Python Bindings to CLAS Libraries

A Case Study for GlueX

Craig Bookwalter[1]

[1]Department of Physics
Florida State University

November 16, 2010

Why should I use a scripting language?

Faster development time

- Less lines of code
- Better error messages
- Rich standard libraries

Why should I use Python?

- Broad support in scientific community $\rightarrow$ lots of solid and useful 3rd-party libraries
- (Somewhat) easy to wrap C/C++ functions
- Syntax is easy to learn for C programmers
- Parallelization is easy (process-based)

CLAS Data Structure

- Uses BOS format (Bank Object System)
- Data stored in banks named by four-letter words
- Banks are loaded into a Fortran common block on demand

Example of BOS I/O (in C)

```
{
  clasPART_t* PART(NULL);

  while (1) {
    dropAllBanks(&bcs_,"E");
    cleanBanks(&bcs_);

    if (!(getBOS(&bcs_,1,"E")))
      break;

    if (!(PART = (clasPART_t*)getBank(&bcs_,"PART")))
      continue;

    int i;
    for (i=0; i < PART->bank.nrow; i++)
      printf("%d %.2f %.2f %.2f %.2f", PART->part[i].pid, PART->part[i].px,
             PART->part[i].py, PART->part[i].pz, PART->part[i].e);
  }
}
```

A Python interface to BOS

- A collection of C++ classes that do the heavy lifting
- Python wrappers generated automagically by SWIG:

```
$] swig -python -c++ -o clas/bos/cppbosint_wrap.cpp clas/bos/cppbosint.i
```

- A C++ interface class to closely control types that SWIG needs to map from Python to C++ and back (`cppbosint.hpp`)
- Object-oriented interface on the Python side.
- Parsing the string done on the Python side
- As fast as `c_bos_io` for simple things like counting events

A Python interface to BOS

```
import clas.bos

bf = clas.bos.open_file("example.bos")

for event in bf:
    PART = event.get_bank("PART", 1)
    for row in PART:
        print row.pid, row.px, row.py, row.pz, row.e
```

```
%module cppbosint
%include "std_string.i"

%{
  #include "cppbosint.hpp"
%}

%include "cppbosint.hpp"
```

A Python interface to the ROOT 4-vector classes

- I don't like ROOT, and I needed 4-vector math in Python, and (surprisingly) there is no 3rd-party package.
- So I wrapped the ROOT 4-vector classes myself, again using SWIG (8 hours of work for `TVector2`, `TVector3`, `TLorentzVector`, `TRotation`, and `TLorentzRotation`)
- No interface classes or anything like that – API is identical to ROOT
- 80% of the work is just figuring out what to tell SWIG to ignore

Looking forward

Design decisions

- the Python wrapper for BOS only reads data from disk–it cannot access reconstruction code to regenerate banks on the fly
- for GlueX it's possible to wrap *all* of the classes and have access to the full reconstruction chain from Python scripts or `ipython`...
- ...but what is the payoff?

Things SWIG doesn't like (that I know of)

- Passing around C arrays or `const char*`
- Setting member variables by value
- Passing/returning aggregate types by value

Conclusions

- GlueX should have some way to access data from Python
- It's possible to wrap all the way down the reconstruction chain as well, but may not be worth the work