



*Building the National Virtual Collaboratory  
for Earthquake Engineering Research*

**NEESgrid**

**Technical Report NEESgrid-2003-01**

[www.neesgrid.org](http://www.neesgrid.org)

(Draft Whitepaper Version: 1.0  
Last modified: January 8, 2003)

## **Integrating Grid Capabilities into the CHEF Collaborative Portal Framework**

**Charles Severance<sup>1</sup>**

<sup>1</sup>University of Michigan, Ann Arbor, MI 48109

Feedback on this document should be directed to [csev@umich.edu](mailto:csev@umich.edu)

---

**Acknowledgment:** This work was supported primarily by the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) Program of the National Science Foundation under Award Number CMS-0117853.

## Introduction

This document describes the design for integrating support for the grid into CHEF [[www.chefproject.org](http://www.chefproject.org)]. CHEF is a set of collaborative tools which are built on top of the Jakarta Jetspeed portal toolkit. The goal of this effort is to allow additional grid-oriented tools to be quickly built in CHEF which make use of grid services and operate within a grid security context. The initial purpose of this effort was to provide a framework to develop a collaborative portal [[www.neesgrid.org](http://www.neesgrid.org)] for structural engineers which uses the grid as its authentication.

The grid provides a distributed authentication and authorization mechanisms based on X.509 certificates. In addition to using X.509 certificates, the grid adds capabilities for certificate management, certificate proxy support, certificate storage and retrieval, and secures web services which require authorization identity. As such, the grid provides a strong set of tools to build distributed systems which require strong authorization and authentication.

There are a number of portal technologies which provide the ability for web sites to operate within a grid context. The Grid Portal Development Kit (GPDK) [[doesciencegrid.org/projects/GPDK/](http://doesciencegrid.org/projects/GPDK/)] provides grid functionality to web sites using JAVA Beans which encapsulate grid functionality. By using beans, the GPDK functionality is accessible using the JSP (Java Server Pages). This allows users to take a relatively straightforward static web site and quickly add grid functionality. The disadvantage to this approach is that beans must be developed to support each capability. A PERL-based portal toolkit called GridPort was developed at the University of Texas [<https://gridport.npaci.edu/>].

The next generation of Grid Portals will need to make use of the rapidly expanding capabilities of the Commodity on Grid Toolkit COG [[www.globus.org/cog/](http://www.globus.org/cog/)] and the emerging Open Grid Services Architecture (OGSA) [[www.globus.org/ogsa/](http://www.globus.org/ogsa/)]. These capabilities move the grid from a distributed job submission, monitoring, and resource discovery system to a toolkit capable of building distributed applications using grid-enabled web services and other capabilities.

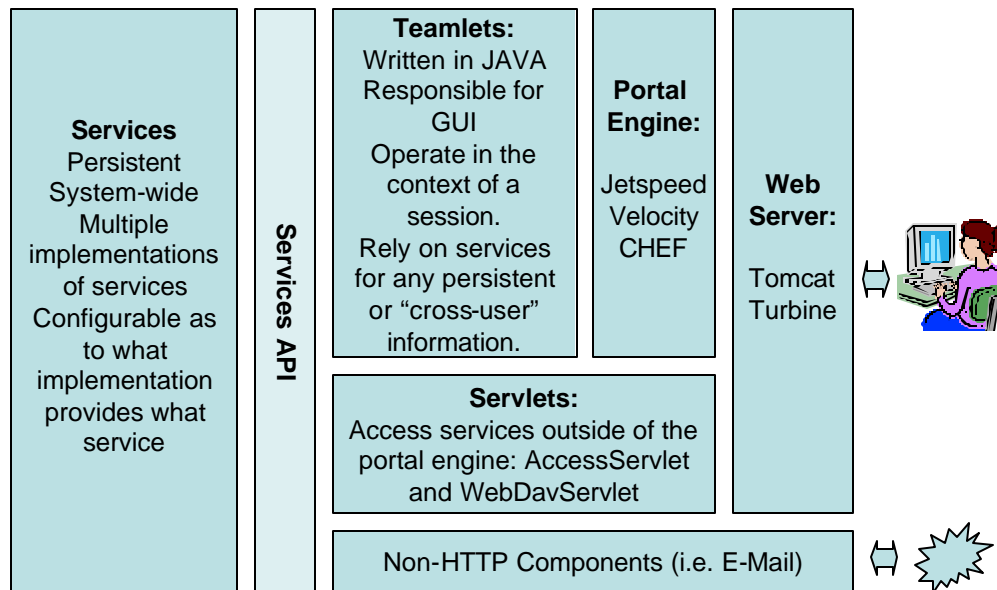
The COG and OGSA deploy their functionality primarily in the JAVA language. By writing applications in a JAVA environment, they can make full use of the complete range of capabilities of these emerging toolsets. This makes environments such as Jetspeed and CHEF an excellent environment for developing grid portal capabilities.

