SoLID Simulation Forward Issues

Seamus Riordan
Stony Brook University
seamus.riordan@stonybrook.edu

March 26, 2015

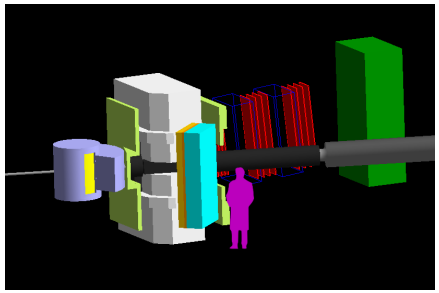
Personal Issues and Lessons Learned

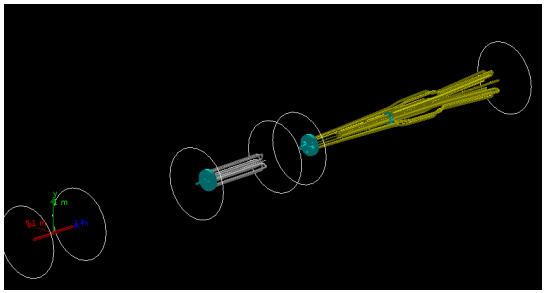
- mySQL server interface for general development arduous - need connectivity to read in geometry, get field map parameters, and decode results. Is a shared and largely uncontrolled space
- Storing floating point intermediately as text is a poor practice
- EVIO primary output - Present structure hasn't supported arrays, only been used as intermediate before getting translated to ROOT trees, and is the only option
- Has no intrinsic physics generators, no good method to handle weighted generators, asymmetries
- Building is intensely cumbersome and has wasted lots of time - many dependencies and environment variables required and are not optional (big packages like Qt and obscure separate packages like EVIO)
- Requests for features don't go to people responsible to collaboration, features in future versions don't meet our needed reponse time scales
- Our code spread across at least three repositories
- Methods to overcome these are workarounds and kludges

Personal Issues and Lessons Learned II

- Mixing inputs between command line (which can get really long), multiple text files, and SQL is messy
- Useful GUIs are critical
- Frequent, dramatic changes in code base are frustrating and obnoxious
- Small tweaks to code base and experimentation are very useful
- Having to require redudundant values from multiple sources and requiring "by hand" consistancy at various stages between multiple parties is a recipe for wasting time
- JLab is difficult to deal with about anything requiring external access and has a tendancy to not care about user needs outside of their firewall

