GWD-C
Category: Community Practice
Grid Computing Environments - RG

G. Fox, Community Grids Lab, Indiana University
M. Pierce, Community Grids Lab, Indiana University
D. Gannon, CS & PTL, Indiana University
M. Thomas, TACC, University of Texas, Austin
2-15-03

Overview of Grid Computing Environments

Status of This Memo

This memo provides information to the Grid community interested in portal access to Grid systems. Distribution is unlimited.

Copyright Notice

Abstract

We present a survey of best practice in Grid Computing Environments coming from a study of some 50 papers. We abstract this best practice in terms of architectural principles – multi-tier service-based model, role of meta-data, workflow, tools and core functionalities forming a GCEShell and aggregation portals. We expect many of these will be further refined in separate documents.

.

Contents

gcf@indiana.edu, marpierc@indiana.edu,
gannon@cs.indiana.edu, mthomas@tacc.utexas.edu

1. Introduction

This document summarizes the current status of Grid Computing Environments. It integrates 15 chapters [38-52] of a recent book [37] with a survey [36, 38] of a set of 29 papers [1-28] gathered together by the GCE (Grid Computing Environment) group [55] of the Global Grid Forum, which was published in 2002 as a special issue of the journal Concurrency and Computation: Practice and Experience [54]. 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. Grid Computing Environments (GCE) roughly describe the "user side" of a computing system which 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) [56] (which is itself evolving) describes these "Core" capabilities and the Globus project [32] is the best known "Core" software project. We can define a Grid Computing Environment as a set of tools and technologies that allow users "easy" access to Grid resources and applications. Often it appears to the user as a Web portal that provides the user interface to a multi-tier Grid application development stack, but it may also be as simple as a Grid Shell that allows a user access to and control over Grid resources in the same way a conventional shell allows the user access to the file system and process space of a regular operating system. The different papers summarized for this document all imply a diagram similar to figure 1 but differ in technology used (Perl versus Python for example), capability discussed and the emphasis on user versus program (back end resource) view.

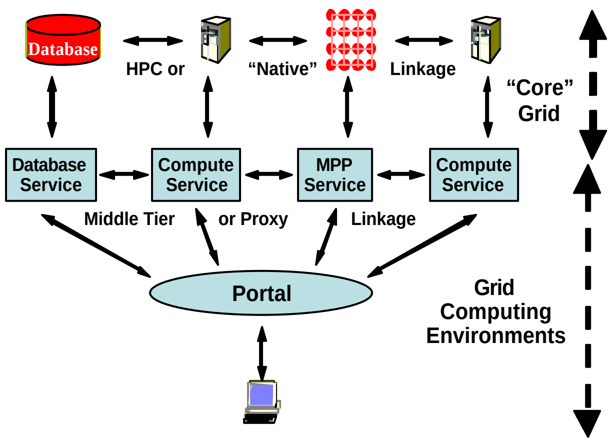


Fig. 1. Middle-Tier and Raw (HPC) Linked Components of an Application showing both the "Core" and Computing Environments

As discussed above, GCE's fulfill (at least) two functions –

- "Programming the User Side of the Grid" which is the topic discussed in sections 2-4 of this document.
- 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. This aspect of GCE's is presented in section 5.

2. Overall Classification of GCE Systems

Grid Computing Environments can be classified in several different ways. One straightforward classification is in terms of technologies used. The different projects differ in terms of languages used, nature of treatment of objects (if any), use of particular technology like Java servlets, the Globus toolkit, or GridFTP, and other implementation issues. Some of these issues are important for performance or architecture but often can look to the user as not so important. For instance, there is a trend to use more heavily Java, XML and Web Services but this will only be interesting if the resultant systems have important properties such as better customizability, sustainability and ease of use without sacrificing too much in areas like performance. The ease of development using modern technologies often yields greater functionality in the GCE for a given amount of implementation effort. Technology differences in the projects are important but more interesting at this stage are the differences in capabilities and the model of computing explicit or implicit in the GCE.

All GCE systems assume there are some backend remote resources (the Grid) and endeavor to provide convenient access to their capabilities. This implies one needs some sort of model for "computing". At the simplest this is running a job, which already has non trivial consequences as data usually needs to be properly set up, and access is required to the running job status and final output. More complex examples require coordinated gathering of data, many simulations (either linked at a given time or following each other), visualization, analysis of results etc. Some of these actions require substantial collaboration between researchers and sharing of results and ideas are needed. This leads to the concept of GCE collaboratories supporting sharing among scientific teams working on the same problem area.