The Alliance Portal Project (NCSA, IU, ANL, Utah) [<http://www.ncsa.uiuc.edu/Expeditions/Portals/>] is developing a set of reusable Grid Components using Jetspeed as the base structure. The CHEF team is working cooperatively with the Alliance portal team to come up with a common standard for Grid-enabled Portlets operating in Jetspeed.

## Overview of CHEF

CHEF is an open source collaborative portal framework which is based on the Apache projects Tomcat [jakarta.apache.org/tomcat/] and Jetspeed [jakarta.apache.org/jetspeed/]. Tomcat provides the servlet container environment and Jetspeed provides the flexible and reconfigurable portal environment. By relying on existing open-source projects, CHEF has been able to building functionality at the tool and service level rather than building base functionality.

# CHEF Architecture



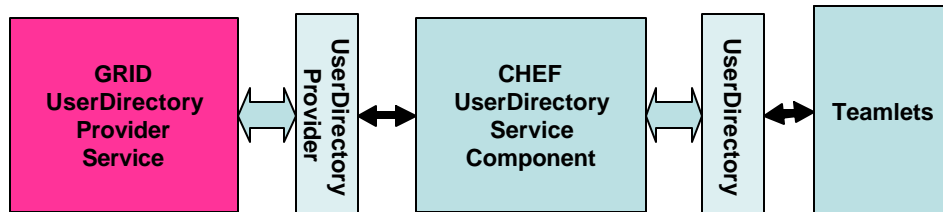
Functionality developed in CHEF is separated into two components: Teamlets and Services. Services provide long term persistence and are accessed using a standardized Services API. As part of each CHEF installation configuration the appropriate service implementations for each CHEF site are "plugged-in" to the generic services APIs.

Teamlets are the components which provide the functionality to the end users. A teamlet is developed for each tool (Chat, Schedule, Resources, etc...). Teamlets are stateless – they rely on services for long-term persistence and on Tomcat/Jetspeed for session persistence.

## Required Modifications to CHEF to Operate within the Grid

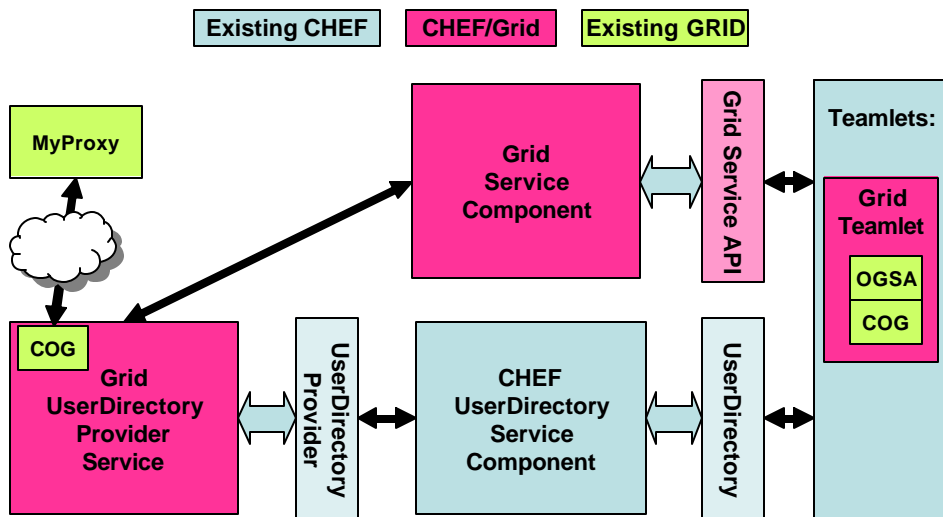
The primary modification to CHEF is to augment the built-in security model within CHEF to use the grid authentication as the authoritative source of authentication and authorization information.

