Interfacing Python and C++

Jonathan Riehl
Department of Computer Science
University of Chicago

http://people.cs.uchicago.edu/~jriehl/notes.html

Motivations

- Can change program behavior.
  - At import time (skipping re-compilation).
  - At run time.
- Configuration language.
- Rapid prototyping.
  - No static type declarations.
  - Simpler language for non-programmers.

Approaches

- Need a means of translating values from their C/C++ representation to Python, and from Python back to C/C++...
- By hand...
  - Using the Python C API.
  - Using the ctypes module.
- Using a wrapper generator...
  - SWIG

Using the Python C API...

- Python wrappers are created in your C/C++ program.
- By following a specific API they can be bound in the interpreter.
  - static PyMethodDef methods [] = {...};
  - Py_InitModule(“extmodule”, methods);
- Can create a .so that Python treats as a native module.
- http://docs.python.org/ext/ext.html

Example of Python C API Extension

```
#include <Python.h>

extern int myfunc (int x, const char * s1);

static PyObject *
wrapped_myfunc(PyObject *self, PyObject *args)
{
    const char *s1;
    int ival;

    if (!PyArg_ParseTuple(args, "is", &ival, &s1))
        return NULL;
    ival = myfunc(x, s1);
    return Py_BuildValue("i", ival);
}
```

Calling Python from C

```
static PyObject * my_callback = NULL;

long call_my_callback (int arg)
{
  long result = -1L;
  PyObject * pyresult = NULL;
  if (my_callback != NULL &&
      PyCallable_Check(temp))
   {
     arglist = Py_BuildValue("(i)", arg);
     pyresult = PyEval_CallObject(my_callback,
                                  arglist);
     Py_DECREF(arglist);
     result = (int)PyInt_AsLong(pyresult);
     Py_DECREF(result);
   }
  return result;
}
```

Using the ctypes module...

- Wraps parts of the Python C API.
- Still need to get initial C/C++ values into the interpreter.
  - dlopen(), dlsym()
  - Casting hacks.
- Two modes of operation:
  - Default: do the sensible thing.
  - Optional type and arity checking.
- http://docs.python.org/lib/module-ctypes.html

Example of ctypes Extension

```
>>> import ctypes
>>> mydll = ctypes.CDLL("./mydll.dll")
>>> myfunc = mydll.myfunc
>>> myfunc.argtypes = [ctypes.c_int, ctypes.c_char_p]
>>> myfunc.restype = ctypes.c_int
>>> print mydll.myfunc(22, "This is a test.")
Input: 22 'This is a test.'
12
>>>
```

Using SWIG...

- Simplified Wrapper Interface Generator
- Inputs:
  - An interface file, “XXX.i”
- Outputs:
  - A C++ interface file, “XXX_wrap.cxx”, used to build extension module, “_XXX.so”.
  - A Python interface file, “XXX.py”, which adds Python “shadow classes” and other high-level wrapping.
- http://www.swig.org/

Example SWIG Interface File

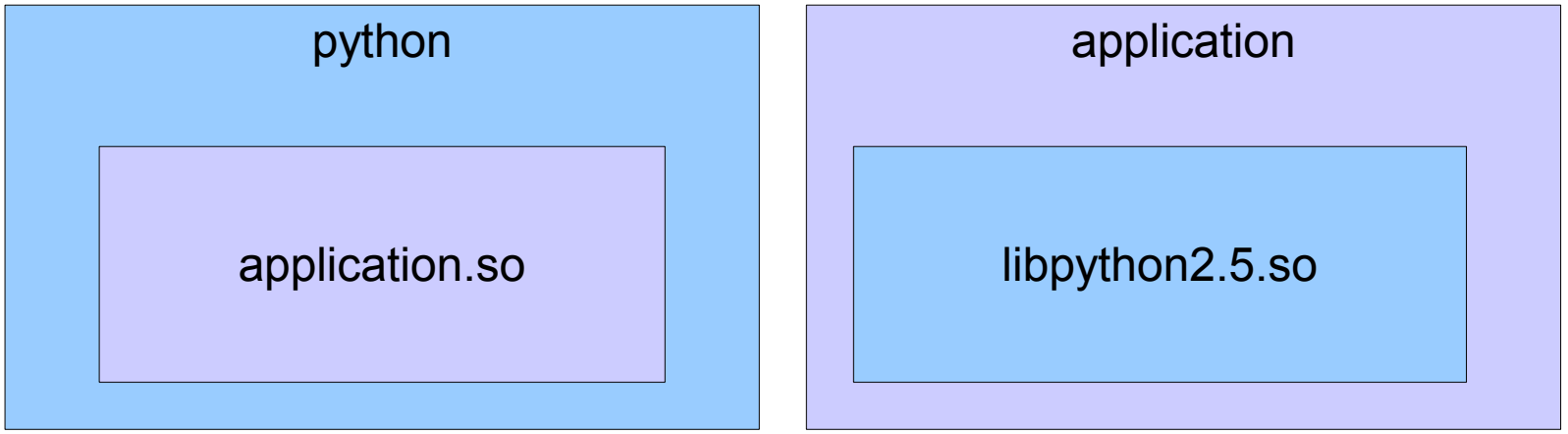
```
%module example

%{
/* Includes the header in the wrapper code */
#include "header.h"
%}

/* Parse the header file to generate wrappers */
%include "header.h"
```

Linkage

- Python in C/C++
  - gcc -o application -lpython2.5
- C/C++ in Python
  - gcc -shared -o application.so -lpython2.5



Demo

- Basic GLUT based simulation with C++ “actors”. Important members:
  - react(t)
  - render()
  - std::list<Actor*> actors
- Two threads:
  - C++ GLUT Application
  - Python interpreter
- SWIG used to generate an interface to the “actors”.