Editor: Geoffrey Fox, gcf@indiana.edu

GRID COMPUTING ENVIRONMENTS

By Geoffrey Fox

THE GRID IS RAPIDLY EVOLVING IN CONCEPT AND IMPLEMENTATION, BUT WITH THIS EVOLUTION COMES CORRESPONDING EXCITEMENT AND CONFUSION AS TO THE "RIGHT" WAY TO THINK ABOUT GRID SYSTEMS. ONE AREA OF INTEREST IS Grid computing environments (GCEs), which essentially describe the user side of a computing system—how users interact via a set of distributed back-end resources.

Figure 1 illustrates a typical Grid architecture, in which there is a fuzzy division between GCEs and what is called the "Core" Grid in the figure. The latter includes access to resources, management of and interaction between them, security, and other such capabilities.

The new Open Grid Services Architecture (OGSA; www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-6-22.pdf), which is itself evolving, attempts to describe these core capabilities. The Globus project (www.globus.org) is perhaps the best-known example of core software implementation. In this article, we will elaborate on GCEs and discuss their relationship to portals and Grid programming environments.

We base our analysis on a recent collection of 28 articles on various approaches to GCEs (www3.interscience.wiley.com/cgi-bin/issuetoc?ID=102522447). This collection stemmed from work of the GCE research group of the Global Grid Forum (www.gridforum.org/7_APM/GCE.htm).

Grid aspects that form GCEs are not cleanly defined, although there is an operational definition stemming from the work areas of the GCE research group of the Global Grid Forum. In this article we describe the two major areas covered by this research group today. First we discuss "programming the Grid" and then we follow with a discussion of portals that control the user view of the Grid. Together they provide a linked component model for the middleware and user interface to the Grid.

Programming the Grid

If we analyze the just-mentioned collection and similar papers, we find common Grid features. The papers suggest a diagram similar to that of Figure 1, but they differ in the technology used (Perl versus Python, for example), capability discussed, and the emphasis on the user versus program (back-end resource) view.

GCEs fulfill at least two functions:

- Programming the user side of the Grid
- Controlling user interaction—such as rendering output and allowing user input to certain Web pages

We've discussed the role of Web services and the Grid in previous installments of this column, so we can assume that our Grid is implemented in terms of XML-specified Web services. This assumption might seem unreasonable, but the Web services model really is a distributed object model, and it has proven to be straightforward in converting other object models to this approach. Thus, we can think of the general approach of most modern GCE work in terms of Web services.

A Programming Model

For this discussion, let's think of application software in a simple, two-level hierarchy. Microscopic software written in Fortran, C++, and Python controls the individual CPUs. We assume that these languages generate nuggets or code modules. Traditional programming then attempts to associate these nuggets with a single resource.

Let's look at some examples. The nugget could be the SQL interface to a database, a parallel image-processing algorithm, or a finite-element solver. This nugget programming must be augmented for the Grid by integrating the distributed nuggets into a complete executable.

Feedback?

Geoffrey Fox wants to hear from you. Comments? Applause? Criticism? Suggestions? What topic shall we tackle next? We want to hear about interesting topics and potential authors that could benefit Web Computing readers.

Programming the nugget internals is currently viewed as being outside the Grid, although projects such as Grid Application Development Software (GrADS; www.hipersoft.rice.edu/grads) are looking to integrate individual resource (nugget) and Grid programming.

For this discussion, let's assume that each nugget is programmed and that we just need to look at the nuggets' overall integration. This integration is actually quite familiar, because it generalizes the Shell and Perl scripts used in single resources for Unix operating systems and the Microsoft COM and ActiveX interfaces in the PC environment.

Several other examples of this style of Grid programming exist. One broad class is called the problem-solving environment (PSE), which includes a portal interface to a set of carefully chosen tool and application services usually customized to a particular problem domain. This portal interface has a graphical user interface described later and some sort of "software bus" to link the PSE's different parts.

The integration of application nuggets is often called *workflow*; it offers users many different paradigms. One common model is a graphical interface in which users can choose nuggets from a palette and link nugget ports (or channels). This approach is familiar from visualization and image processing, in which systems such as AVS (www.avs.com) and Khoros (www.khoral.com) are well established.