We can build a picture of different GCE approaches by viewing the problem as some sort of generalization of the task of computing on a single computer. So we can highlight the following classes of features:

1) Handling of the basic components of a distributed computing system – files, computing and data resources, programs, and accounts. The GCE will typically interface with an environment like Globus or a batch scheduler like PBS to actually handle the back-end resources. However the GCE will present the user interfaces to handle these resources. This interface can be simple or complex and often constructed hierarchically to reflect tools built in such a fashion. We can follow the lead of UNIX (and Legion [43] in its distributed extension) and define a basic GCEShell providing access to the core distributed computing functions. For example, JXTA [35] also builds Grid-like capabilities with a UNIX shell model. GCEShell would support running and compiling jobs, moving among file systems etc. GCEShell can have a command line or more visually appealing graphical user interface.
2) The 3-tier model of fig. 1, which is typically used for most systems, implies that any given capability (say run a matrix inversion program) can appear at multiple levels. Maybe there is a backend parallel computer running an MPI job; this is front-ended perhaps as a service by some middle-tier component running on a totally different computer, which could even be in a different security domain. One can "interact" with this service at either level; a high performance I/O transfer at the parallel computing level and/or by a slower middle-tier protocol like SOAP at the service level. These two (or more) calls (component interactions) can represent different functions or the middle tier call can be coupled with a high performance mirror; typically the middle tier provides control and the back end "raw data transfer". The resultant rather complicated model is shown in fig.1. We have each component (service) represented in both middle and HPC (raw) tiers. Intra-tier and inter-tier linkage is shown. Ref. [39], Programming the Grid, has an excellent review of the different programming models for the Grid.
3) One broadly important general-purpose feature is Security (authentication, authorization and privacy), which is addressed in some way or other by essentially all environments.
4) Data management is a another broadly important topic, which gets even more important on a distributed system than it is on single machines. It includes file manipulation, databases and access to raw signals from instruments such as satellites and accelerators.
5) One augments the basic GCEShell with a library of other general purpose tools and this can be supported by the GCE. Such tools include (Grid)FTP, (Grid)MPI, parameter sweep and more general workflow, and the composition of GCEShell primitives.
6) Other higher-level tools are also important and many tend to be rather application dependent; visualization and intelligent decision support as to what type of algorithm to use can be put here.
7) Looking at commercial portals, one finds that they usually support sophisticated user interfaces with multiple sub-windows aggregated in the user interface. The Apache Jetspeed project is a well-known toolkit supporting this [33]. This user interface aggregation is often supported by a GCE. This aggregation is described in the final section 5.

As well as particular features, a GCE usually implies a particular computing model for the Grid and this model is reflected in the GCE architecture and the view of the Grid presented to the user. For example object models for applications are very popular and this object view is reflected in the view of the Grid presented to the user by the GCE. Note the programming model for a GCE is usually the programming of rather large objects – one can describe programs and hardware resources as objects without this object model necessarily changing the software model used in applications.

With this preamble, we can now classify the papers summarized for this document. There are, as always, no absolute classifications for a complex topic like distributed Grid systems. Hence it is often the case that these projects can be looked at from many overlapping points of view.

3. Summary of GCE Projects and Features

3.1 Technology for building GCE Systems

In the previous section of this book we have described the basic architecture and technologies needed to build a Grid and we have described the basic component for the different types of GCEs above. As previously mentioned, ref. [39] provides an excellent overview of many of the back-end application programming issues.

The Globus toolkit [32] is the most widely used Grid middleware system, but it does not provide much direct support for building GCEs. Refs. [6, 14, 15, 27] and [44] describe respectively Java, CORBA, Python and Perl Commodity Grid interfaces to the Globus toolkit. These provide the basic building blocks of full GCE's. Ref. [1] describes the Grid Portal Development Tooklit (GPDK), a suite of JavaBeans suitable for Java based GCE environments; the technology is designed to support JSP (Java Server Pages) displays. Together, the COG Kits and GPDK constitute the most widely used frameworks for building GCEs that use the Globus environment for basic Grid services. The problem solving environments in Refs. [7], [8] and [20] build on top of the Java Commodity Grid Kit [6]. The portals described in ref. [51] build directly on top of the Perl Commodity Grid Kit [27].

Another critical technology for building GCEs is a notification/event service. Ref. [21] notes that current Grid architectures build more and more on message-based middleware and this is particularly clear for Web Services; this paper designs and prototypes a possible event or messaging support for the Grid. Refs. [21,49] describes the Narada Brokering system, which leverages peer-to-peer technology to provide a framework for routing messages in the wide-area. This is extremely important in cases where the GCE must cross the trust boundaries between the user's environment and the target Grid.

Ref. [9] provides C support for interfacing to the Globus toolkit and portals exposing the toolkit's capabilities can be built on the infrastructure of this paper. Ref. [17] proposes interesting XML based technology for supporting the runtime coupling of multidisciplinary applications with matching of geometries. Ref. [28] describes a rather different technology; namely a Grid simulator aimed at testing new scheduling algorithms.

