Simplified Grid Application Development in Python

Keith Jackson and David Konerding

Lawrence Berkeley National Laboratory

Overview

- Globus Project and Toolkit
- Python Grid Project Goals
- Why Python?
- Pre-Web Service Python Grid Support
- Web service/WS-RF Python Support
- SAGA (Simplified API for Grid Applications)
- Visual Programming

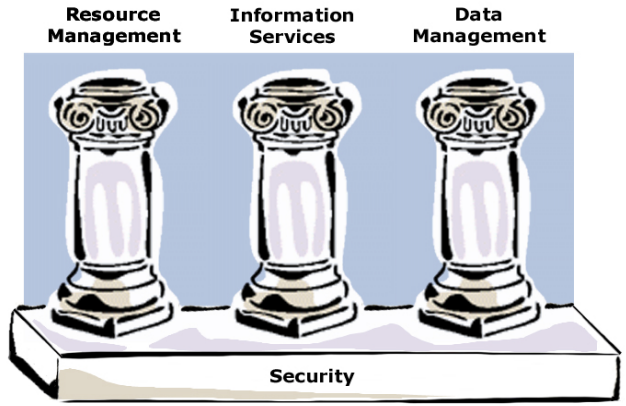
Globus Project™

Globus Project™

- The Globus Toolkit® is an open source software base for building Grid middleware and applications
  - ANL and ISI
  - Implementation began in 1996
- Developed standard protocols and APIs to enable interoperability, portability, and shared infrastructure
- Global Grid Forum provides a venue for Grid standardization

GT2 Key Protocols

- The Globus Toolkit v2 (GT2) centers around four key protocols
  - Connectivity layer:
    - *Security*: Grid Security Infrastructure (GSI)
  - Resource layer:
    - *Resource Management*: Grid Resource Allocation Management (GRAM)
    - *Information Services*: Grid Resource Information Protocol (GRIP)
    - *Data Transfer*: Grid File Transfer Protocol (GridFTP)
- Also key collective layer protocols
  - Info Services, Replica Management, etc.



Courtesy of the Globus Project™

Python Grid Project

Python Grid Project Goals

- Provide a high-level object-oriented software development kit for building Collaborative and Grid applications
- Provide tools to scientists in environments that they are familiar with.
- Support the rapid prototyping of Grid and Collaborative applications.
- Support the ability to integrate legacy codes easily into Grid applications

Why Python?

- Easy to learn/read high-level scripting language
  - Very little syntax
  - Straightforward object orientation
- A large collection of modules to support common operations, e.g., networking, HTTP, SMTP, LDAP, XML, Web Services, etc.
- Excellent for “gluing” together existing codes
  - Many automated tools for interfacing with C/C++/Fortran (SWIG and f2py are most popular)
- Support for platform independent GUI components
- Runs on all popular operating systems, e.g., UNIX, Win32, MacOS

