SoLID Software Framework

Ole Hansen

Jefferson Lab

SoLID Collaboration Meeting
September 11, 2015

Software Framework Planning

- Need specifications for (at least) each of
  - Simulation
  - Digitization
  - Databases
  - File formats
  - Reconstruction Framework
  - Calibrations
  - Physics Analysis
- Formed working group w/ Simulation and Reconstruction subgroups
  - Bi-weekly meetings over the summer
  - Considered Hall A/B/D/Phenix & EIC frameworks for ideas
  - Developing specifications
  - Computing document being drafted (Private `JeffersonLab/SoLID-docs-softspec` repository on GitHub)
- Intended to meet long term goals

Simulation & Digitization Wishlist

- Package: GEMC (Hall B). Required SoLID modifications
  - Add interface to SoLID database system
  - Store relevant database parameters & metadata in output
  - Ensure database consistency between simulation, digitization and reconstruction, esp. geometry
- Generators etc. → Seamus's talk, next
- Digitization
  - Separate, standalone package
  - Could run within reconstruction framework
  - Develop long-term implementation after reconstruction framework in place
  - Include trigger emulation and hardware-level digitization (*e.g.* ADC reading instead of amplitude etc.)
  - Write CODA (EVIO) output

Reconstruction Framework: Feature Wishlist I

- General: Based on ROOT, C++ throughout
- Try to combine best features of Hall A analyzer, Hall D framework and Phenix's Fun4All
- User experience
  - Scriptable user-interface (ROOT's interpreter)
  - Completely configurable at runtime
    - Dynamic configuration of experimental setup, expandable via plugins for new hardware/detectors
    - Flexible input sources and output formats
    - User-configurable output contents
  - Data represented by data objects (streamable ROOT classes), produced by data producers ("factories"). Variables directly accessible from ROOT prompt for interactive analysis.
  - File formats: ROOT (all data classes), EVIO (CODA-type data only)
  - Support multi-stage analysis: DST files supported as both input and output
  - Self-describing output: DSTs contain database parameters and metadata from previous stages

Reconstruction Framework: Feature Wishlist II

- Technical
  - User-transparent multithreading
  - Probably should require ROOT 6, C++11
  - Minimize other software dependencies
  - Options to sync event stream at special events (it e.g. helicity flips, scaler events), or to preserve strict event ordering
  - Optimize for low memory per core (trend for new compute nodes), *i.e.* share read-only data (parameters etc.) across threads
- Simulation support
  - Propagate and access MC truth data for certain data classes (if input comes from MC)
  - Option for substituting any input data with MC truth data
  - Support for mixing data and MC events

Impressions of Hall D Framework (JANA)

- 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.
  - No output queue. Output implementation largely left to user.
  - No test package
  - EVIO decoder is not configurable, hardcoded for Hall-D DAQ

Some Concepts Stolen From JANA

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

Data Objects: Toy Code

Hypothetical GEM Hit class

```
class GEMHit {
public:
  GEMHit() {}
protected:
  int strip;
  double position;
  double amplitude;
  ...
  ClassDef(GEMHit,1)  // maybe not needed
};

class GEMHits : public DataObject {
public:
  GEMHits( const string& name ) : DataObject(name) {}
  virtual const char* GetClassName() { return "GEMHits"; }
  const vector<GEMHit>& GetHits() { return hits; }
protected:
  vector<GEMHit> hits;
  ClassDef(GEMHits,1)
};

// Retrieval perhaps like so:

GEMHits* hitobj = Get<GEMHits>("plane1.hits");
const vector<GEMHit>& hits = hitobj->GetHits();
// Process "hits" ...
```

Database Specs (preliminary)

- Single database of simple key/value pairs, accessible via a generic API
- Values stored as strings. User must know type.
- Indexed by run number, with support for "variations"
- Complex information stored as a set of "core parameters" (geometry) or external references (field map)
- Support for version control/parameter history
- Flexible backends. Hall B's CCDB is the default backend.