3.2 Largely Problem Solving Environments

We have crudely divided those GCE's offering user interfaces into two classes. One class focusing on a particular application (set) which are sometimes called application portals or Problem Solving Environments (PSE's). The second class offer generic application capabilities and have been termed user portals; in our notation introduced above, we can call them GCEShell portals. Actually one tends to have a hierarchy with PSE's building on GCEShell portals; the latter building on middleware like GPDK; GPDK builds on the Java CoG Kit [6] which itself builds on the Globus toolkit that finally builds on the native capabilities of the Grid component resources.

This hierarchy is for one set of technologies and architecture but other approaches are similarly built in a layered fashion.

Several papers surveyed include discussion of Grid PSE's. Ref. [5] has an interesting discussion of the architectural changes to a "legacy" PSE consequent on switching to a Grid Portal approach. Ref. [11] illustrates the richness of PSE with a survey of several operational systems; these share a common heritage with the PSE's of Ref. [16] although the latter paper is mainly focused on a recommender tool described later.

Five further papers describe PSE's that differ in terms of GCE infrastructure used and applications addressed. Ref. [7] describes two PSE's built on top of a GCEShell portal with an object computing model. A similar portal is the XCAT Science portal [29], which is based on the concept of application Notebooks that contain web pages, Python scripts and control code specific to an application. In this case the Python script code plays the role of the GCEShell. The astrophysics collaboratory [20] includes the Globus toolkit link via Java [6] and the GPDK [1]; it also interfaces to the powerful Cactus distributed environment [31]. Ref. [18] and [47] presents a portal for computational physics using Web services – especially for data manipulation services. The Polder system [24] and SCIRun [25] offer rich visualization capabilities within several applications including biomedicine. SCIRun has been linked to several Grid technologies including NetSolve [10], and it supports a component model (the CCA [34] which is described in ref. [53]) with powerful workflow capabilities.

The Discover system described in ref. [48] describes a PSE framework that is built to enable computational steering of remote Grid applications. This is also an important objective of the work on Cactus described in ref. [42], Classifying and Enabling Grid Applications.

3.3 Largely Basic GCEShell Portals

Here we describe the set of portals designed to support generic computing capabilities on the Grid. Ref. [3] is interesting as it is a Grid portal designed to support the stringent requirements of DoE's ASCI program. This reflects not only security and performance issues but the particular and well established computing model for the computational physicists using the ASCI machines. Ref. [4] describes a portal interface to the very sophisticated Legion Grid which has through the Legion Shell a powerful generic interface to the shared object (file) system supported by Legion [43]. This paper also describes how specific problem solving environments can be built on topic of the basic GCEShell portal.

Unicore [23] was one of the pioneering full featured GCEShell portals developed originally to support access to a specific set of European supercomputers but recently has been interfaced to the Globus toolkit and, as described in ref. [50], to the Open Grid Services Architecture [56]. Unicore has developed an interesting abstract job object (AJO) with full workflow support.

Refs. [7, 13, 45] describe well developed GCEShell portals technology on which several application specific PSE's have been built. Ref. [51] describes the NPACI Grid Portal toolkit, GridPort. which is middleware using the Perl Community Grid Kit [27] to access the Globus toolkit. Ref. [26] also describes HotPage, a GCEShell built on top of GridPort.

3.4 Workflow

Workflow corresponds to composing a complete job from multiple distributed components. This is broadly important and is also a major topic within the commercial Web service community. It is also inherently a part of a GCEShell or PSE, since these systems are compositions of specific sequences of tasks. Several projects have addressed this but currently there is no consensus how workflow should be expressed, although several groups have developed visual user interfaces to define the linkage between components. Workflow is discussed in papers [3], [8], [17], [23] and [25]. The latter integrates Grid workflow with the dataflow paradigm, which is well established in the visualization community. BPEL4WS is an important new workflow proposed standard [60] that may have a large impact on the Grid community. Ref. [17] has stressed the need for powerful runtime to support the coupling of applications and this is implicit in other papers including Ref. [8]. We discuss this further in section 4.2.

3.5 Data Management

Data intensive applications are expected to be critical on the Grid but support of this is not covered in this report. Interfaces with file systems, databases and data transfer through mechanisms like GridFTP are covered in several papers. This is primarily due to the fact that data management software is still relatively new on the grid. Ref. [47] describes a SOAP based web service and a portal interface for managing data used within a large scientific data grid project. Almost all modern Grid portals have GridFTP components that allow users to use the portal to upload and download files and move them from one place to another.