Industry has also developed XML specifications for nugget linkage. Examples include the BPEL4WS Business Process Execution Language for Web Services (www-106.ibm.com/developerworks/webservices/library/ws-bpel) and the WSCL Web Services Conversation Language (www.w3.org/TR/wscl10).

Simpler and perhaps more powerful is "just" programming the linkage with scripting (such as Python) or compiled (like Java) languages. We expect multiple paradigms and multiple languages to be useful, but it is unlikely that any one of these is "best." Important Grid approaches for describing the programming of nuggets include the US Department of Energy's Common Component Architecture (www.cca-forum.org) and the Imperial College e-Science Networked Infrastructure project (www.lesc.ic.ac.uk/iceni) of the UK e-Science Program.

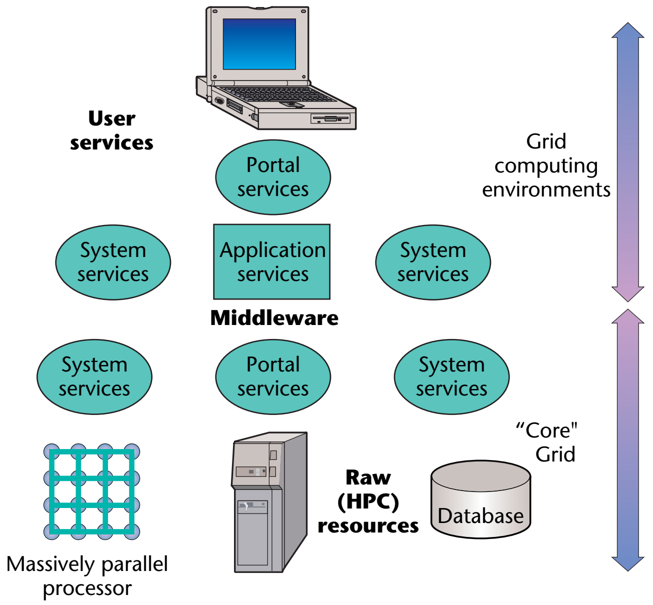


Figure 1. A Grid architecture showing portal services and Grid computing environments. We show typical resources at the bottom (a parallel computer, server, and database from left to right); the terminal at the top represents the user. The user and resources are linked by middleware built as (Web) services. At the bottom, we have "Core" services such as registration and look-up, security, and basic information. This is followed by less generic services that include applications built as Web services. These are interfaced to the user via the portal services described in the article. Grid computing environments support the portal services and the interface of application services between themselves and to system services.

Programming Services. Although related to tasks familiar in programming PCs or workstations, the "programming the user view of the Grid" function in GCEs is significantly more complicated. As Figure 1 shows, the executable (integrated nuggets) is a mixture of both system and application services. We use system services on a single workstation, but the Grid's meta-operating system services have programmable interfaces; many of the corresponding workstation (Windows, Unix) services are more opaque.

Part of the OGSA initiative plans to define and implement many of the system services shown in Figure 1. In fact, all services could be OGSA services when the dust clears—certainly all will be Web services. Alternatively, the OGSA and Web service specifications might just merge.

Many Grid systems separately maintain both "real" entities (such as a software nugget) and separate entities representing the metadata describing the "real" entity. We expect this separation to continue and even expand because there is a clear need to define more metadata. This metadata will most likely be stored separately from the resource it describes. There is growing interest in using ontologies (rich application-specific information) to describe the "semantics" (true inner meaning versus "just" the program

