Grid Computing Environments

Geoffrey Fox
Computer Science, Informatics and Physics
Indiana University
Community Grid Computing Laboratory,
501 N Morton Suite 224, Bloomington IN 47404
gcf@indiana.edu

Introduction

The Grid is rapidly evolving in both concept and implementation and there is a corresponding excitement and confusion as to the "right" way to think about Grid systems. One area of interest is called Grid Computing Environments (GCE) and roughly this describes the "user side" of a computing system where users interact via which controls a set of distributed back end resources. This is illustrated in figure 1 where there is a fuzzy division between GCE's and what is called "Core" Grid in the figure. The latter would include access to the resources, management of and interaction between them, security and other such capabilities. The new Open Grid Services Architecture (OGSA) http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf (which is itself evolving) describes or perhaps will describe these "Core" capabilities and the Globus project http://www.globus.org/ is the best known "Core" software. In this article, we will elaborate GCE's and discuss their relationship to portals and Grid programming

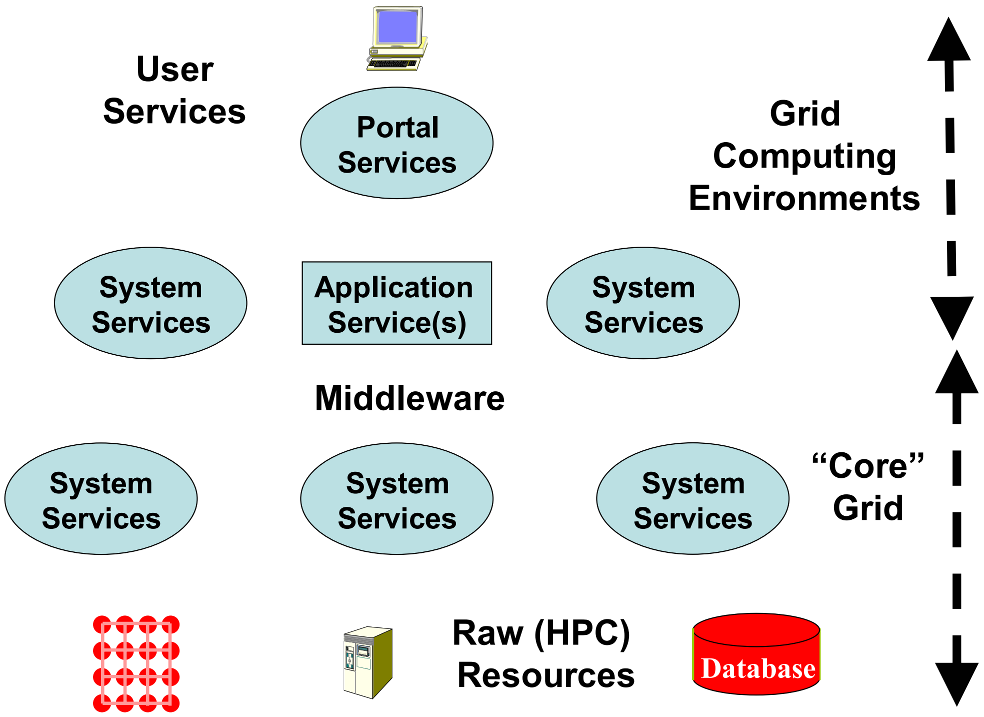


Fig 1: A Grid Architecture showing Portal Services and Grid Computing Environments

environments.

We will base our analysis on a recent collection by Fox, Gannon and Thomas of 28 articles on various approaches to GCE's – these are published in http://www3.interscience.wiley.com/cgi-bin/issuetoc?ID=102522447. This collection stemmed from work of the GCE research group of the Global Grid Forum, which can be found at http://www.gridforum.org/7_APM/GCE.htm. Although there appears to be quite a bit of confusion in the field, analysis of these (and other) papers shows some common general features which we will discuss here. Further details can be found in chapters 20-34 of *Grid Computing: Making the Global Infrastructure a Reality* (Berman, Fox and Hey editors, to be published by Wiley: Chichester, 2003) with Chapter 21 by Craig and Talia having a broad discussion of programming the Grid. The classification of GCE approaches is discussed by Fox, Gannon and Thomas in Chapter 20 of this book. The different papers all imply a diagram similar to figure 1 and differ in technology used (Perl versus Python for example), capability discussed and the emphasis on user versus program (back end resource) view.