3.6 GCEShell Tools

In our GCE computing model, one expects a library of tools to be built up that add value to the basic GCEShell capabilities. The previous two subsections describe two tools – workflow and data management of special interest and here we present a broad range of other tools which appeared in several papers in the Grid Computing Environments special issue.

Netbuild [2] supports distributed libraries with automatic configuration of software on the wide variety of target machines on the Grids of growing heterogeneity. NetSolve [10, 46] pioneered the use of agents to aid the mapping of appropriate Grid resources to client needs. Ref. [16] describes a recommendation system, which uses detailed performance information to help users on a PSE, choose the best algorithms to address their problem.

Many projects have noted the importance of "parameter sweep" problems where a given application is run many times with different input parameters. Such problems are very suitable for Grid environments and Ref. [22] describes a particular parameter sweep system Nimrod-G. This paper focuses on a different tool – namely a novel scheduling tool based on an economic model of Grid suppliers and consumers. Ref. [40] describes a another well regarded parameter sweep system APST building on the AppLeS application level scheduling system.

HotPage described in Refs. [26 and 51], is well known for pioneering the provision of job status information to portals; such a tool is clearly broadly important.

We should stress visualization as a critical tool for many users and here Refs. [25] and [17] describe this area. There are many other important tools like data-mining which fall into this category of sophisticated Grid analysis capabilities.

We can perhaps provide some sense of order to this area by borrowing familiar ideas from UNIX with the basic Grid "programming primitives" usefully be expressed as a "GCE Shell" introduced earlier. 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 mentio ned above. The Legion Grid system [4] 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 services 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 [35]
- 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 at its heart "just" a catalog of the primitive functions needed to program the Grid. In fact, the list above is a subset of the core services that are part of OGSA. Grid programming paradigms are particular ways to manipulate them to build applications. Portal services described in sections 3.3 and 5 are the way of interacting with the user. Putting this all together gives you a Problem Solving Environment as discussed in section 3.2.

4. GCE Computing Model

4.1 Survey of GCE Models

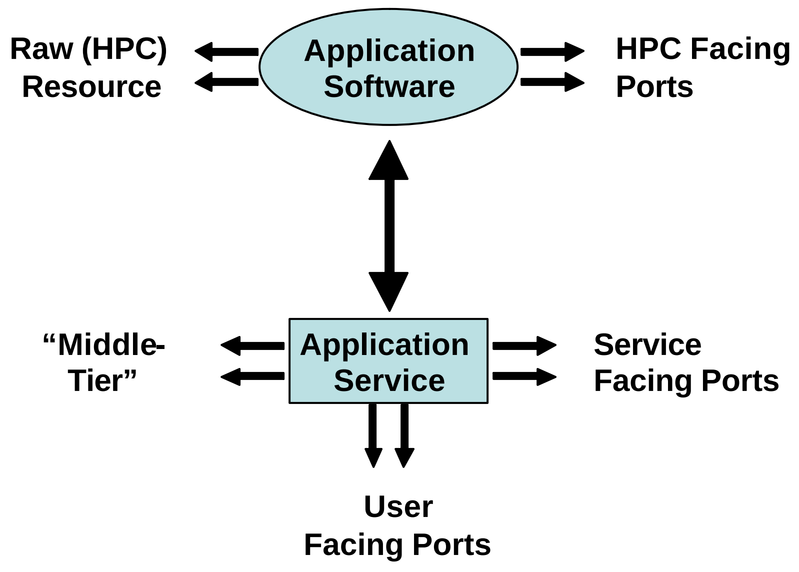


Figure 2. A Proxy Service Programming Model showing 4 types of Interactions :to and from users (portal interface), between proxy and raw resource, other middle-tier components and between other raw (hpc) resources

In the preamble we suggested that it was interesting to consider the computing model underlying Grid Computing Environments. This refers to the way we think about the world of files, computers, databases and programs exposed through a portal. NetSolve described in Refs. [10 and 46] together with the Ninf effort, refs. [30 and 41], have developed the important Network Service model for distributed computing. Rather than each user downloading a library to solve some part of their problem, this task is dispatched to a Network resource providing this computational service. Both Ninf and NetSolve support the new GridRPC remote procedure call standard, which encapsulates a key core part of their Grid computing model described in ref. 39. GridRPC supports scientific data structures as well as Grid specific security and resource management.

Ref. [12] describes Grid implementations of MPI (message passing standard for parallel computing), which address the incompatibilities between MPI implementations and binary representations on different parallel computers. Note that in the notation of Fig. 1, MPI is at the "HPC backend linkage" layer and not at the middleware layer. Ref. [20] supports the Cactus environment [31, 42] which has well developed support for Grid computing at the HPC layer i.e. it supports backend programming interfaces and not the middle-tier GCEShell environment. The astrophysics problem solving environment of Ref. [20] augments Cactus with a full middle tier environment.

