Clarens Web Services Framework

LISHEP 2004, Feb 2

Grid Architecture, Infrastructure & Middleware

Conrad Steenberg, Julian Bunn, Harvey Newman, Michael Thomas, Frank van Lingen

California Institute of Technology

Developed as part of the

Particle Physics DataGrid

Overview

- Background
  - Grids
  - Web services
- Standards
- Clarens architecture
- Implementations
- Service tutorial
- Security and Virtual Organizations

Background

- Grid Ideal

  A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to ~~high-end computational capabilities.~~ Kesselman & Foster, 1998

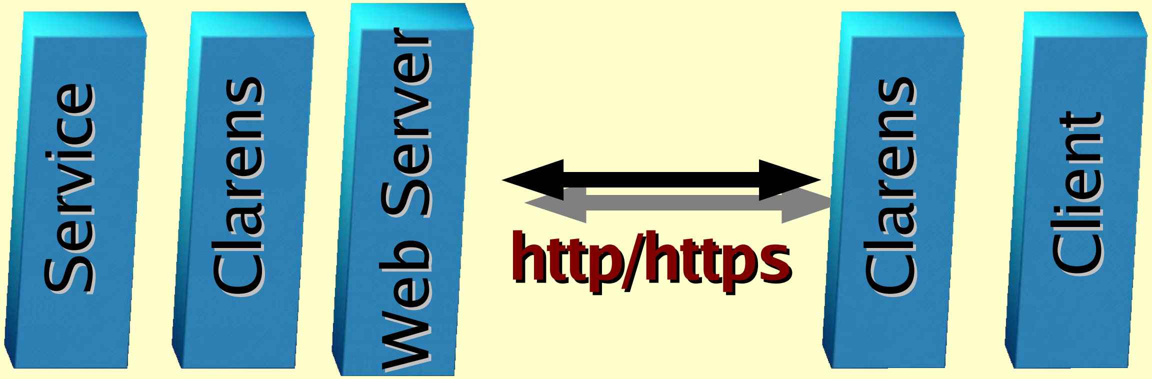
- Web Services
  - Document web vs. Programmatic web
  - Leverages weak coupling, simplicity, and standardized approach of the web
  - Mostly implies XML messaging over HTTP
  - NOT server-generated HTML pages
- Grid does not imply an implementation
  - Web services form an ideal vehicle for implementing Grids

Standards

- Commoditized standards hallmark of web services
  - SOAP for messaging
  - WSDL service descriptions
  - SSL encryption, key exchange (PKI/X509 certificates)
  - HTTP for transport mechanism
  - UDDI for service discovery (*)
- This is not enough for building real applications
  - Need a framework for providing services (library, conventions)
  - Distributed security, administration
  - Not only clients and servers, but truly distributed *system*

Clarens architecture

- Thin, high performance web services layer to allow programmatic access to computational resources
- Allows lightweight clients up to heavyweight servers to access services
- Use web service standards, allows commodity clients
- Strong focus on security
  - X509 certificates for authentication, optional SSL encryption
  - Authorization at resource level (method ACLs, VO ACLs)
  - Logging of requests and responses



Implementations

- Two implementations
  - Multi-process Apache server using embedded Python interpreter (mod_python) and C/C++ - used for tutorial
  - Multithreaded Tomcat servlet container with own or Apache web server using Java (unreleased)
- Additional standards supported
  - GSI authentication, HTTP Basic authentication (both also using X509 certificates)
  - XML-RPC for messaging
- Clients available:
  - C/C++
  - Python
  - Java (PDA, workstation)
  - Javascript (Browser)

Server Notes

- Each RPC is handled by own server process
  - Crashing module doesn't affect neighbours
  - Long-running requests does not block server
  - Leverages SMP when available
  - Server farm with load-balancing can appear as single virtual server
- Stateless protocol
  - Clients do not hold connection
- Session data stored in DB
  - Clients can survive server restarts, sees temporary server unavailability

Service tutorial

- Server installation as *root* or ordinary user
  - See http://clarens.sf.net
- Use Python OO “interpreted” language
  - http://www.python.org
- Use **mod_python** interface to Apache web server
  - http://www.modpython.org
- Powerful database access with e.g.
  - Berkeley DB - http://www.sleepycat.com (session management)
  - MySQL - http://www.mysql.com
  - SQLite - http://www/sqlite.org
- Use C/C++ extensions where speed is needed

Service Plug-Ins

- Services implemented via plug-ins
- Directory name determines root of method name
  - e.g. system.* methods reside in system directory
- Users can install modules under login directory
  - This can be disabled if needed!