GCE's fulfill (at least) two functions –

- "Programming the User Side of the Grid"
- Controlling user interaction – rendering any output and allowing user input in some (web) page. This includes aggregation of multiple data sources in a single portal page.

We have already discussed Web Services and we will implicitly assume that our Grid is built in term of Services and implemented in terms of XML specified Web Services. This may seem unreasonable, as most of the references given above do not use Web Services. However Web Services are "just a distributed Object Model" and it has proven straightforward to convert other object models to this approach. Thus the general approach of essentially all modern GCE work can be thought of in terms of Web Services.

Programming the User View of the Grid
We will here think of application software in a simple two level hierarchy. There is "microscopic" software controlling individual CPU's and written in familiar languages like Fortran, C++ and Python. We assume that these languages generate "nuggets" or code modules and it is making these nuggets associated with a single resource that "traditional" programming addresses. To give examples, the nugget could be the SQL interface to a database, a parallel image processing algorithm or a finite element solver. This well understood (but of course still unsolved) "nugget programming" must be augmented for the Grid by the integration of the distributed nuggets together into a complete "executable". Programming the nugget internals is currently viewed as outside the Grid although projects like GrADS (Grid Application Development Software http://www.hipersoft.rice.edu/grads/) are looking at integration of individual resource (nugget) and Grid programming. Here we will assume that each nugget has been programmed and we "just" need to look at their integration. This integration is actually quite familiar and generalizes "Shell/Perl…" scripts used in single resources for UNIX operating systems and the Microsoft Com/ActiveX/…. interfaces in PC Case.

There are several other examples of this style of Grid Programming. One broad class is called Problem Solving Environments that feature a Portal Interface to a set of carefully chosen tool and application services usually customized to a particular problem domain. This has both a graphical user interface described in following section and some sort of "software bus" to link the different parts of the PSE together.

The integration of application nuggets is often called "workflow", and the user can be offered many different paradigms for expressing this. One common model is a graphical interface where one can choose nuggets from a palette and link "ports" or channels of the nuggets. This is familiar from visualization and image processing where systems like AVS (http://www.avs.com/) and Khoros (http://www.khoral.com/) are well established. Industry has developed XML specifications for this nugget linkage with approaches like BPEL4WS (Business Process Execution Language for Web Services http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/) and WSCL. (Web Services Conversation Language http://www.w3.org/TR/wscl10/ where it's the nuggets having conversations and not the users!) Simpler and perhaps more powerful is "just" to program the linkage with scripting (such as Python) or compiled (like Java) languages. We can expect it to be useful to have multiple paradigms and multiple languages and it is unlikely that any one of these is "best". Important Grid approaches for describing the programming of nuggets include the CCA (Common Component Architecture http://www.cca-forum.org/) from DoE and the ICENI project (http://www.lesc.ic.ac.uk/iceni/) of the UK e-Science Program.

The above examples indicate that "programming the user view of the Grid" has overlaps with (distributed) object technology but in this column, we are not trying to "push a particular programming model" but rather to illustrate the "issues to be addressed" and to stress the commonality of the problem being addressed with however major differences occurring in the implementations. Although related to tasks familiar from programming PC's or workstations, "Programming the user view of the Grid" is significantly more complicated. As illustrated in fig. 1, the "executable" (integrated nuggets) is a mixture of both system and application services; one uses system services on a single workstation but the meta-OS services of the Grid are currently expected to have programmable interfaces whereas many of the corresponding workstation (Windows, UNIX) services are more opaque. OGSA is part of the picture as many system services in fig. 1 will be those defined and implemented as part of the OGSA initiative. In fact perhaps all services will be OGSA services when the dust clears – certainly all will be Web Services (or whatever these become) and maybe the OGSA and Web service specifications will just merge. Currently for example portal services described in the next section come from the Web not Grid community.