Legion described in refs. [4, 43], built a very complete Grid object model. Ref. [8] describes a CORBA distributed object model for the Grid and Refs. [19 and 48] describes the surprisingly hard issues involved in providing interoperability between multiple CORBA GCE's. We can hope that Web services will prove to be easy to make interoperable, as the technology used (XML, SOAP) is more open than CORBA, which has evolved with several often incompatible implementations as listed in Ref. [14].

Refs. [7, 13, 23, 45, 50] and the XCAT Science Portal [29, 53] also present an object model for GCE computing but with one critical feature – namely the middle tier objects are always proxies, which hold the meta-data that describe "real resources" which operate in conventional environments. This proxy strategy appears useful for many Grid resources although the true Network service model of NetSolve is also essential. Let us give a simple example from UNIX and suppose one wanted to send data between two programs (in different machines). One could choose the mechanism within the program and use a simple socket or FTP or RMI interaction mechanism. Alternatively the programs could be written generically with output and input or "standard I/O". The programs could then have the output of one "piped" to the input of the other from a UNIX shell command. Such a hybrid programming model with actions partly specified internally and partly specified at service level is important of the success of the Grid and should be built into programming models for it.

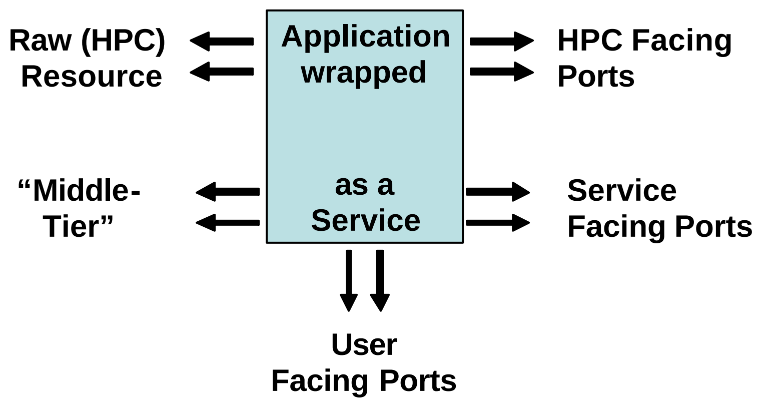


Figure 3. A Wrapped Application Programming Model showing 3 types of Interactions to and from users (portal interface) to and from other middle-tier components and between other raw (HPC) resources

Any GCE computing model should support both the meta-data only and wrapped styles of Grid objects. Actually going back to point 2) in Section 2, the proxy and NetSolve model are not really different as indicated in figures 2 and 3. Both models effectively wrap application (software) resources as objects. In the proxy model, one exposes the interaction between middle-tier and back-end. In the wrapped service model of NetSolve and Ninf, one presents a single entity to the user. In both cases, one can have separate middle-tier and HPC ("real") communication. To complicate the classification, there can of course be a difference between programming model abstraction (proxy or not) and implementation. In the XCAT model, a software component system [53] is used which implements the wrapped service or proxy model. The component system is based on Web Service standards so it is possible that the wrapped service components may be arbitrary Grid Services.

An additional aspect of the computing model that must be addressed by GCE systems is the way in which resources are managed. In refs. [22, 52], Grid Resource Allocation and Control Using Computational Economies, the authors make the case for an economic model of resource allocation and provisioning. There is a good chance that such approaches will be used as we scale Grid system to very large sizes.

4.2 Two-level Programming Model

We can usefully discuss some GCE computing models by thinking 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 [57] 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 manifestations of this style of Grid Programming. One broad class is the Problem Solving Environments of section 3.2 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 section 5 and some sort of "software bus" to link the different parts of the PSE together.

The integration (or software bus) of application nuggets is often called "workflow", and as discussed in section 3.5 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 [58] and Khoros [59] are well established. Industry has developed XML specifications for this nugget linkage with approaches like BPEL4WS (Business Process Execution Language for Web Services [60]) and WSCL (Web Services Conversation Language [61] 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 (this is the Software "bus" idea). 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 [34, 53]) from DoE and the ICENI project [62] 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 document, 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.