CHEF Teamlets access information about a user through the service API to the CHEF UserDirectoryService. The UserDirectoryService is also consulted by the CHEF framework when a user is initially logged into the system. The CHEF UserDirectoryService component can be configured to delegate basic user authentication operations such as verifying an id/password combination to a UserDirectoryProvider. By writing and configuring a Grid UserDirectoryProvider plug-in, CHEF treats the Grid as the authoritative source of user authorization.



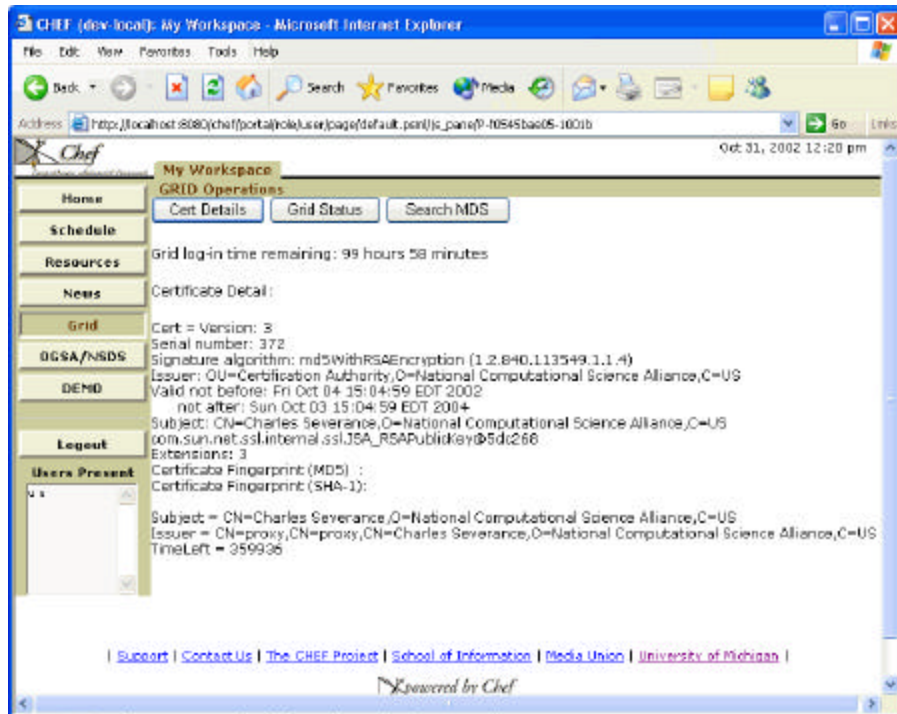
The grid UserDirectoryProvider will consult a MyProxy [www.ncsa.uiuc.edu/Divisions/ACES/MyProxy/] server which has been configured by the CHEF administrator when it is presented with an id/password pair for log-in to CHEF. If MyProxy returns a valid credential, the user is logged into CHEF.

A new service has also been added to CHEF specifically to support Teamlets which make use of Grid Services via the COG and/or OGSA. As the user is logged in, their certificate is passed from the UserDirectoryProvider to the Grid Service Component for storage during the user's session.



Using the Grid Service API, a Teamlet can retrieve the certificate associated with a particular user so that the Teamlet may perform operations using the OGSA or COG libraries on behalf of the user.

In addition, several sample Teamlets have been written to show how to use the Grid Service and make GOG or OGSA calls.



The above image shows the output of one of the sample Grid Teamlets to display the contents of the current certificate for the current logged in user.

### Writing Grid Enabled Teamlets in CHEF

The following is a simplified version of the code elements needed in the above Teamlet. The Teamlet uses Turbine to find the system-configured Grid Service and then uses the Grid Service API to retrieve and print the proxy information for the user who is logged into this session.

```
import org.chefproject.service.GridService;

import org.globus.security.*;
import org.globus.myproxy.*;

// Use turbine to find the CHEF Grid Service
GridService m_Grid = (GridService)TurbineServices
    .getInstance().getService(GridService.SERVICE_NAME);

// Retrieve the proxy for this user from the service
proxy = m_Grid.getCurrentUserGlobusProxy();

System.out.println("Cert:" + m_Grid.toString(proxy));
```

