Consider only FWDGT clusters
         momentumSmearing : 0.1          // 10% momentum smearing
         vertexSmearing: [ 0.1, 0.1, 0.1 ]  // Vertex position smearing
      }
      recoKalman : {
         module_type : RecoKalman        // Kalman track fitter
         input : idealTracker            // using idealTracker clusters
      }
      pidCorrelator : {
         module_type : PidCorrelator
      }
   }
   reco_chain : [ idealTracker, recoKalman, pidCorrelator ]
   output_to_file : [ rootOut ]

   trigger_paths : [ reco_chain ]
   end_paths : [ output_to_file ]
}
```

Choosing A Computing Model

3 minute run → 18M SIDIS events, 50 GB raw data
Assume 20 ms/event → to keep up with data taking, need 2000 cores

- **Single-threaded:** no framework support for parallelism
  - 2000 runs in parallel → 100 TB disk space for input
  - ≈ 100 hours turn-around time per run
  - Problems: cost & turnaround time
- **Multi-process:** parallelism through the job scheduler
  - *E.g.* 32 single-threaded jobs working on different event ranges of one run
  - 62.5 runs in parallel → 3 TB disk space for input, 3 hours/run
  - Potential problems: I/O bottlenecks (disk head thrashing), limited scalability, complexity outsourced to job scheduler
- **Multi-threaded:** event-level parallelism through modern CPU architecture
  - Similar to multi-process, but reduced random disk access & memory footprint
  - Problems: scalability limited by cores/node, code complexity
- **Distributed:** event-level parallelism through built-in scheduler
  - 1 run in real time, 0.05 TB disk space for input.
  - Virtually unlimited scalability
  - Potential problems: even more code complexity, network bottlenecks

Possible Multi-Threaded Architecture



- Thread Pool with three thread-safe queues
- Queues hold working sets: event object, analysis chain & modules
- Option to sync event stream at certain events (*e.g.* scaler events, run boundaries)
- Option to preserve strict event ordering (at a performance penalty)

Conclusions

- A good number of suitable frameworks on the market
- Objective choice is difficult, at least on short timescale without local expertise
- Joint effort with EIC development would be beneficial if sufficient overlap and interest
- SoLID would be best served if we made a decision relatively soon and started porting and developing algorithms