Pre-Web Services Python Grid Toolkit

Pre-Web Services refers to GT2.x and pre-WS GT3.x components

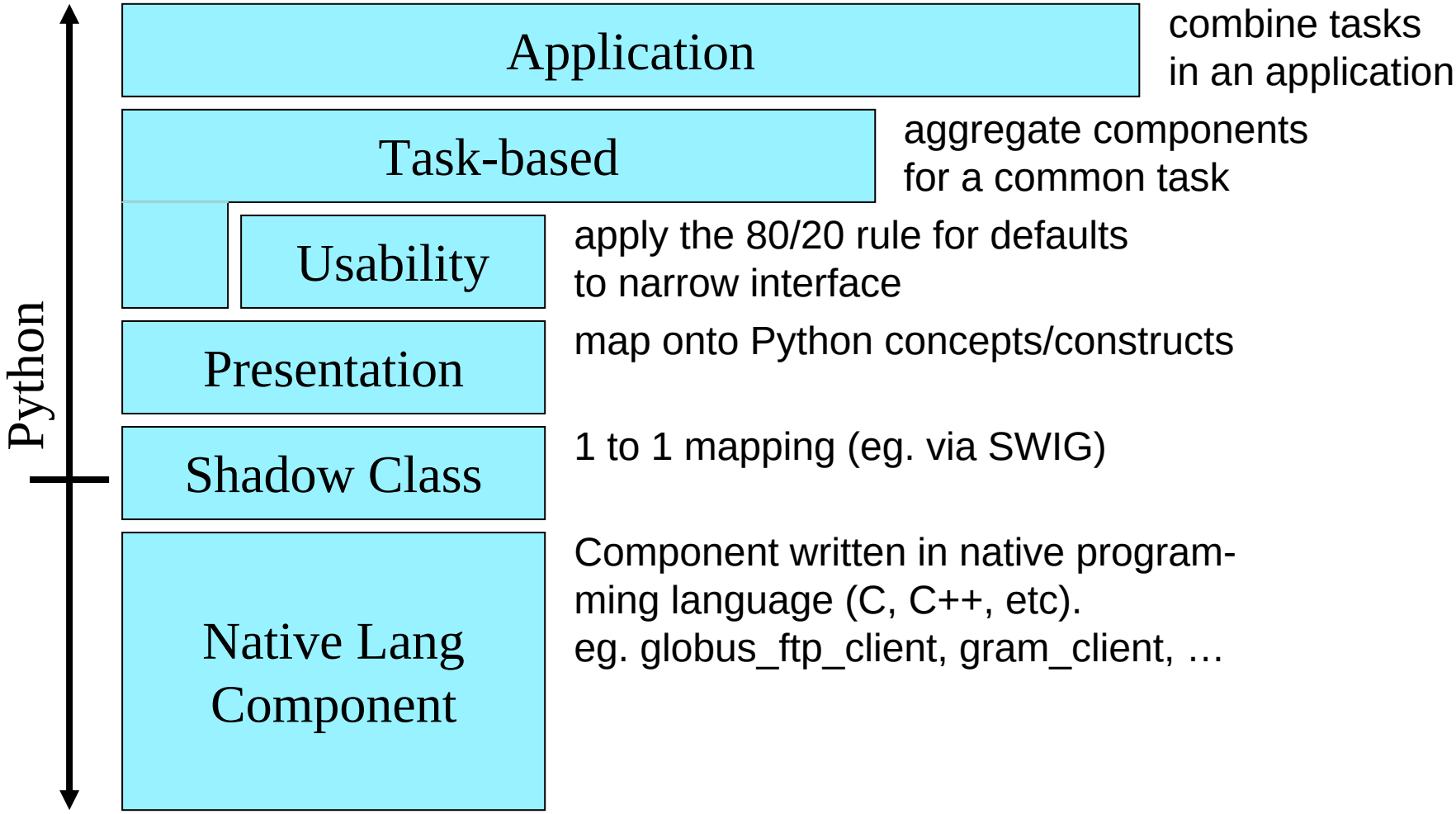
pyGlobus Technical Approach

- Wrap the existing Globus Toolkit code.
  - The Python CoG Kit (pyGlobus) is a series of Python extension modules and higher-level abstractions.
  - Uses SWIG (http://www.swig.org) to automatically generate wrappers around GT2.
    - Minimizes changes to the Python CoG Kit when GT changes.
    - Isolates end users from these changes.
- Map GT abstractions into Python constructs.
  - Globus errors are mapped to pyGlobus exceptions.
  - Data structures with many associated functions become classes with methods

pyGlobus Technical Approach

- Reduce tedious and error prone programming by hiding much of the complexity of GT2 behind object-oriented abstractions.
  - hide the underlying memory management
  - automate module activation/deactivation.
- Work with application groups to develop components that encapsulate high-level tasks.
  - Move a file between two GridFTP servers
  - Stage a data set, submit a computation, move the output data to a GridFTP server

