Hall D Framework Overview

Ole Hansen

Jefferson Lab

SoLID Software Meeting
June 11, 2015

Framework Comparison (very preliminary)

Feature	Hall A Podd	Hall B CLARA	Hall D JANA/DANA	Phenix Fun4All
Language	C++	Java & C++	C++	C++
Base Package	ROOT	–	–	ROOT
Raw data format	EVIO	EVIO	EVIO	(non-EVIO)
DST format	ROOT	EVIO	REST (HDDM)	ROOT(?)
Configurable Output	yes		no	no
Database	Text	CCDB	CCDB, XML	
User Interface	CINT	Groovy	command line	CINT
Plugins	yes	yes(?)	yes	yes
Multi-threaded	soon	yes	yes	no(?)
Distributed	no	yes	no	yes(?)
Multi-stage analysis	no		yes	yes
Recalibration support	no	no	no	yes
Sim truth data API	yes		yes	yes
Event display	no	yes	yes	

Hall D Analysis Framework

- JANA
  - General data processing framework in C++. Static library and inline-compiled templated class headers
  - Standalone, i.e. not based on another data analysis framework like ROOT (some ROOT support available, e.g. writing ROOT output files)
  - Extensible via plugins
  - Multi-threaded event processing. Supports any kind of event data. Aside from event concept, basically no physics-specific features
- DANA
  - Large collection of precompiled Hall D-specific analysis classes
  - Several standard event processors provided (equivalent to standard analysis scripts), loadable as plugins
  - Command line interface. Typical user command:
    `hd_ana -PPLUGINS=danarest file.evio`

JANA Concepts (I)

- **Event sources** ("`JEventSource`")
  - Completely format-agnostic
  - Read events (whatever they are) from some sort of input (files, network, databases) into internal "event buffer" (roughly a processing queue)
  - Multiple event sources may be defined
- **Data Objects** ("`JObject`")
  - Data structures representing information of interest (e.g. hits, clusters, tracks, PID likelihoods etc.)
- **Factories** ("`JFactory`")
  - Algorithm classes
  - Templated with the type of data object they produce
  - Produce their data objects exactly once per event (unless persistence requested, then once per run)
  - Request input data from other factories
  - Lowest level data ultimately retrieved from event sources via dummy factories
  - Run in threads, operating on thread context data ("`JEventLoop`")

Example Factory

DFCALHit.h

```
class DFCALHit : public JObject {
public:
  JOBJECT_PUBLIC(DFCALHit);
  DFCALHit()
  int row;
  int column;
  float x, y, E, t, intOverPeak;
  ...
};
```

DFCALHit_factory.cc

```
// class DFCALHit_factory : public jana::JFactory<DFCALHit>

jerror_t DFCALHit_factory::evnt(JEventLoop *loop, int eventnumber) {
    /// Generate DFCALHit object for each DFCALDigiHit object.
    /// This is where the first set of calibration constants is applied
    vector<const DFCALDigiHit*> digihits;
    loop->Get(digihits);
    for (unsigned int i=0; i < digihits.size(); i++) {
        const DFCALDigiHit *digihit = digihits[i];
        ...
        DFCALHit *hit = new DFCALHit;
        hit->row    = digihit->row;
        hit->x      = pos.X();
        ...
        _data.push_back(hit);
     }
}
```

Data Retrieval

JEventLoop.h

```
template<class T> JFactory<T>* JEventLoop::Get(vector<const T*> &t, const char *tag) {
   /// Retrieve or generate the array of objects of
   /// type T for the curent event being processed
   string className(T::static_className());
   for(iter=factories.begin(); iter!=factories.end(); iter++) {
      if( className == (*iter)->GetDataClassName() )
         factory = (JFactory<T>*)(*iter);
      if( factory == NULL ) continue;
      if( !strcmp(factory->Tag(),tag) )  break;
      else factory=NULL;
   }
   if( factory->evnt_was_called() ) {
      factory->CopyFrom(t);
      return factory;
   }
   ... (Check if objects available from an event source)
   factory->Get(t);
   return factory;
}
```

JFactory.h

```
template<class T> jerror_t JFactory<T>::Get(vector<const T*> &d) {
   ...
   evnt(eventLoop, event_number);
   evnt_called = 1;
}
```

JANA Concepts (II)

- **Event processors** ("`JEventProcessor`")
  - Request top-level data from defined factories
  - Handle output
  - Several processors may be chained
  - `JEventProcessor` methods are called from threads (`JEventLoop::OneEvent`). User must handle locking!

Example Event Processor

JEventProcessor_danarest.cc

```
// class JEventProcessor_danarest : public jana::JEventProcessor

jerror_t JEventProcessor_danarest::evnt(JEventLoop *locEventLoop, int eventnumber) {
   // Ignore EPICS events
   vector<const DEPICSvalue*> locEPICSValues;
   locEventLoop->Get(locEPICSValues);
   if(!locEPICSValues.empty()) return NOERROR;
   // Write this event to the REST output stream.
   vector<const DEventWriterREST*> locEventWriterRESTVector;
   locEventLoop->Get(locEventWriterRESTVector);
   locEventWriterRESTVector[0]->Write_RESTEvent(locEventLoop, "");
}
```

DEventWriterREST.cc

```
// class DEventWriterREST : public JObject

bool DEventWriterREST::Write_RESTEvent(JEventLoop* locEventLoop, string locOutputFileNameSubString) const {
   vector<const DMCReaction*> reactions;
   locEventLoop->Get(reactions);
   vector<const DRFTime*> rftimes;
   locEventLoop->Get(rftimes);
   vector<const DFCALShower*> fcalshowers;
   locEventLoop->Get(fcalshowers);
   ...
   japp->WriteLock("RESTWriter");
   *(locRESTFilePointers.second) << locRecord;
   japp->Unlock("RESTWriter");
}
```

Impressions

- Likes
  - Very general concepts. Any data in, any data out.
  - Fine-grained control over analysis (at level of data objects)
  - Design encourages good structuring of algorithms
  - Analysis chain configures itself
  - Plugin support
  - Configuration parameters settable at run time
  - Decent multi-threading & database support
  - Very well commented code (in JANA, not DANA)
- Dislikes
  - Command line interface. No scripting, everything must be (re)compiled. Design lends itself to hardcoding.
  - Excessive reliance on templates. Design weaknesses affecting performance.
  - Convoluted callback logic
  - Difficult to handle multiple instances of a detector type efficiently (e.g. tracker planes)
  - No output queue. Output implementation largely left to user.
  - No test package
  - EVIO decoder is not configurable, hardcoded for Hall-D detectors