{root}	/system/__init__.py	system.auth system.logout system.*
	/file/__init__.py	file.read file.md5 file.*
	/proxy/__init.py	proxy.store proxy.retrieve proxy.*

{home/user/clarens}	/analysis/__init__.py	~user.analysis.init ~user.analysis.chi2 ~user.analysis.*
	/transform/__init.py	~user.transform.init ~user.transform.fft ~user.transform.*

Basic Service

File echo/__init__.py:

Import support modules:

```
 from clarens_util import *
 from mod_python import apache
```

Define function:

```
 def echo(req,method_name,args):
```

Construct response:

```
   response = build_response(req,method_name,args)
```

Write response:

```
   write_response(req,response)
```

Return:

```
   return apache.OK
```

Service Registration

File echo/__init__.py:

- Let the world know about our new method:

```
methods_list={'echo':echo}
```

- Method name: 'echo'
- Method object: echo
- Method signature:

```
methods_sig= {'echo':['string,string']}
```

Complete example

```
from clarens_util import *
from mod_python import apache


def echo(req,method_name,args):
  response = build_response(req,method_name,args)
  write_response(req,response)
  return apache.OK


methods_list={'echo':echo}
methods_sig= {'echo':['string,string']}
```

Error handling

Use build_fault() to construct an exception:

```
def echo(req,method_name,args):
  try:
    response = build_response(req,method_name,args)
  except:
    response = build_fault(req,method_name,
                           apache.HTTP_BAD_REQUEST,
                           "Bad request echo %s"%(args))
  write_response(req,response)
  return apache.OK
methods_list={'echo':echo}
methods_sig= {'echo':['string,string']}
```

More useful method

Use build_fault() to construct an exception:

```
def get_dn(req,method_name,args):
  try:
    response = build_response(req,method_name,
                              req.clarens_dn)
  except:
    response = build_fault(req,method_name,
                           apache.HTTP_BAD_REQUEST,
                           "Bad request %s"%(method_name))
  write_response(req,response)
  return apache.OK

methods_list={'echo':echo,
              'DN'  :get_dn}
methods_sig= {'echo':['string,string'],
              'DN'  :['string,array']}
```

WSDL Service Description

Previously specified XML-RPC method signatures

WSDL much more complete format

Add to file __init__.py:

```
methods_wsdl= “””
  <?xml version="1.0" encoding="UTF-8"?>
  <wsdl:definitions targetNamespace="urn:echo" ...
  ...
  ...
“””
```

Debugging

- Print debugging output:

```
err_msg("Output message")
```

- Server error log:

```
[Tue Feb 10 03:21:25 2004] [notice] Output message
```

- Send HTML formatted tracebacks to client:

```
import cgitb; cgitb.enable()
```

- Use command-line Python debugger

Add line to mod_python configuration file:

