

# pyGlobus: Python CoG Kit

Keith R. Jackson  
krjackson@lbl.gov

# Overview

- Why use pyGlobus?
- pyGlobus architecture
- pyGlobus status
- Compilation and installation
- Simple programming examples
- pyOGSI overview
- pyOGSI client binding
- WS-Security
- pyOGSI server support
- Future plans
- Acknowledgements

# What is the Python CoG Kit?

- The Python CoG Kit provides a mapping between Python and the Globus Toolkit®. It extends the use of Globus by enabling to access advanced Python features such as exceptions and objects for Grid programming.
- The Python CoG Kit is implemented as a series of Python extension modules that wrap the Globus C code.
- Uses SWIG (<http://www.swig.org>) to help generate the interfaces.

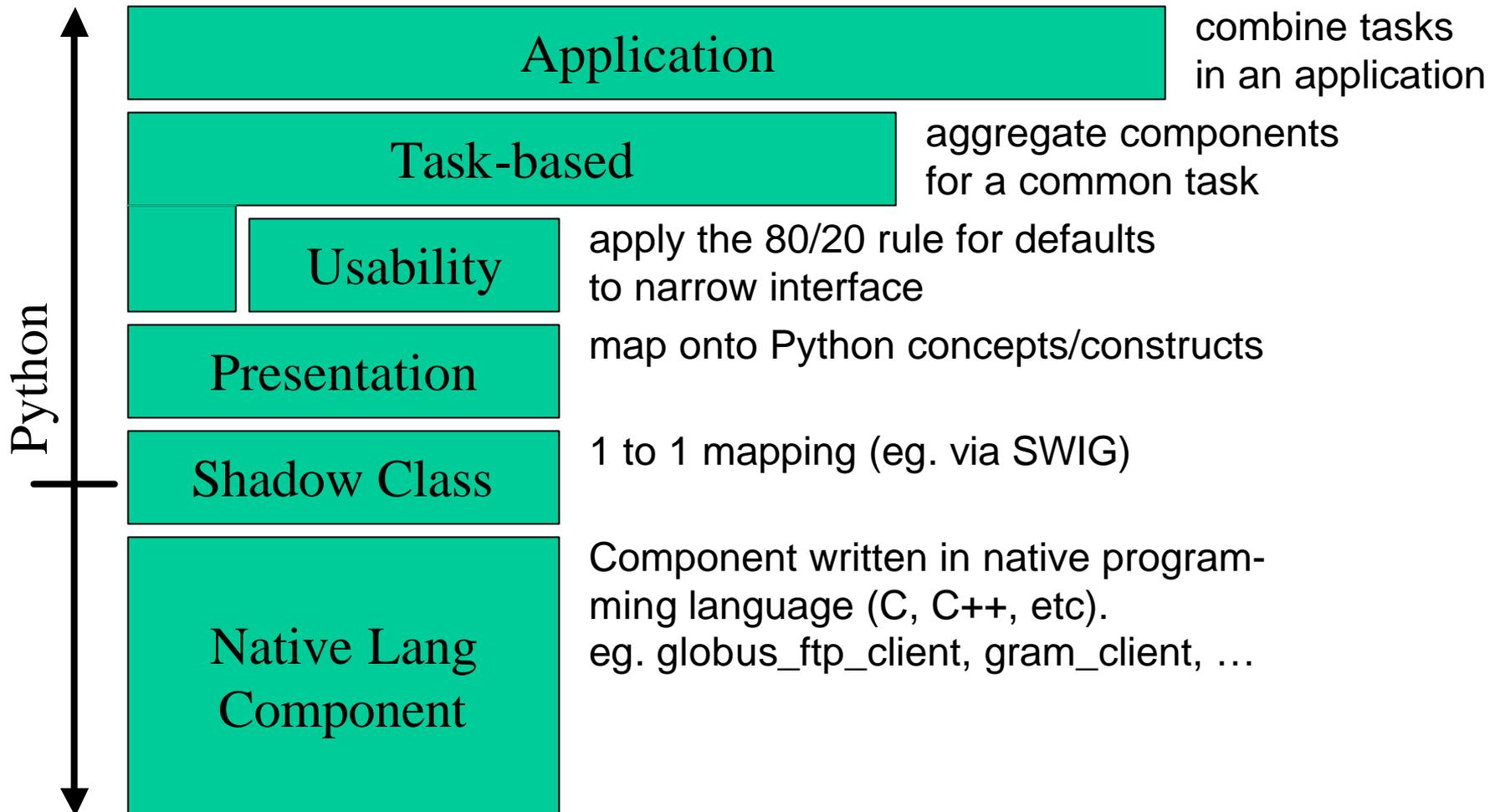
# Reasons for Using the Python CoG Kit

- Why use the Python CoG Kit?
  - Provides a full interface to the Globus Toolkit
  - Little/no changes are involved in switching between different versions of the Globus Toolkit
  - High level language allows for easier Grid programming
    - Supports rapid prototyping of Grid services/applications
  - Many automated tools exist for exposing legacy C/C++ or Fortran codes as Python objects
- Why not use the Python CoG Kit
  - Small performance penalty for using any interpreted language
    - Minimized because the Python CoG Kit is a thin wrapper over the native C code.
  - No static type checking

# Motivation: Python CoG Kit

- Use and leverage existing technologies for Grid programming
  - The capabilities of the framework onto which Grid Services are mapped can be exploited:
    - Objects, Events, Exceptions, ...
  - Objects like jobs/tasks can be defined.
  - XML support is provided.
  - GUI's, ....., IDE's can be used (IDLE, BOA Constructor...)
- Maximize software flexibility, extensibility, and reusability
- Provide foundations for application developer teams that are familiar to develop applications in this framework
  - Reduce development and maintenance cost
- Use as glue for many technologies