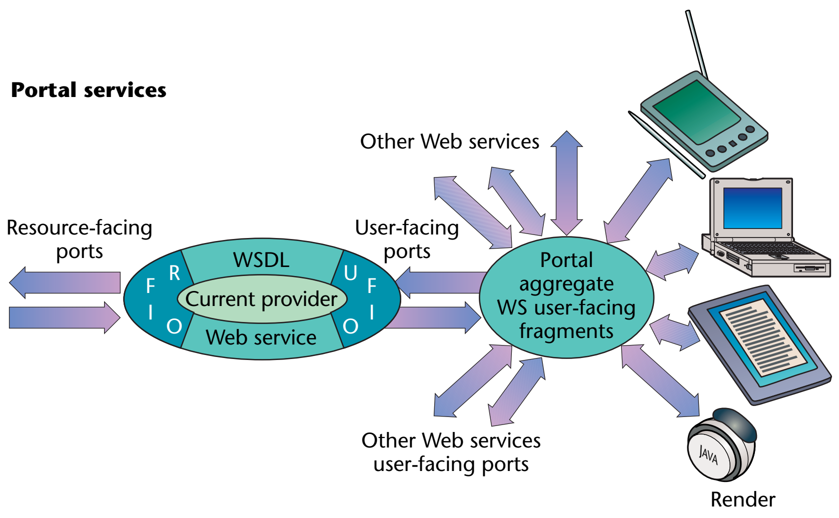


Figure 2. This illustration shows a portal providing an aggregation service for document fragments produced by user-facing ports (on the right) of a content-providing Web service. Generally, portals let you take multiple Web pages, automatically produce controls to link between them, and let subsets of them be displayed on a single Web page. Note that the Web service on the left consists of content or some way of producing content (such the output of a simulation) surrounded by input and output channels (or ports). Some ports (shown on the left) communicate with other services or resources; another set (on the right) communicate with the user. Several such Web services (each with user-facing ports) are aggregated for the user into a single client environment.

interfaces) of services. This topic, which is related to the W3C's Semantic Web initiative, deserves an article itself.

As a typical nugget programming challenge, we must take into account the needed latency and bandwidth of application and network constraints (firewalls) to decide the most appropriate communication mechanism between nuggets. This runtime specification of a service–service interaction's implementation has no agreed-upon approach. There are, of course, many examples; agents, brokers, and profiles are typical of the language we often use to describe this adaptive mechanism.

All the articles and papers we reviewed are partly differentiated by their emphasis on two different aspects:

- The programming paradigm and within a paradigm, particular languages (scripted, visual, or compiled)
- The runtime library (which could be shared among different paradigms in functionality but might be expressed rather differently in each separate approach)

The GCE Shell. We can borrow familiar ideas from Unix with the basic Grid programming primitives, usefully expressed as a *GCE shell*, which is a catalog of the primitive functions needed to program the Grid. Shell primitives are exposed to users in different ways using different paradigms and expressions of those paradigms. One way of exposing the shell primitives is as a command-line interface, but in many cases, we would present a higher-level view. Complete domain-specific, high-level systems are "just" PSEs.

The Legion Grid system (http://legion.virginia.edu) illustrates the GCE shell clearly, with the Legion shell being a natural extension of a familiar one from Unix. The GCE shell has some features in common with the Unix shell—for example, file manipulation is critical in both Unix and the Grid. However, there are some interesting differences. For instance, the Grid (and hence the GCE shell) must express

- The negotiated interaction between nuggets and users
- Files and services hierarchies at all levels of system, including local client, middle-tier, and back-end resource
- The distinction between an object and its metadata (copying an object might be a major high-performance task, but copying the metadata is typically a modest effort)

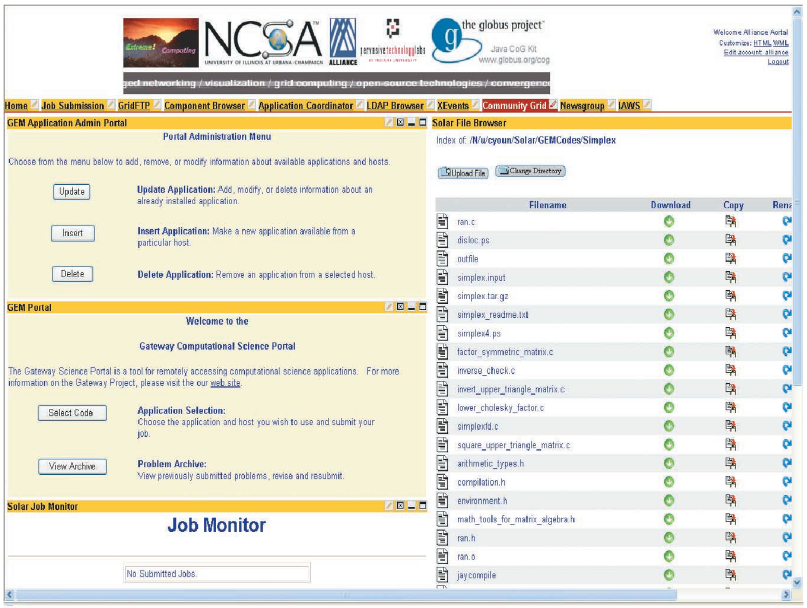
Looking at the primitives needed, the GCE shell must add several features (compared to the Unix shell) such as