Architecture



pyGlobus Components

- 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)
  - Remote file IO (gassFile)

pyGlobus Components

- Applications and Utilities:
  - All apps and utilities are instrumented using the NetLogger toolkit to support performance analysis (www-didc.lbl.gov/NetLogger/)
  - Full GridFTP Server
    - Used stand-alone or embedded in a Python application
    - Includes support for MD5 checksumming
      - First implementation to do so
  - GUI GridFTP client
  - pyglobusrun and pyglobus-url-copy
    - Same interface as the GT2 tools, but instrumented with NetLogger
  - GUI widgets for building domain specific applications

C Submission example

Job creation

```
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(“clipper.lbl.gov”,
      “&(executable=‘/bin/sleep’)(arguments=15)’,
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);
```

Callback for state changes

```
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_libc_printf(
         "GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING\n");
         break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
         globus_libc_printf(
         "GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED\n");
         monitor->done = GLOBUS_TRUE;
         break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
         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);
}
```

Python Job Submission Example

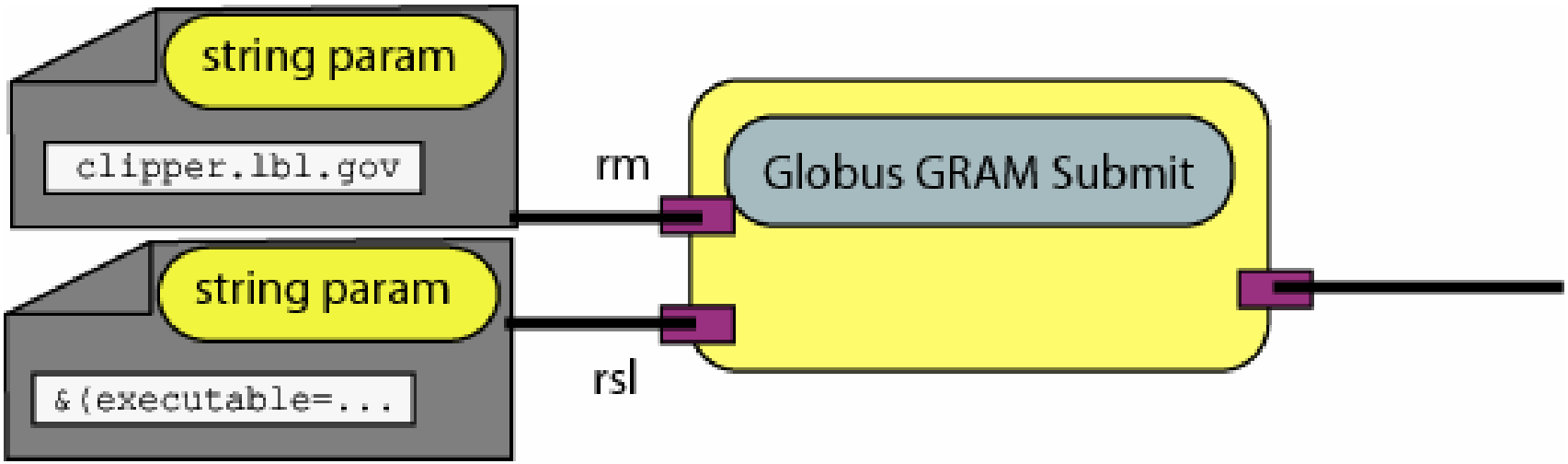
Creating a job

```
lock = Condition()
done = GLOBUS_FALSE
try:
    gramClient = GramClient.GramClient()
    callbackContact = gramClient.set_callback(func, (lock, done))
    rslSpec = RSL("/bin/sleep", **{'argument':15}
    jobContact = gramClient.submit_request("clipper.lbl.gov",rslSpec,
                            GramClient.JOB_STATE_ALL)
except GramClient.GramClientException, ex:
    print ex.msg
lock.acquire()
while done != GLOBUS_TRUE:
    lock.wait()
```