  - Python is well suited to tying together many different languages/technologies

# pyGlobus Architecture



# Status: Python CoG Kit

- Basic services are provided accessing:
  - Security (security)
  - Remote job submission and monitoring (gramClient)
  - Secure high-performance network IO (io)
  - Protocol independent data transfers (gassCopy)
  - High performance Grid FTP transfers (ftpClient)
  - Support for building Grid FTP servers (ftpControl)
  - Support for building Gass servers (gassTransfer)
  - Information Services access
- High level services for easier usage
- Task based services to encapsulate common usage patterns

# Python CoG Kit Setup Options

- Download source
  - Set Environment variables
  - Compile
  - Install
- Easy to create RPM's for Linux systems
- Binary installer coming for win32
  - As soon as Globus officially releases the win32 port
- Can build binary packages for any platform
- Uses the standard Python distutil module

# Requirements

- Python CoG
  - Python 2.0 +
    - <http://www.python.org>
  - Globus Toolkit Installation
  - GPT Installation

# Python CoG Kit Compilation

- Ensure that GPT\_LOCATION and GLOBUS\_LOCATION are set appropriately
- `cd pyGlobus-{Version}`
- `python setup.py build`
  - `--prefix=/path/to/installation/directory`
  - Will only build those packages that you have Globus installs of
- `python setup.py install`
  - To install in the site-extensions directory requires root privilege

# Examples

- Python examples:
  - Basic examples are in
    - pyGlobus/examples
  - Test directories contain the unittest code that provide more advanced examples

# Python Job Submission Example

- Creating a job.

```
try:
    gramClient = GramClient.GramClient()
    callbackContact = gramClient.set_callback(func, condV)
    jobContact =
    gramClient.submit_request("clipper.lbl.gov",
        "&(executable=/bin/sleep)(argument=15)",
        GramClient.JOB_STATE_ALL)
except GramClient.GramClientException, ex:
    print ex.msg
```