Not only do we have the richness of both system and application nuggets, many Grid

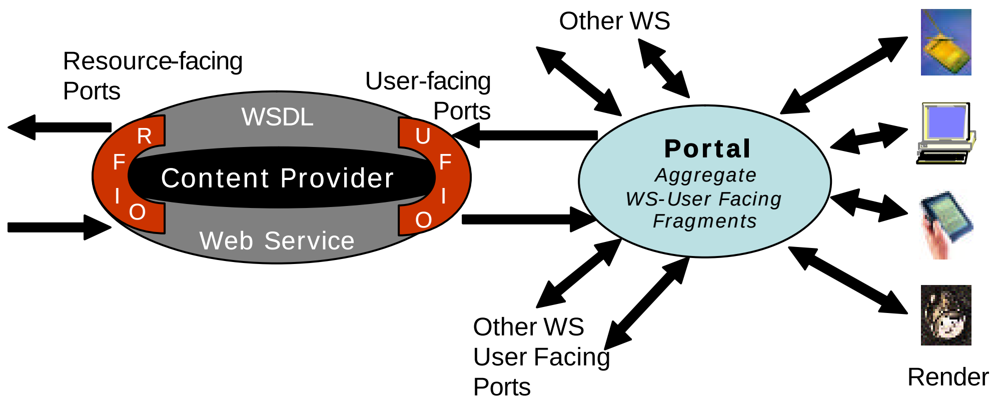


Fig. 4: Portal providing aggregation service for document fragments produced by user-facing ports of a Content providing Web Service

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

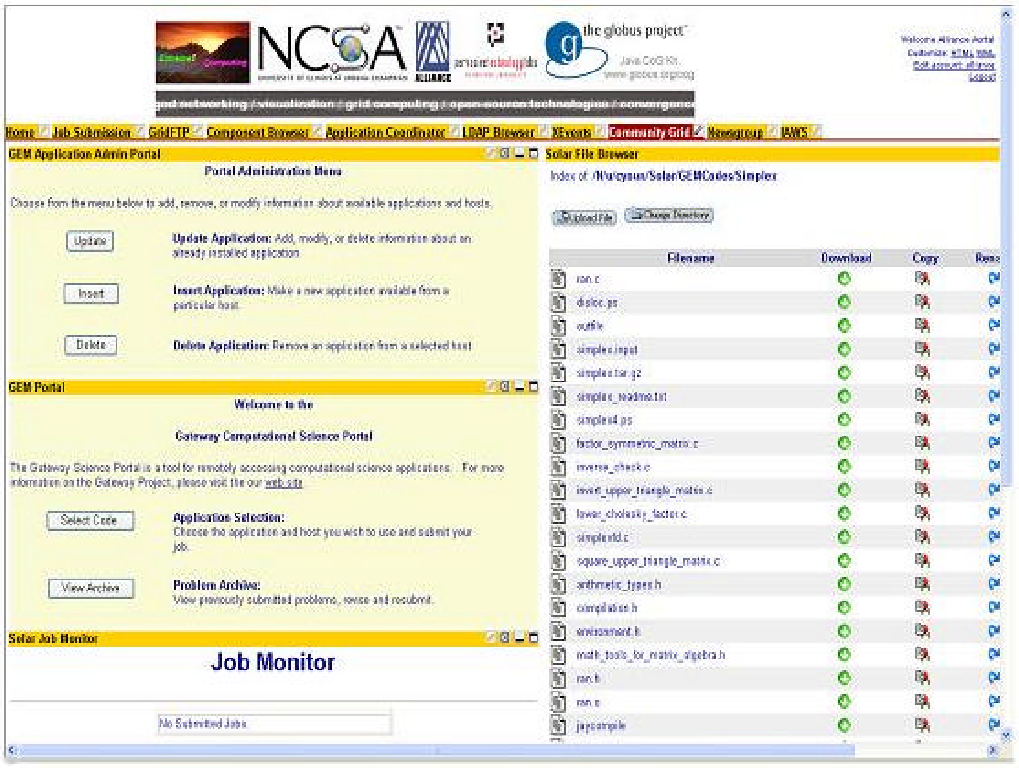
- The programming paradigm and within a paradigm one can choose particular languages – this could be scripted, visual, or compiled.
- The run-time library, which could be largely shared between different paradigms in functionality but might be expressed rather differently in each separate approach.

The many articles discussed in this document are partly differentiated by their emphasis on these two different aspects of the problem.

5. Portal Services

Portal services control and render the user interface/interaction and Fig. 4 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. 4), 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).

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 5 illustrates these ideas with a portal being developed for the NCSA Alliance in a project led by Gannon and Plale. One sees four separate interfaces (3 on left and one on right) to different GCE Web services. Each of these Web services can be associated with one or more GCEShell capabilities. 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.



GEM Application Admin Portal

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

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

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

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

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	●	●	●
simplexld.c	●	●	●
square_upper_triangle_matrix.c	●	●	●
arithmetic_types.h	●	●	●
compilation.h	●	●	●
environment.h	●	●	●
math_tools_for_matrix_algebra.h	●	●	●
ran.h	●	●	●
ran.c	●	●	●
jaycompile	●	●	●

Fig. 5: Example of a Jetspeed-based Portal with aggregation of interfaces to several computing services

Fig. 6 points out 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. 4 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.