Callback for state changes

```
def func(arg, contact, state, error):
   lock, variable = arg
   lock.acquire()
   if state == GramClient.JOB_STATE_PENDING:
        print "Job is pending"
   elif state == GramClient.JOB_STATE_FAILED:
        print "Job is active"
         variable = GLOBUS_TRUE
   elif state == GramClient.JOB_STATE_DONE:
        print "Job is done"
        variable = GLOBUS_TRUE
   lock.notify()
   lock.release()
```

Visual Workflow Job Submission



pyGlobus Status

- Releases available in NMI, VDT, and GT3.x (standard BSD license)
- Current development code available from CVS
- Used by several Grid projects
  - LIGO moves production terabytes using GridFTP modules
  - Access Grid client software

Future Plans

- Continue to support GT2.x users
- Provide a stable migration path to Grid Services via SAGA
- Use SWIG more effectively to reduce hand-built wrapping code

Python Web and Grid Services Support

Grid Evolution

- GT2 implementations based on a heterogeneous protocol base
  - Used standard protocols where applicable
    - TLS, LDAP, HTTP, etc.
  - Invented new protocols
    - GRAM, GRIP, GSI, GridFTP
- Along the way problems were discovered:
  - Lack of compatibility between versions
    - Protocols kept changing
  - Protocols (and APIs) were
    - Poorly documents
    - Sometimes only defined in the code

Grid Evolution

- Ubiquitous adoption demands open, standard protocols
  - Internet and Web as guides
  - Enables innovation/competition on end points
- At the same time industry was beginning to standardize on Web Service protocols
  - Could provide a common protocol base for the Grid, but …
- Led to the adoption of Web Services as the underlying framework for Grid Services

pyGridWare

- Developing a full Open Grid Services Architecture implementation
- Specifically addressing interoperability with WS-RF implementations including Globus Toolkit, IBM, and Fujitsu
- ZSI library is used for
  - Client proxy and server stub generation
  - SOAP parsing
- Twisted application server for the hosting environment
  - Also support standalone Grid Services

pyGridWare Client Bindings

- Builds on the automatic wsdl2python generator we've built to use with ZSI
- Security support automatically added into generated code

pyGridWare Web and Grid Services Server-Side Support

- Provides a container to host Grid Services
  - Based on the Twisted project (http://twistedmatrix.com/products/twisted)
  - Provides a base-class to encapsulate all required Grid Service functionality
  - Will support exposing legacy Fortran/C/C++ codes as Python components
  - Will provide automatic server stub generation from WSDL

Current Status

- Code released under the standard BSD license
- Client bindings are included in the Globus 3.2 release
  - Includes interoperable message level security
  - Working to integrate the common Web Service code back into the ZSI project
  - Still working on the border cases with complex type encoding and interoperability with non-BP compliant services
- Server code exists in beta form but has not been officially released
- Refactoring the code base to reflect the recent change from OGSI to WS-RF
  - WS-RF is easier for us since it uses less “obscure” features of XML Schema

Future Plans

- Finish reworking all of our OGSI code to support WS-RF
  - Initial investment in basic Python WS tooling makes this easier
  - Uses fewer XMLSchema “features”
- Plan to have at least client code ready for the Globus Toolkit 4.0 release

SAGA (Simplified API For Grid Applications)

Making the grid more accessible to scientists

GGF SAGA Research Group

- Existing toolkits and services have a steep learning curve
- SAGA aims to develop an ultra-simplified API to Grid toolkits
- Keith Jackson originally explored this concept in pyGlobus and later recognized the API could be implemented in multiple languages (which helps when migrating from “test” codes in scripting languages to “production” codes in C and Fortran)
- Common operations with default settings will be convenient (single call) while more complex operations will be available to advanced programmers
- Natural mapping to specific language OO features

