Premise: We're Not Moving the C++/Python Boundary

So what we want is a way to map our natural C++ interfaces to natural Python interfaces. What are our options?

(I'm not saying this *is* what we want.*)

(*well, I think it is what I want, but I'm also trying to keep an open mind.)

Living With Swig

We all hate it. It's not just weirdos who use CRTP.

- It's extremely hard to debug problems, especially for new users – most problems happen in the parts we can't step through: the parser and code generator, and its most common failure modes are not well documented and have poor error messages (e.g. the order of declarations).
- Some of the memory management is dangerous (dangling pointers).
- The extension/customization language has all the elegance and readability of the C preprocessor.
- Build times and module sizes are huge – we can almost certainly do better (especially if we throw money at Fulton), but we don't know how much better (or how much effort it will take).
- Swig's biggest problems are deeply architectural. It's not going to get better at any of the above, *ever.*

But...

- It's the best tool if we want to comprehensively wrap our C++ codebase with minimal additional effort, as long as we don't care how natural the Python interface is.
- We could make the resulting interface Pythonic with either another pure-Python layer on top or a lot of effort in the .i files (or some combination thereof).

Adopting Boost.Python

And I do mean *adopt:*

- Boost.Python has a large user base, but essentially no ongoing support or active development – even simple patches don't get in because there's no one really working to accept them (and *I'm* still the administrator of the main mailing list!)
- To put together a complete system that uses Boost.Python, you need to get the STL and NumPy connectors from other stale repos (well, the NumPy one isn't stale, but that's because I maintain it!). The best code generator (Py++) is similarly stale.

But...

- It's a very well thought-out, high-quality library, written by very smart people (same for Py++).
- It's already almost feature complete (with the add-ins mentioned above); all that's missing is C++11 standard library support (the architecture means C++11 *language* features are essentially automatically supported).
- If we *want* to hand-craft a Python interface while starting from a similarly natural C++ interface, Boost.Python is *easily* the most concise, readable way to do that.

Considering Cython

I've got almost no experience with Cython, but what I've seen did not leave a good impression.

- It's making several of the same big architectural mistakes as Swig:
  - Inventing a new pseudo-language (pseudo-Python instead of pseudo C)
  - Putting the most important logic in a code generator you can't step through to debug.
  - Trying to reimplement the C++ type system instead of using RTTI.
- It's just not aimed at dealing with complex C++ libraries (on its own).

But...

- It's really gathering mindshare in the scientific Python world.
- Maybe with something like https://github.com/xdress/xdress it could handle a complex C++ library.

My Open-Secret Home-Grown One-Weekend-a-Month Someday-it-Might-Get-Done Here's-How-it-*Should*-Be-Done Alternative

- I actually think I know exactly what kind of tool we want, and I've been writing it very slowly over the past three years.
- It's reached the point where it can actually benefit from more developers working it simultaneously.
- The slow pace of development so far is indicative of how little time I've been able to put into it, not how hard it is to build.

If what we want to do is map a natural C++ API to a natural Python API, I think this is worth a serious look, at least to the level of trying to scope out how much effort it would actually take (and how that compares to the money/effort we'd otherwise spend on fixing Swig's problems or taking ownership of Boost.Python or working around Cython's limited C++ support).

https://github.com/TallJimbo/mcpib