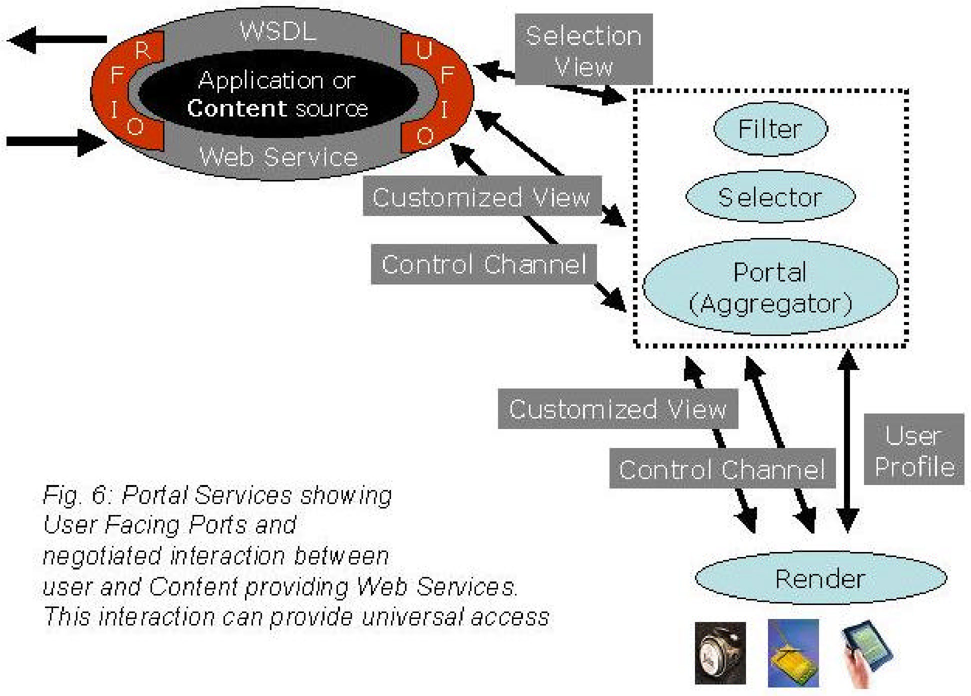


Fig. 6: Portal Services showing User Facing Ports and negotiated interaction between user and Content providing Web Services. This interaction can provide universal access

5.1 The Open Grid Service Architecture Implications for GCE Portals

OGSA consists of a set of core Grid web services defined in terms of the Open Grid Service Infrastructure (OGSI) specification. An OGSI compliant Grid web service defines a subclass of web services whose ports all inherit from a standard Grid Service port. Using this port there are standard ways that a remote portal can interrogate the service to discover such things as the other port types the service implements and what operations can be made on those ports and the public internal state of the service. OGSI services can also implement a simple event subscription and notification mechanism in a standard way. OGSI also provides a mechanism for services to be group together into service collections. The simple and standard nature of OGSI makes it possible for us to build on-the-fly compilers to generate portal portlets interfaces to any OGSI compliant grid service.

The core services defined by OGSA include registries, directories and namespace binding, security, resource descriptions and resource services, reservation and scheduling, messaging and queuing, logging, accounting, data services (caches and replica managers), transaction services, policy management services, workflow management and administration services. Each of these core services is rendered as a Grid web service. (At the time of this writing, this list is probably incomplete and it is certainly not yet official.) Applications that are designed for an OGSA compliant Grid can assume that these services are available and, with the proper authorization, that they can be used.

As part of the standard GGF-GCE portal framework, we can build and distribute portlets for accessing both the client and management ports for these services. OGSA will provide a natural and easy to use building block platform for both GCE portals and applications.

6. Security Considerations

One of the primary tasks of any Grid portal is to manage secure access to Grid resources. Consequently security is discussed in most papers on this topic. The GCEs based on Globus and Legion use the Public Key Infrastructure. Kerberos is required by some installations (DoD and DoE for instance in the USA) and Grid Computing Environments developed for them [3] [7] [13] are based on this security model.

Many current portals rely on https connections between the users browser and portal server. However, identity certificates are often stored on a trusted third part known as a "MyProxy" server. The user is required to store a proxy certificate, which is valid for a day or so, on the MyProxy server with a private password. The portal server can then request the password from the user and then ask MyProxy server for a copy of the proxy certificate. Once the portal has a copy of the proxy certificate, it can then delegate it to the po rtlets to use on behalf of the user when interacting with remote Grid services.