SAGA directions

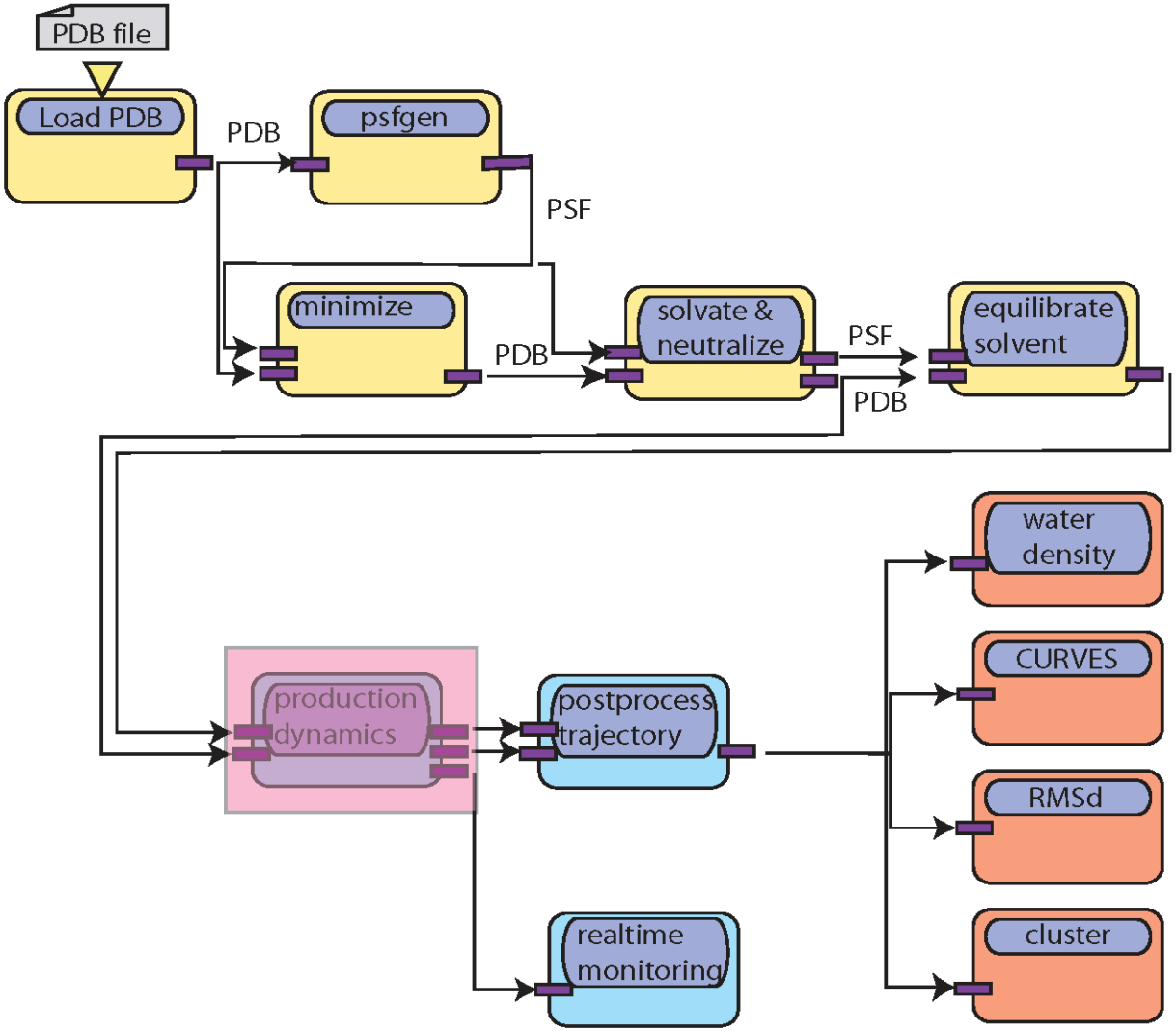
- Facilitate an easy transition from pre-Grid to Grid architecture
- Allow scientists to incorporate grid components when they want to, without having to fully buy in to the Grid architecture
- Most common operations are file transfer and remote job submission, but other more sophisticated operations, including replica directory management and service composition will be targeted
- Explicitly support the continuum of tooling (from GT2.x to Web Services to GT3.x/WS-RF) with a single API (future proof!)