Hall A Analyzer Database Example

Example Hall A DB File

```
[ 2015-02-01 14:30:00 ]
#-- Mapping
B.mwdc.planeconfig = u1 u1p x1 x1p v1 v1p
                     u2 x2 v2
                     u3 u3p x3 x3p v3 v3p

B.mwdc.cratemap =  3 6  21 1877 500 96
                   4 4  11 1877 500 96
                   4 17 24 1877 500 96

#-- Geometry
B.mwdc.nwires       = 200  # Default
B.mwdc.u1.nwires    = 141  # Fewer wires

B.mwdc.size         = 2.0  0.5  0.0
B.mwdc.x1.size      = 1.4  0.35 0.0

#-- Configuration
B.mwdc.u.maxmiss    = 5

#-- Calibrations
B.mwdc.x1.res       = 0.255

[ 2015-02-02 16:45:00 ]
# only changed parameters here ...
B.mwdc.x1.res       = 0.258
```

Podd 1.6+ Database Access

```
THaInterface.C:
  // Set up default DB at program start, may override
  THaDB* gHaDB = new THaFileDB( DB_DIR );

MyDetector::ReadDatabase( const TTimeStamp& date ) {
  DBRequest request[] = {
    { "planeconfig", &planeconfig, kString },
    { "MCdata",      &mc_data,     kInt,    0, 1 },
    { 0 }
  };
  Int_t err = LoadDB( date, request, fPrefix );
};
```

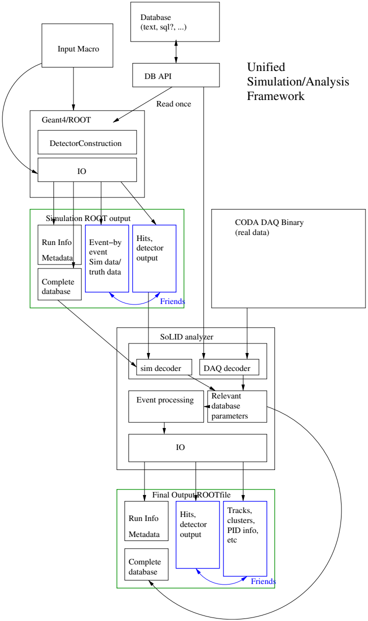
Unified simulation and analysis database API

General considerations

- Single common database abstraction
- Database holds "core parameters", esp. for geometry information
- Each component expands these parameters to a suitable internal representation
- Store sets of core parameters (for relevant run numbers) as objects in output files

Database requirements

- Should be easy to set up a local database
- DB should contain change history
- DB should support "variations" and local overrides of parameters
- Hall B's CCDB seems to fit the bill



Physics Analysis Scope, Specs (preliminary)

- Standard modules
  - Beam properties (position, helicity)
  - Calibrated detector data
  - Tracks, vertices
  - 4-vectors
  - PID, particle hypothesis likelihoods
  - Kinematical quantities for typical reactions
- Users may modify and extend provided methods (*e.g.* PID scheme, kinematics)
- Condition testing, event selection: Evaluate user-defined logical tests as input filter for each module and output (similar to Hall A analyzer test package)

Current Active Collaborators

- Ole Hansen, Alex Camsonne (JLab)
- Tom Hemmick, Seamus Riordan, Yuxiang Zhao (Stony Brook)
- Zhiwen Zhao, Zhihong Ye, Weizhi Xiong (Duke)
- Rich Holmes, Rakitha Beminiwattha (Syracuse)

Manpower Estimate

Task	Existing code (excl. ROOT)	FTE-years
Core reconstruction framework, ROOT file input and output, simulation APIs	Podd, JANA, Podd parallelization prototype code	7.5
Algorithms (tracking, calorimeter clustering, PID, physics analysis)	Various tracking prototypes, Hall D tracking, various clustering & PID implementations	12
Database API, backend, object streaming	Podd, CCDB	3
Decoders, EVIO input file support	Podd, JANA	4.5
Farm integration, testing, optimization	Halls B & D, JLab SciComp	1.5
Level 3 trigger	Hall D	3
Simulation integration (see Seamus's talk)	GEMC	9
Next-level simulation efforts & design iteration (see Seamus's talk)		4
Sum		44.5