Not only do we have the richness of both system and application nuggets, many Grid systems separately maintain both "real" entities (such as a software nugget) and separately entities representing the meta-data describing the "real" entity. We expect this separation to continue and indeed expand in use for there is a clear need to define more meta-data and it seems likely that this metadata will often be stored separately from the resource it describes.

As a typical nugget programming challenge, one must take into account both needed latency/bandwidth of application and network constraints (firewalls) to decide most appropriate communication mechanism between nuggets. This typically runtime specification of the implementation of a particular service-service interaction has no agreed approach. There are of course many examples of its use with particular implementation strategies. "Agents", "brokers" and "profiles" are typical of the language one often uses to describe this adaptive mechanism. In fact it seems possible that the field of agents will merge with that of the Grid. Further in developing Grid programming one has to study both

- The programming paradigm and within a paradigm one can choose particular languages – this could be scripted, visual, or compiled.
- The run-time library, which could be largely shared between different paradigms in functionality but might be expressed rather differently in each separate approach.

The many articles mentioned above are partly differentiated by their emphasis on these two different aspects of the problem.

One can borrow familiar ideas from UNIX with the basic Grid "programming primitives" usefully be expressed as a "GCE Shell". As described above, Shell primitives will be exposed to the user in different ways using different paradigms and their expression. One way of exposing the Shell primitives will be as a command line interface but in many cases one will present a higher-level view. Complete domain specific high-level systems are "just" Problem Solving Environments mentioned above. The Legion Grid system (http://legion.virginia.edu/) illustrates the GCE Shell clearly with a Legion shell naturally extending that familiar from UNIX. The GCE Shell has some features in common with the UNIX shell as for instance file manipulation is critical both in 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 at all levels of system – local client, middle-tier, backend resource
- Distinction between an object and its meta-data; copying an object might be a major high-performance task; copying the meta-data is typically a modest effort.

Looking at primitives needed, the GCE Shell needs to 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 (http://www.jxta.org)
- Meta-data handling
- Management and Scheduling
- Networks
- Negotiation primitives for service interaction

Thinking about the GCE Shell, one can simplify discussion by using a uniform service model so that files and executables are both services and not distinct as in UNIX. One probably needs 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.

The GCE shell is a catalog of the primitive functions needed to program the Grid. Grid programming paradigms are particular ways to manipulate them to build e-Science applications. Portal services described next are the way of interacting with the user. Putting this all together gives you a Problem Solving Environment.