Issues associated with WS to WS-RF transitions

- Many more domain scientists are using WS than WS-RF/Grid as the the interaction technology (typically with nonstandard methods of state management)
- To use Globus functionality, WS-RF conventions for managing and interacting with stateful resources must be used
- pyGridware (through ZSI) supports both plain WS and WS-RF
- Using SAGA and pyGridware (or other Grid implementations) may reduce the effort barrier associated with refactoring WS codes to support WS-RF features

Visual programming for workflow composition

Molecular simulation workflow

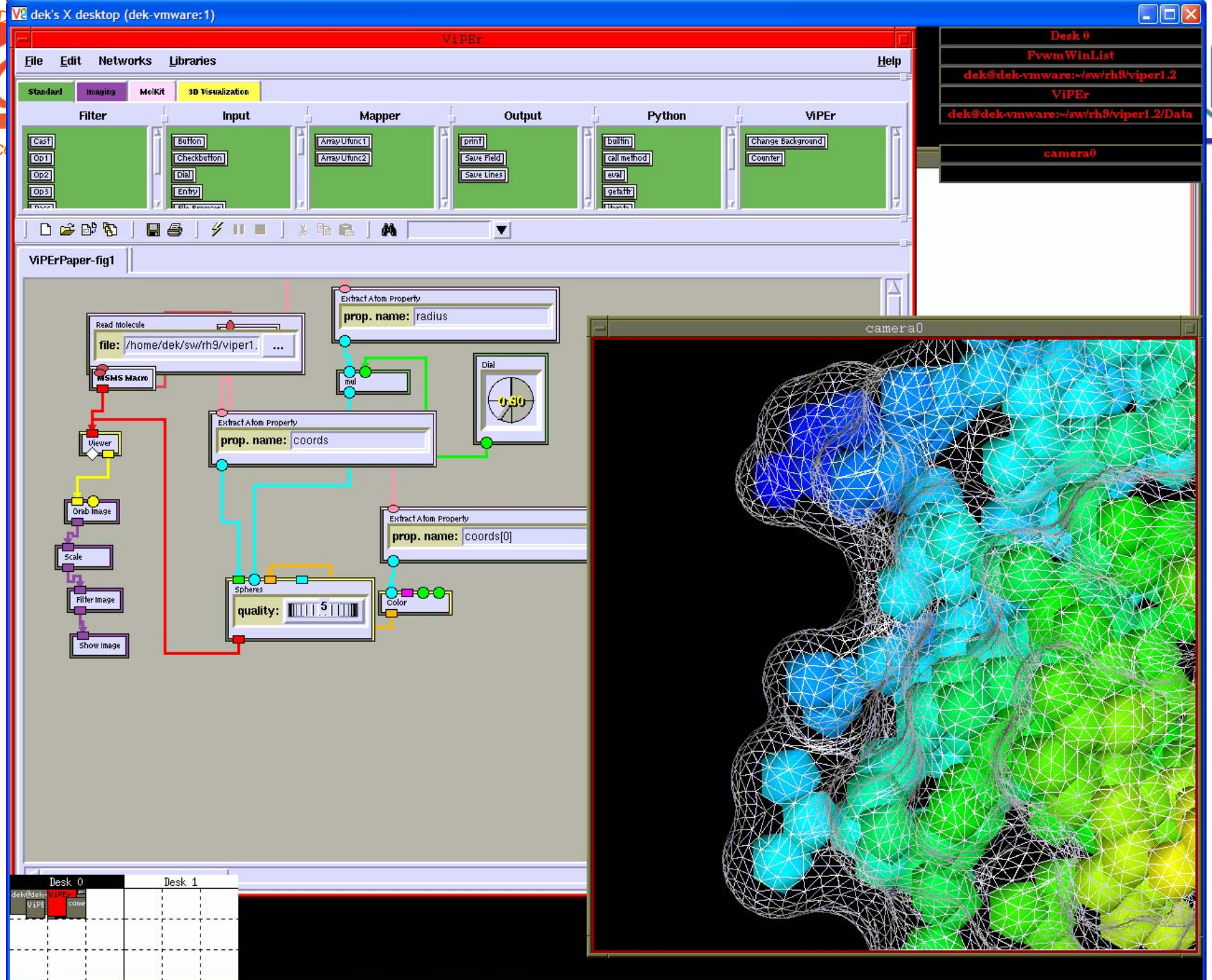


Visual Programming

- The previous example was implemented using a hand-built Python script which required significant operator interaction
- Instead, users will compose workflows visually; by dragging and dropping
  - nodes that refer to published web and grid services
  - nodes that perform minor business logic implemented on the local machine
  - form nodes that are used to populate instances of WSDL types
- A Consumer node's input ports can only be attached to a producer's output which emit the appropriate WSDL type (sufficiently complex nodes can use introspection to deal with wildcard types)
- We are specifically targeting this technology at two groups
  - Web/grid service software developers who want people to reuse their components
  - Application domain scientists who don't want to develop web or grid services but want to reuse components in their own workflows

VIPER- Visual Programming in Python

- We were inspired by VIPER, a visual programming environment written by Michel Sanner
- VIPER is used to compose networks of Python scripts for molecular data processing and visualization
- Data “flows” from parent nodes to children node as it becomes available; changing data at a particular location in the tree only causes children to be recomputed