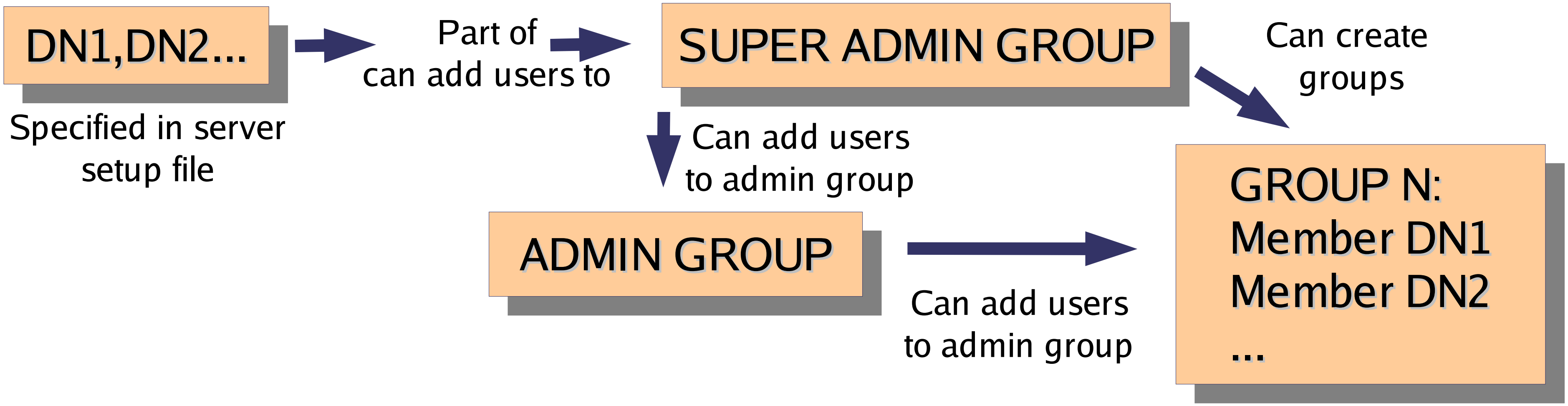
```
PythonEnablePdb ON
```

Start Apache server with only one process:

```
httpd -X -DONE_PROCESS
```

Security and Virtual Organization

- Authentication via X509 certificates
  - Verifies certificate chain up to a list of accepted Certificate Authority certificates
  - Client identified internally by the certificate distinguished name (DN) – uniqueness ensured by CA
- Authorization done using an internal VO
  - VO consists of a hierarchy of groups and users
  - Does not need to store client certificates, uses Dns
  - VO data stored in DB



DN1,DN2...

Specified in server setup file

Part of

can add users to

SUPER ADMIN GROUP

Can create groups

Can add users to admin group

ADMIN GROUP

Can add users to admin group

GROUP N:
Member DN1
Member DN2
...

Security and Virtual Organization II

- Authorization for methods based on ACLs
  - ACLs bootstrapped from .clarens_access files in module directories
  - Store in DB, can be administered remotely
  - Based on model of Apache .htaccess files
- E.g. for system.auth() method which is required for login:
  - Order allow, deny
  - Allow all in specified group(s) or list of DNs to access method
  - Unless member of group(s) in deny list, or DN in deny list
  - Similar for order deny, allow
- Authorization is hierarchical based on method name
  - E.g. the ACL for 'system' has precedence over 'system.listMethods', making it easy to specify ACLs with the minimum information
- System ACL is special
  - Can specify access to all methods
  - Normal module .clarens_access files cannot specify access controls for other modules

Security and Virtual Organization III

Example *.clarens_access* file for *system* module

```
access=[('system',
      [ORDER_DENY_ALLOW,                  # Order
      ['/O=doesciencegrid.org/OU=People'], # Allow DOE certificates
      ['CMS'],                           # Allow group CMS
      [],                                # Deny individuals
      ['revoked_certs'],                 # Deny group members
      [None, None, None]]),              # modtime, start_time, end_time
     ('system.updateMethods',
      [ORDER_ALLOW_DENY,                 # Order
      ['/O=doesciencegrid.org/OU=People/CN=Conrad Steenberg'], # Allow
      ['admin'],                         # Allow group admin
      [],                                # Deny individuals
      []  ,                              # Deny default
      [None, None, None]])]              # modtime, start_time, end_time
```

Security and Virtual Organization IV

Example *.clarens_access* file for *demo* module

```
access=[('',                             # module name is prepended
      [ORDER_DENY_ALLOW,                 # Order
      [''],                              # Allow
      ['Caltech', 'UFL'],                # Allow 2 groups
      [],                                # Deny individuals
      ['revoked_certs'],                 # Deny group members
      [None, None, None]]),              # modtime, start_time, end_time
     ('DN',                              # method name
      [ORDER_ALLOW_DENY,                 # Order
      ['/O=doesciencegrid.org/OU=People/CN=Conrad Steenberg'], # Allow
      ['admin'],                         # Allow group admin
      [],                                # Deny individuals
      []  ,                              # Deny default
      [None, None, None]])]              # modtime, start_time, end_time
```

Security and Virtual Organization V

- For normal modules, the module name is prepended to the method name
- Authorization does not require changes in the certificate structure
- ACLs and VOs can be remotely administered without system admin intervention
- VO administration allows for multiple group administrators
- Does not require certificate revocation lists – ACLs can be used to deny access to revoked certificates via the VO
- ACLs currently limited to method access, but can also be used for file access control
- More info at `http://clarens.sf.net`

Summary

- The Clarens architecture presents users and developers with a high performance, scalable and fault-tolerant way to implement web services in a Grid environment
- Benefits derived from the commodity Apache server platform
- VO and authorization (ACL) administration can be done remotely after bootstrapping essential information from text files once after installation
- Currently deployed in a variety of projects in the US, at CERN and Pakistan
- Used as a “portal” to classical Globus Toolkit Grids
- Used as basis for Grid-enabled Analysis Environment (GAE) in CMS experiment.
- More info at http://clarens.sf.net