- Search
- Discovery
- Registration
- Security
- Better workflow than pipe or tee in Unix shell
- Groups and other collaboration features as in JXTA (www.jxta.org)
- Metadata handling
- Management and Scheduling
- Networks
- Negotiation primitives for service interaction

We can simplify the discussion by using a uniform service model so that files and executables are services and not distinct (as in Unix). To do this, we need a "virtual service" concept so that an individual file access is a service in the shell, even though it could be implemented differently. This is an example of possible areas for new compiler research.

Portal Services

Portal services control and render the user interface–interaction; Figure 2 shows a key architectural idea emerging in this area. We assume that all material presented to users originates from a



Extreme! Computing
NCSA
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
ALLIANCE
pervasive technology labs
the globus project
Java CoG Kit
www.globus.org/cog
ged networking / visualization / grid computing / open-source technologies / convergenc

Welcome Alliance Aortal
Customize: HTML WML
Edit account: alliance
Logout

Home | Job Submission | GridFTP | Component Browser | Application Coordinator | LDAP Browser | XEvents | Community Grid | Newsgroup | IAWS

GEM Application Admin Portal

Portal Administration Menu

Choose from the menu below to add, remove, or modify information about available applications and hosts.

Update — **Update Application:** Add, modify, or delete information about an already installed application.

Insert — **Insert Application:** Make a new application available from a particular host.

Delete — **Delete Application:** Remove an application from a selected host.

GEM Portal

Welcome to the

Gateway Computational Science Portal

The Gateway Science Portal is a tool for remotely accessing computational science applications. For more information on the Gateway Project, please visit the our web site.

Select Code — **Application Selection:** Choose the application and host you wish to use and submit your job.

View Archive — **Problem Archive:** View previously submitted problems, revise and resubmit.

Solar Job Monitor

Job Monitor

No Submitted Jobs.

Solar File Browser

Index of: **/N/u/cyoun/Solar/GEMCodes/Simplex**

Upload File | Change Directory

Filename	Download	Copy	Rena
ran.c	●	●	●
disloc.ps	●	●	●
outfile	●	●	●
simplex.input	●	●	●
simplex.tar.gz	●	●	●
simplex_readme.txt	●	●	●
simplex4.ps	●	●	●
factor_symmetric_matrix.c	●	●	●
inverse_check.c	●	●	●
invert_upper_triangle_matrix.c	●	●	●
lower_cholesky_factor.c	●	●	●
simplexfd.c	●	●	●
square_upper_triangle_matrix.c	●	●	●
arithmetic_types.h	●	●	●
compilation.h	●	●	●
environment.h	●	●	●
math_tools_for_matrix_algebra.h	●	●	●
ran.h	●	●	●
ran.o	●	●	●
jaycompile	●	●	●

Figure 3. This screen shows an example of the Jetspeed-based portal with an aggregation of interfaces to several computing services. This NCSA computing portal shows a set of "tabs" below the banner at the top. The tab shown has four separate portlets (produced by four separate Web services); for example, there is a file browser on the right and a Job Monitor interface at the bottom left.