- Callback for state changes.

```
def func(cv, contact, state, error):
    if state == GramClient.JOB_STATE_PENDING:
        print "Job is pending"
    elif state == GramClient.JOB_STATE_ACTIVE:
        print "Job is active"
```

# Redirecting stdout with gramClient

```
from pyGlobus import gassServerEZ
opts = gassServerEZ.STDOUT_ENABLE
server = gassServerEZ.GassServerEZ(opts)
url = server.getURL()
rsi =
    "&(executable=/bin/sleep)(arguments=15)
      (stdout=%s/dev/stdout)" % url
```

# Compare:

## C Job Submission Example

```
callback_func(void *user_arg, char *job_contact,
              int state, int errorcode)
{
    globus_i_globusrun_gram_monitor_t *monitor;
    monitor = (globus_i_globusrun_gram_monitor_t *) user_arg;
    globus_mutex_lock(&monitor->mutex);
    monitor->job_state = state;
    switch(state)
    {
        case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
        {
            globus_i_globusrun_gram_monitor_t *monitor;
            monitor = (globus_i_globusrun_gram_monitor_t *) user_arg;
            globus_mutex_lock(&monitor->mutex);
            monitor->job_state = state;
            switch(state)
            {
                case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
                    if(monitor->verbose)
                    {
                        globus_libc_printf("GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED\n");
                    }
                    monitor->done = GLOBUS_TRUE;
                    break;
                case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
                    if(monitor->verbose)
                    {
                        globus_libc_printf("GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE\n");
                    }
                    monitor->done = GLOBUS_TRUE;
                    break;
            }
        }
        globus_cond_signal(&monitor->cond);
        globus_mutex_unlock(&monitor->mutex);
    }
}
```

# Compare:

## C Job Submission Example (cont.)

```
globus_l_globusrun_gramrun(char * request_string, unsigned long options, char *rm_contact){
    char *callback_contact = GLOBUS_NULL;
    char *job_contact = GLOBUS_NULL;
    globus_i_globusrun_gram_monitor_t monitor;
    int err;
    monitor.done = GLOBUS_FALSE;
    monitor.verbose=verbose;
    globus_mutex_init(&monitor.mutex, GLOBUS_NULL);
    globus_cond_init(&monitor.cond, GLOBUS_NULL);

    err = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if(err != GLOBUS_SUCCESS)
    { ... }
    err = globus_gram_client_callback_allow(
        globus_l_globusrun_gram_callback_func,
        (void *) &monitor,
        &callback_contact);
    if(err != GLOBUS_SUCCESS)
    { ... }
    err = globus_gram_client_job_request(rm_contact,
        request_string, GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
        callback_contact, &job_contact);
    if(err != GLOBUS_SUCCESS)
    { ... }
    globus_mutex_lock(&monitor.mutex);
    while(!monitor.done) {
        globus_cond_wait(&monitor.cond, &monitor.mutex);
    }
    globus_mutex_unlock(&monitor.mutex);
    globus_gram_client_callback_disallow(callback_contact);
    globus_free(callback_contact);

    globus_mutex_destroy(&monitor.mutex);
    globus_cond_destroy(&monitor.cond);
}
```

# Python GridFTP Example

```
from pyGlobus import ftpClient
from pyGlobus.util import Buffer
handleAttr = ftpClient.HandleAttr()
opAttr = ftpClient.OperationAttr()
marker = ftpClient.RestartMarker()
ftpCInt = ftpClient.FtpClient(handleAttr)
ftpCInt.get(url, opAttr, marker, done_func, condV)
buf = Buffer(64*1024)
handle = ftpCInt.register_read(buf, data_func, 0)

def data_func(cv, handle, buffer, bufHandle, bufLen,
             offset, eof, error):
    g_dest.write(buffer)
    if not eof:
        try:
            handle = g_ftpClient.register_read(g_buffer,
            data_func, 0)
        except Exception, e:
```

# Performance Options for GridFTP

- Setting tcpbuffer size

```
from pyGlobus import ftpControl
battr = ftpControl.TcpBuffer()
battr.set_fixed(64 * 1024)
```