Portal Services

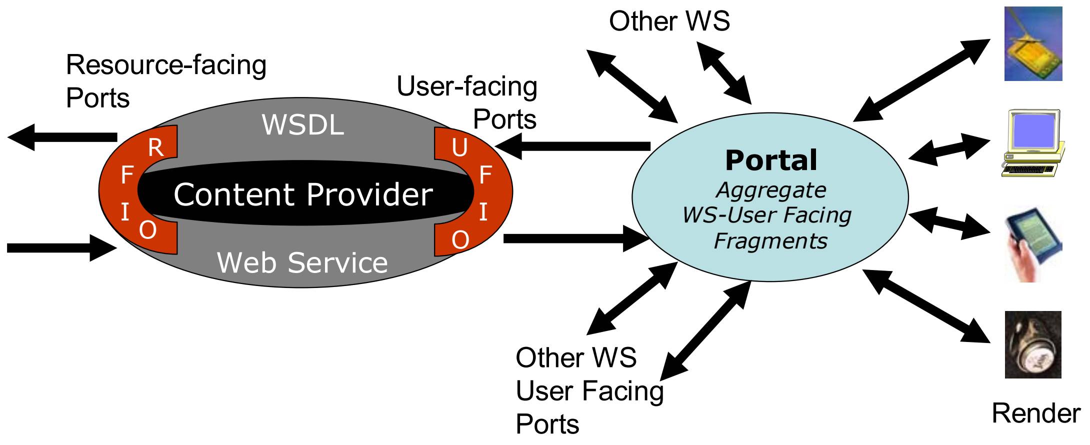
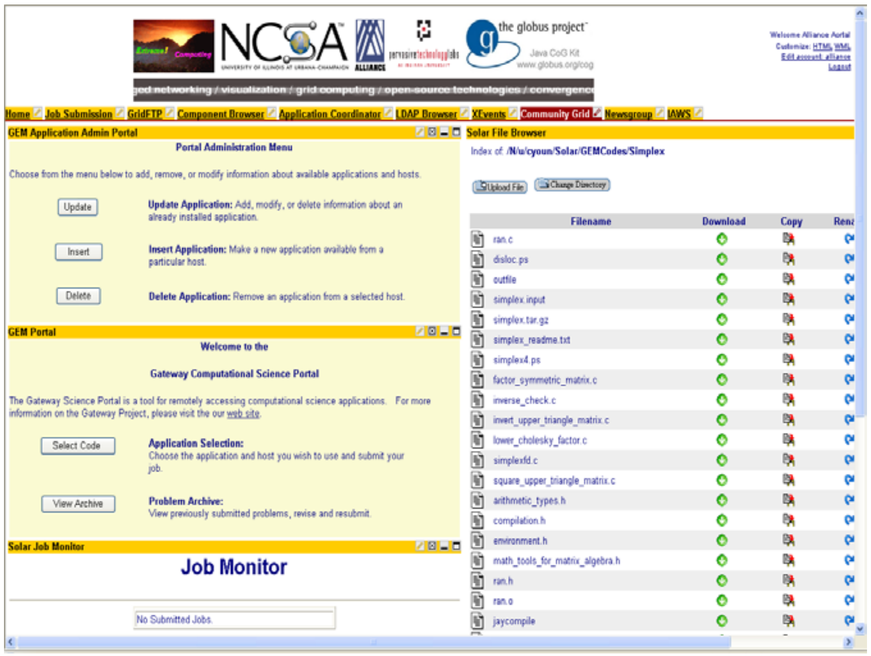


Fig. 2: Portal providing aggregation service for document fragments produced by user-facing ports of a Content providing Web Service

Portal services control and render the user interface/interaction and Fig. 2 shows a key architectural idea emerging in this area. We assume that all material presented to the user originates from a Web service which is called here a content provider. This content could come from a simulation, data repository or stream from an instrument. Each such Web Service has resource or service facing ports (RFIO in fig. 2), which are those used to communicate with other services. Here we are more concerned with the user-facing ports which produce content for the user and accept input from the client devices. These user-facing ports use an extension of WSDL, which is being standardized by the OASIS organization. This is called WSRP or Web Services for Remote Portals http://www.oasis-open.org/committees/wsrp/. It implements the so-called portlet interface, which is being standardized in Java as part of a JCP (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. One could integrate this in a custom application-specific Web service but it is attractive to provide a generic aggregation service. This allows the user and/or administrator to choose which content providers to display and what portion of the display real estate they will occupy. In this model each content provider defines its own "user-facing document fragment" which is integrated by a portal. Such aggregating portals are provided by the major computer vendors and also by Apache in its well known Jetspeed project (http://jakarta.apache.org/jetspeed/). 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 re-usability and modularity. One then has an elegant view with workflow integrating components (Web services representing nuggets) in the middle tier and aggregating portals integrating them for the user interface. Figure 3 illustrates these ideas with a



Portal Administration Menu

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

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

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

Delete Application: Remove an application from a selected host.

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.

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

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

Job Monitor

No Submitted Jobs.