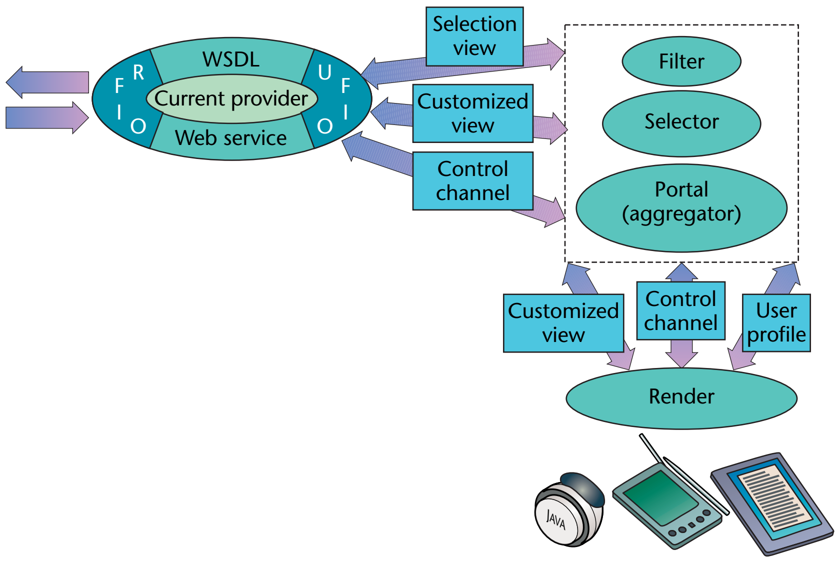


Figure 4. Portal services showing user-facing ports and negotiated interaction between the user and content-providing Web services. This interaction can provide universal access. It also expands the capabilities shown in Figure 2. The Web service interacts with the aggregation portal through a control channel that lets the user-facing ports declare support of multiple client renderings. The client user profile is combined with this declaration to select a customized view displayed as a portlet.

Web service, called a content provider in this case. This content could come from a simulation, data repository, or stream from an instrument.

Each Web service has resource- or service-facing ports that communicate with other services. However, we are more concerned with the user-facing ports, which produce content for users and accept input from client devices. These ports use an extension of the Web Services Definition Language (WSDL), which is being standardized by the Organization for the Advancement of Structured Information Standards (OASIS). This extension of the WSDL is called Web Services for Remote Portals (WSRP; www.oasis-open.org/committees/wsrp). Web Services compatible with WSRP implement the so-called *portlet interface*, which is being standardized in Java as part of a Java Community Process project.

Most user interfaces need information from more than one content provider. For example, a computing portal could feature separate panels for job submittal, job status, visualization, and other services. We could integrate this in a custom, application-specific Web service, but providing a generic aggregation service is more attractive. This lets users and administrators choose which content providers to display and what portion of the display real estate the content will occupy.

In this model, each content provider defines its own *user-facing document fragment*, which is integrated by a portal. Major computer vendors, such as Apache in its Jetspeed project (http://jakarta.apache.org/jetspeed) provide such aggregating portals. Portlets represent a component model for user interfaces in the same way that Web services represent a middleware component model. Using this approach has obvious advantages of reusability and modularity. It gives us an elegant view of the system, with workflow-integrating components (Web-services-representing nuggets) in the middle tier and aggregating portals integrating them for the user interface.

A portal being developed for the NCSA Alliance illustrates these ideas.

Figure 3 shows four separate interfaces (three left and one right) to different GCE shell commands implemented as Web services. Further capabilities are aggregated using tabs at the top.

This project involves many different institutions developing particular user interface fragments, with the component interface architecture providing convenient integration. The aggregation of the work of the different groups is provided by Web services in the middle tier and by systematic use of portlets at the user interface.

Figure 4 illustrates some other portal services that correspond to the ability of adapting rendered content to accommodate particular clients. This approach addresses both differences between devices (for example, immersive versus desktop versus handheld) and issues of universal access (to accommodate possible user physical limitations).

The architecture in Figure 2 becomes more complex because now we need negotiation between the client and the content provider to define the rendered view. This requires a portal selection service to process user profiles and choose appropriate content. We can package common filters, for example, to reduce resolution for multimedia content. This work on universal access is familiar in audio–video conferencing and is being pursued by W3C as part of its accessibility initiative.

The collection of aggregator, selector, and filtering capabilities illustrates common portal services multiple Grid applications can share.

We have described two very different aspects of Grid computing environments. The Grid consists of a set of distributed services covering application and system functionalities. GCEs support the integration or orchestration of these services, which you can think of as a set of middleware components that form a GCE shell. This middleware component model induces a corresponding component model for the user interface supported by aggregation portals.

The component view should lead to more modular reusable and productive environments with lower life-cycle costs. The Grid programming ideas supported by GCEs are active and exciting research topics. GCEs spanning both software engineering and research are one of the most active areas of progress for the Grid community.

Geoffrey Fox is director of the Community Grids Lab at Indiana University. He has a PhD in theoretical physics from Cambridge University. Contact him at gcf@indiana.edu; www.communitygrids.iu.edu/IC2.html.