For a Teamlet to make use of OGSA services, it must import the additional classes from OGSA and provide the proxy to the OGSA tool using the appropriate interface:

```
import org.globus.ogsa.samples.NsdsServiceLocator;
import org.globus.ogsa.samples.NsdsPortType;
import java.net.URL

NsdsServiceLocator locator = new NsdsServiceLocator();
NsdsPortType nsds = locator.getNsdsPort(new URL(uri));

// Provide the credentials
((Stub)nsds)._setProperty(GSIConstants.GSI_CREDENTIALS, proxy);
String channel = nsds.createChannel();
```

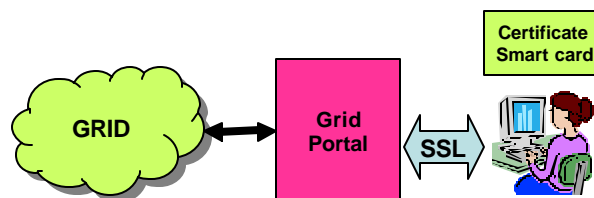
In general, most of the source code from the OGSA and COG sample applications can be dropped into a Teamlet and operate without change beyond the specification of the appropriate proxy to use when performing authenticated services.

## Grid and MyProxy Security Model

A user's identity within the grid consists of a signed X.509 certificate. Within a certificate the owner's identity is called the subject:

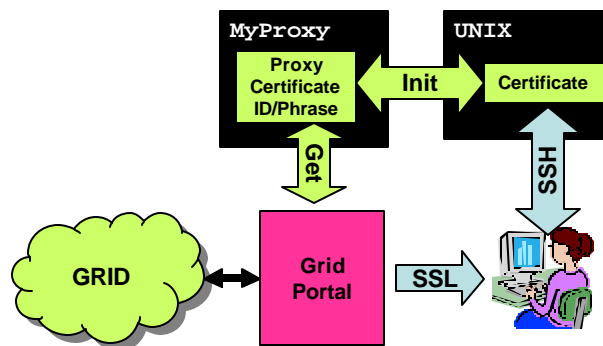
```
/C=US/O=National Supercomputing Alliance/CN=Charles Severance
```

This string is the unique Grid-wide identity. In an ideal world using certificate based authentication, the user's certificate would be stored in a smart card which would be read as part of a bi-directional SSL handshake between the user's browser and the Grid Portal.



Unfortunately, there is little support for smart cards in browsers, and in general, browser support for user certificates in the SSL handshake is problematic.

Because of these limitations in browser support, the Grid has created a way to manage user certificates outside of the browser and the user's desktop. This capability is called MyProxy. There are two typical scenarios in a MyProxy environment. In the first the certificate is stored on a UNIX system and in the second the certificate is stored on the user's personal workstation.



In the first scenario, before the user can use the grid, and the Grid Portal, they log into a UNIX server, and “check in” a limited duration derived certificate to the MyProxy server. This limited duration proxy certificate is protected by an ID/phrase combination. Each time the user checks in their certificate, they can use a different ID/phrase combination. The default for the ID is the user’s UNIX account on the UNIX system.

An example of how the user checks in their certificate is as follows:

```

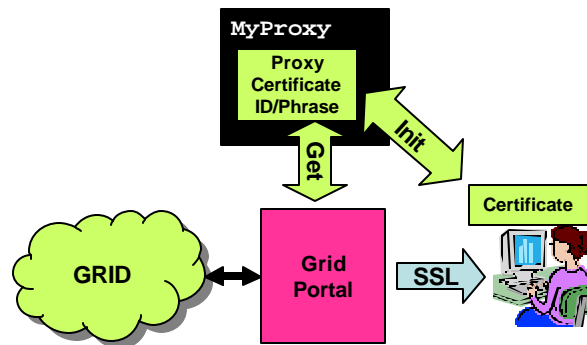
login as: crs
password: secretunixpassword
Last login: January 14, 2003
$ myproxy-init -s myproxy.ncsa.uiuc.edu
Your Identity: /C=US/O=National Supercomputing Alliance
               /CN=Charles Severance
Enter passphrase for this certificate: secretcertificatepassword
Creating proxy .....Done.
Enter myproxy pass phrase: secretmyproxypassword
A proxy has been created for 168 hours.
$
  
```

There are three passwords which are used in the above interaction:

- The password for the user’s UNIX account
- The password which unlocks the user’s certificate. This password is specified when the user is requesting the certificate.
- The password for the temporary proxy certificate store. This is the MyProxy ID/password which the user presents to the grid Portal

Once the user has checked in their certificate, they connect to the grid portal. This interaction is a unidirectional SSL connection where the host has a certificate, but the user’s browser is not required to provide a certificate. A unidirectional SSL connection provides a secure channel for the data and establishes the host’s identity, but does not establish the user’s identity. The user logs into the portal using basic authentication or form-based authentication with their MyProxy ID and phrase through the SSL connection. The portal uses this information to check out the temporary certificate from MyProxy for the user so the portal can perform grid functions on behalf of the user.

This same scenario can be done with the user storing their certificate on their personal workstation which is capable of performing the myproxy-init operation. This software is available for Macintosh, Windows, or UNIX operating systems. This scenario has the advantage that it is more convenient, but it requires that the personal system be relatively well-secured, especially when a system is shared among users.



Once the proxy has been checked in from the workstation using **myproxy-init**, the user's interaction with the portal is the same as in the previous scenario.

### Blending the Grid/MyProxy and CHEF Security Models

While the grid maintains the authoritative source of each user's account, password, and unique identity, CHEF needs to maintain local profile information for each user so they can operate with CHEF. This profile contains the following information:

- User ID (typically a short identifier)
- Password
- First Name
- Last Name
- E-Mail Account

The user ID and password in CHEF align with the MyProxy ID and phrase directly. CHEF ignores the local password for grid users. While it may seem logical to pull the first and last name from the common name (CN) stored in the certificate subject, the semantics of the CN field is up to the certificate authority (CA) so CHEF simply treats the entire subject string as a single unique identifier.

This would be rather simple except for the fact that the user is able to select their MyProxy user ID each time they check-in their proxy. As such CHEF must store the subject information associated with each account as the account is first created. When a user is authenticating, CHEF must use the ID/password to get the user's credential. Upon successful retrieval of a proxy, the subject of the proxy must match the subject stored in the user's CHEF account. Authentication is denied if (a) a proxy cannot be retrieved, or

(b) the subject field of the retrieved proxy does not match the subject field stored in CHEF.

Because CHEF supports arbitrary named attributes for users, there are no required modifications to the base CHEF code. The only modifications are contained in the grid service components which have been added to CHEF.

In addition to the CHEF accounts which are authenticated using the MyProxy, CHEF will also provide support for accounts which do not have corresponding Grid accounts. The typical use for these accounts is for CHEF administration activities. In a demonstration site, there may also be a need for local-only CHEF accounts.

## **Conclusion**

Adding grid capabilities to CHEF is remarkably straightforward. A number of extensions were developed but there were very little changes required to the standard CHEF code. Deploying a CHEF/Grid service is as simple as setting some configuration options in the standard CHEF release. CHEF supports both local CHEF accounts and CHEF accounts which are backed by grid credentials.

Once CHEF has been configured to use MyProxy for its authentication, it becomes relatively straightforward to develop tools which use the COG or OGSA capabilities. Much of the sample source code provided with these products works without any modifications within the context of a CHEF Teamlet.

## **References**

NEESGrid [[www.neesgrid.org](http://www.neesgrid.org)]

CHEF [[www.chefproject.org](http://www.chefproject.org)].

GridForum [[www.gridforum.org](http://www.gridforum.org)]

Grid Portal Development Kit (GPDK) [[doesciencegrid.org/projects/GPDK/](http://doesciencegrid.org/projects/GPDK/)]

Commodity on Grid Toolkit COG [[www.globus.org/cog/](http://www.globus.org/cog/)]

Open Grid Services Architecture (OGSA)[[www.globus.org/ogsa/](http://www.globus.org/ogsa/)].

MyProxy [[www.ncsa.uiuc.edu/Divisions/ACES/MyProxy/](http://www.ncsa.uiuc.edu/Divisions/ACES/MyProxy/)]

Foster I, Kesselman K, (editors) *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.

Laszewski G, Foster I, Gawor J, GoG Kits: A Bridge Between Commodity Distributed Computing and High-Performance Grids, *Proceedings of the ACM JAVA Grande Conference 2000*.

GridPort UTexas.