dek's X desktop (dek-vmware:1)
ViPEr
File
Edit
Networks
Libraries
Help
Standard
Imaging
MolKit
3D Visualization
Filter
Input
Mapper
Output
Python
ViPEr
Cast
Op1
Op2
Op3
Button
Checkbutton
Dial
Entry
Array Ufunc1
Array Ufunc2
print
Save Field
Save Lines
builtin
call method
eval
getattr
Change Background
Counter
Desk 0
FvwmWinList
dek@dek-vmware:~/sw/rh9/viper1.2
ViPEr
dek@dek-vmware:~/sw/rh9/viper1.2/Data
camera0
ViPErPaper-fig1
Read Molecule
file: /home/dek/sw/rh9/viper1.
MSMS Macro
Extract Atom Property
prop. name: radius
mul
Dial
0.50
Extract Atom Property
prop. name: coords
Viewer
Grab Image
Scale
Filter Image
Show Image
Extract Atom Property
prop. name: coords[0]
Spheres
quality: 5
Color
camera0
Desk 0
Desk 1

Grid Visual Workflow Environment (VIPUS)

- We began by adding pyGlobus nodes (file transfer, remote job submission) to VIPER; for example, a file selection dialog can be replaced by a file transfer dialog so a network can access remote data
- Support for advanced grid services, complex workflows (control flow like loops and conditionals) and multiple language support required a new design oriented around web services
- The user interface allows users to compose new types and services visually (type editor), write code for local nodes (code editor), and generate proxy nodes from remote web services (web service browser).
- Condor's DAGMAN is used to execute the graphical workflow, but we plan to support multiple DAG execution mechanisms.

Acknowledgements

- The Python Grid Project is funded by the U.S. Department of Energy Office of Science
- Thanks to the Globus Project

- More information can be found at
  - http://dsd.lbl.gov/gtg/projects/pyGlobus/
  - http://dsd.lbl.gov/gtg/projects/pyGridWare/
  - https://forge.gridforum.org/projects/saga-rg/
- Bug submission
  - http://dsd.lbl.gov/bugzilla/
- Email:
  - krjackson@lbl.gov
  - python-discuss@globus.org