Things I've learned from other projects



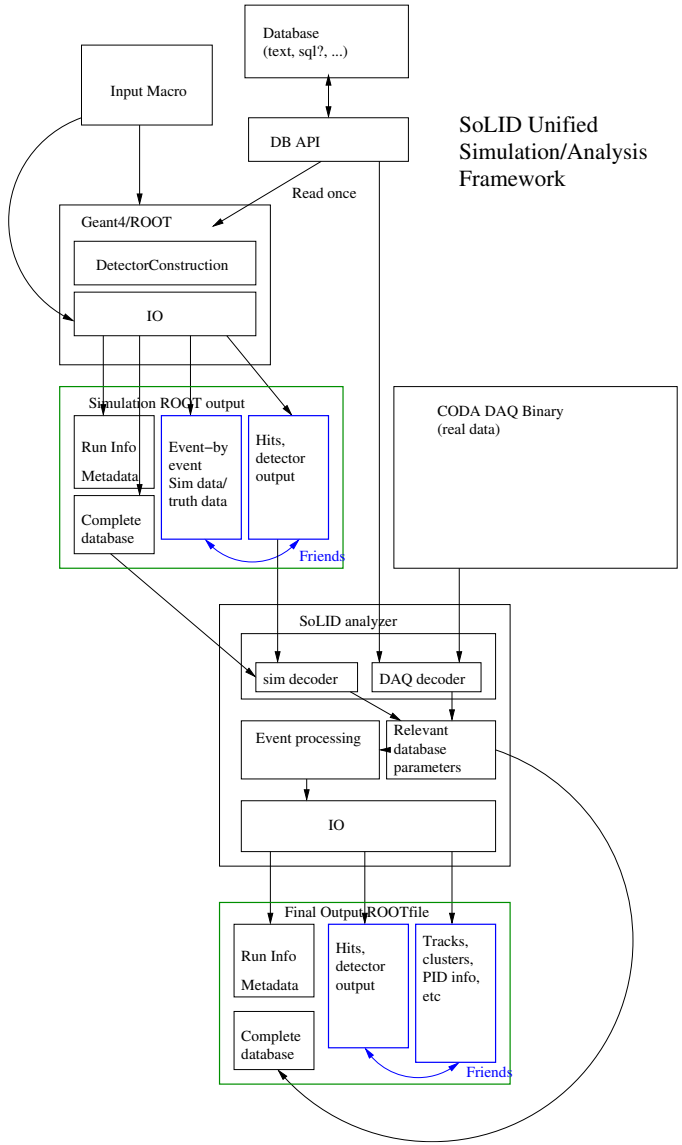


- Retrospection of all inputs, code base, and build environment has saved me lots of time and debugging frustration
- Simple build systems that allow optional packages and "just work" on anything with Geant4 and ROOT installed saves me a lot of time supporting other users
- Structuring general geometries with a few (or more) numerical inputs is not necessarily "bad", especially for lots of small variations on a similar theme
- GDML is a pain to work with for complex geometries
- Users will force you to work across lots of different environments and don't typically have the skills to solve build problems on their own and frequently can't install dependencies themselves
- git is a really great versioning system for this type of work - github managed JLab repositories I think have been very successful
- CMake is widely installed but can be a bit esoteric for complicated situations, however so far has only required one knowledgable person and "just works"

Framework Issues and Wish Lists

Things that I would like in a simulation

- Being able to run locally and out of the box - no required external servers, no offsite servers (but as optional is probably desirable)
- Can build and run with minimal external libraries
- Output goes (at least) directly to ROOT, but also must be readable as pseudodata into analysis framework
- All input (especially geometry configuration) is saved with output, is immutable, and in a "comparable" way
- No redundant inputs (or at least minimized)
- Analysis framework can get all it needs to know about how to handle simulation from the simulation output
- Digitization stages are optional, maybe even a separate analysis stage



Issues to be considered

- Common databasing between simulation and analysis framework
  - Need to examine potential analysis frameworks - likely to drive what we want for a simulation
  - Robust SQL is important, text files do have their advantages, abstracting to multiple systems with a common storage class/format might be something to shoot for (makes many "practical" problems external to software development)
  - Databasing tools will need to be developed with this
- How to incorporate generators
- How to "mix" events
  - Often want combined results from many single electrons through target, or that with different generator

Issues to be considered - II

- Input minimization - scheme needs to be carefully designed
- How geometry is handled - some get very complicated (see baffles)
- Output standardization - scheme needs to be carefully designed
- Digitization handling
  - Separate packages or optional flags?
  - How far does handling go?
  - Keep analysis framework "agnostic"?
- GUI and response visualization development
- Repository and code management
- Do we fork something for collaboration control if we use any existing framework?

Generators

Generators are available for lots of processes
Not sure if this is an issue for now

- Prevertex external radiation, MS included
- Moller
- *ep* elastic
  - Internal radiation included
- DIS with A_{PV}
- Inelastic (Christy/Bosted)
  - Ported to C, could include A_{PV}
- $\pi^{+/-}$
  - Wiser fits with EPA and Tiator/Wright
- Hyperons forthcoming
- Pythia and Hall D generators will be important

FIN

Recreation

```
root [1] .ls
    TFile**                 remollout_ep_6.root
    TFile*                  remollout_ep_6.root
    KEY: TTree  T;1         Geant4 Moller Simulation
    KEY: remollRunData  run_data;1
    root [2] run_data->Print()

    git repository info
    ----------------------------------------
    commit 8a5fff7008edeb3543c7899da86c67bbe5eb1856
    Merge: 818f894 4365e2d
    Author: Seamus Riordan <sriordan@physics.umass.edu>
    Date:   Tue Mar 5 14:19:07 2013 -0500

    Merge branch 'master' into optical
    ## optical
```

Recreation

```
git checkout 8a5fff7008edeb3543c7899da86c67bbe5eb1856
```

will restore to that version
To extract built GDML files

```
root [3] run_data->RecreateGDML()
```

or

```
root [3] run_data->RecreateGDML("new path")
```