Or

```
battr.set_automatic(16 * 1024, 8 * 1024,
    64 * 1024)
```

```
opAttr.set_tcp_buffer(battr)
```

- Setting parallelism

```
para = ftpControl.Parallelism()
para.set_mode(ftpControl.PARALLELISM_FIXED)
para.set_size(3)
opAttr.set_parallelism(para)
```

# Python GassCopy

- Provides a protocol independent API to transfer remote files.

```
srcAttr      = GassCopyAttr()
handleAttr   = GassCopyHandleAttr()
destAttr     = GassCopyAttr()
ftpSrcAttr   = FtpOperationAttr()
ftpDestAttr  = FtpOperationAttr()
srcAttr.set_ftp(ftpSrcAttr)
destAttr.set_ftp(ftpDestAttr)
copy = GassCopy(handleAttr)
copy.copy_url_to_url(srcUrl, srcAttr, destUrl, destAttr)
```

# pyOGSI

- Developing a full Open Grid Services Architecture implementation
- ZSI library is used for SOAP parsing
- WebWare application server for the hosting environment
  - Also support standalone Grid Services

# pyOGSI Client Bindings

- Added support to ZSI for automated binding generation
  - wsdl2python command line utility
  - Also support dynamic invocation
    - Performance overhead due to use of “eval” function
- Added Schema parsing to support automatic encoding of complex types
  - Creates Python classes

# pyOGSI Security

- Support normal TLS protocol (https)
- Support GSI enable TLS protocol (httpg)
- Adding support for WS-Security message level security
  - Implementing XML-DSig and XML-Encryption
  - Implementing WS-Secure Conversation to interoperate with the Java OGSI code

# pyOGSI Server Support

- Support both a OGSI “hosting environment” and standalone OGSI servers
- Standalone server will:
  - Use the standard Python SimpleHTTPServer
  - Provide a super-class that contains all of the necessary Grid Service code
    - Lifecycle management
    - Service Data/Notification
    - Security
    - Factory

# pyOGSI Server Support (cont.)

- Will provide a container to host Grid Services
  - Based on the WebWare project (<http://webware.sourceforge.net>)
  - Provides a base-class to encapsulate all required Grid Service functionality
  - Will support exposing legacy Fortran/C/C++ codes as Python OGSI components
  - Will provide automatic server stub generation from WSDL/GSDL document
    - Automatic support for generating GSDL from WSDL

# pyOGSI Status

- Client bindings currently available from CVS
  - Working to integrate the common Web Service code back into the ZSI project
  - Still working on the border cases with complex type encoding
- Server side support is under development
  - Working on automated server stub generation
  - Working on WSDL to GSDL decoration tool
  - Currently have working Web Service code, working on adding required Grid Service port types

# Future Plans

- What happens to pyGlobus when everything is OGSA based?
  - We will continue to support our higher-level interfaces where possible.
    - GRAM
    - GridFTP
    - Task based layer
- New pyGlobus release to support GT2.4
  - Next two weeks
  - Will continue tracking GT releases
  - Will continue to build “task based” interfaces as needed by users

## Future Plans (cont.)

- Plan to release a first version of the client binding by mid-May
  - Will not yet support WS-Security
- WS-Security code by mid-June
- Tentative plans for server side release in July
  - Will only have required Grid Service port types
  - Future work will add the other Grid Service port types

# Acknowledgement

- The Python CoG Kit is funded by the U.S. Department of Energy Office of Science
- More information can be found at
  - <http://www.cogkits.org>
  - <http://www-itg.lbl.gov/gtg/projects/pyGlobus/>
  - <http://www-itg.lbl.gov/gtg/projects/pyOGSI/>
- Bug submission
  - <http://www-itg.lbl.gov/bugzilla/>
- Email:
  - [krjackson@lbl.gov](mailto:krjackson@lbl.gov)