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

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	●	▨	↻

Fig. 3: Example of a Jetspeed-based Portal with aggregation of interfaces to several computing services

portal being developed for the NCSA Alliance in a project led by Gannon and Plale. One sees 4 separate interfaces (3 on left and one on right) to different GCE 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 allowing convenient integration. The aggregation of the work of the different groups is provided by Web services (OGSA) in the middle tier and by systematic use of portlets at the user interface.

Fig. 4 points some other portal services which correspond to the ability of adapting rendered content to accommodate particular clients. This addresses both differences between devices (for example immersive versus desktop versus handheld) and issues of universal access – accommodating to possible physical limitations of the user. The architecture of fig. 2 becomes more complex as now one needs a negotiation between client and content provider to define the rendered view. This requires a portal selection service to process user profiles and choose appropriate content. One also can package common filters to for example reduce resolution for a multi-media content. This work on universal access is familiar in audio-video conferencing (protocols like H323 negotiate "best" codecs to fit client) and is being pursued by W3C as part of its accessibility initiative.

The collection of aggregator, selector and filtering capabilities illustrate common portal services that can be shared by multiple Grid applications.

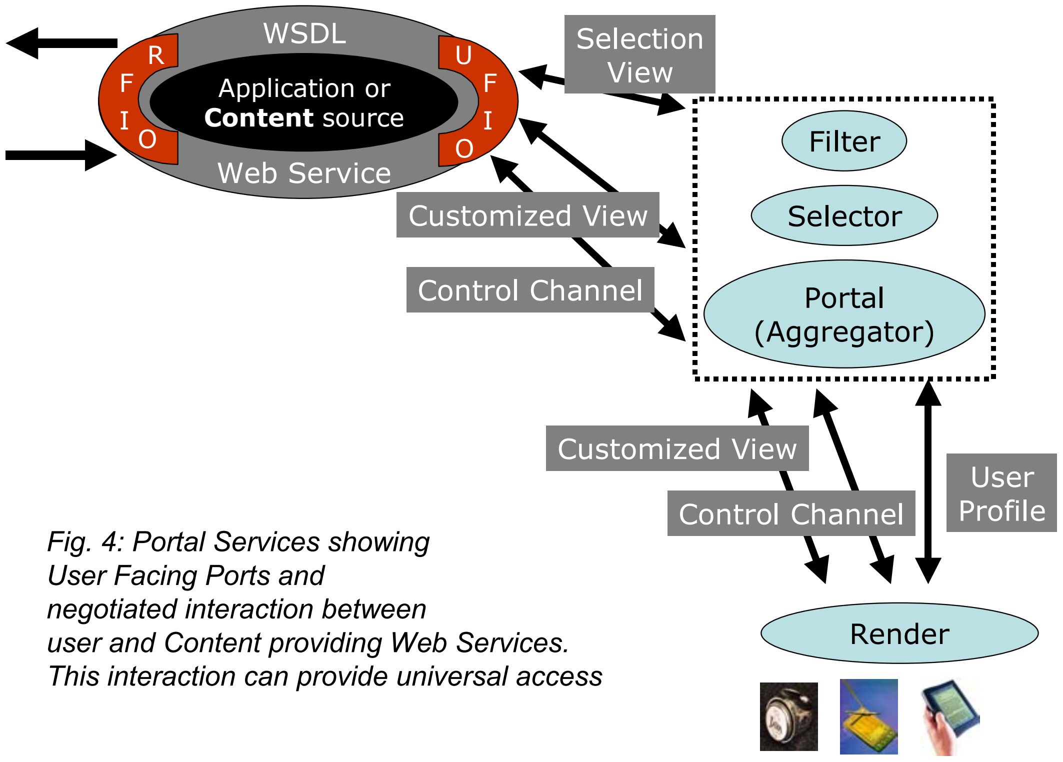


Fig. 4: Portal Services showing User Facing Ports and negotiated interaction between user and Content providing Web Services. This interaction can provide universal access