Security is of fundamental importance to GCE portals and we will continue to observe the progress of the GGF OGSI security working group.

Author Information

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

Marlon Pierce
Community Grid Computing Laboratory,
Indiana University
501 N Morton Suite 224
Bloomington IN 47404
marpierc@indiana.edu

Dennis Gannon
Department of Computer Science
Lindley Hall 215
Indiana University
gannon@cs.indiana.edu

Mary Thomas
Texas Advanced Computing Center, The
University of Texas at Austin, 10100 Burnet
Road,
Austin, Texas 78758
mthomas@tacc.utexas.edu

Glossary

The Glossary will be added in the next draft.

Intellectual Property Statement

Full Copyright Notice

References

1) Jason Novotny, “The Grid Portal Development Kit”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1129-11444, 2002. This article is also found as Chapter 27, pages 657-674 of Ref. [37].
2) Keith Moore and Jack Dongarra, “NetBuild: Transparent Cross-Platform Access to Computational Software Libraries”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1445-1456, 2002.

3) Randal Rheinheimer, Steven L. Humphries, Hugh P. Bivens and Judy I. Beiriger, “The ASCI Computational Grid: Initial Deployment”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1351-1364, 2002.

4) Anand Natrajan, Anh Nguyen-Tuong, Marty A. Humphrey and Andrew S. Grimshaw, “The Legion Grid Portal”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, 1365-1394, 2002.

5) Karen Schuchardt, Brett Didier and Gary Black , “Ecce - A Problem Solving Environment's Evolution Toward Grid Services and a Web Architecture”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1221-1240, 2002.

6) Gregor von Laszewski, Jarek Gawor, Peter Lane, Nell Rehn, and Mike Russell “Features of the Java Commodity Grid Kit”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1045-1056, 2002.

7) Tomasz Haupt, Purushotham Bangalore and Gregory Henley, “Mississippi Computational Web Portal”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1275-1288, 2002.

8) Andreas Schreiber, ”The Integrated Simulation Environment TENT”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1553-1568, 2002.

9) Giovanni Aloisio and Massimo Cafaro, “Web-based access to the Grid using the Grid Resource Broker Portal”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1145-1160, 2002.

10) D. Arnold, H. Casanova, and J. Dongarra, “Innovations of the NetSolve Grid Comput ing System”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1457-1480, 2002.

11) Naren Ramakrishnan, Layne T. Watson, Dennis G. Kafura, Calvin J. Ribbens, and Clifford A. Shaffer, “Programming Environments for Multidisciplinary Grid Communities”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1241-1274, 2002.

12) M. Mueller , E. Gabriel and M. Resch, “A Software Development Environment for Grid Computing”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1543-1552, 2002.

13) Marlon. E. Pierce, Choonhan Youn, and Geoffrey C. Fox, “The Gateway Computational Web Portal”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1411-1426, 2002.

14) Gregor von Laszewski, Manish Parashar, Snigdha Verma, Jarek Gawor, Kate Keahey, and Nell Rehn, “A CORBA Commodity Grid Kit”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1057-1074, 2002.

15) Keith Jackson “pyGlobus: A Python interface to the Globus Toolkit”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1075-1084, 2002.

16) E. Houstis, A. C. Catlin, N. Dhanjani and J. R. Rice, N. Ramakrishnan and V. Verykios, “MyPYTHIA: A Recommendation Portal for Scientific Software and Services”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1481-1506, 2002.

17) erry A. Clarke and Raju R. Namburu, “A Distributed Computing Environment for Interdisciplinary Applications”, Concurrency and Com putation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1161-1174, 2002.

18) William A. Watson III , Ian Bird, Jie Chen, Bryan Hess, Andy Kowalski and Ying Chen, “A Web Services Data Analysis Grid”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1303-1312, 2002.

19) Vijay Mann and Manish Parashar, “Engineering an Interoperable Computational Collaboratory on the Grid”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1569-1594, 2002.

20) Gregor von Laszewski, Michael Russell, Ian Foster, John Shalf, Gabrielle Allen, Greg Daues, Jason Novotny and Edward Seidel, “Community Software Development with the Astrophysics Simulation Collaboratory”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1289-1302, 2002.

21) Geoffrey Fox and Shrideep Pallickara, “An Event Service to Support Grid Computational Environments”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1097-1128, 2002.

22) Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger, “Economics Paradigm for Resource Management and Scheduling in Grid Computing”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1507-1542, 2002.

23) Dietmar W. Erwin, “UNICORE – A Grid Computing Environment”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1395-1410, 2002.

24) K. A. Iskra,R. G. Belleman, G. D. van Albada, J. Santoso, P. M. A. Sloot, H. E. Bal, H. J. W. Spoelder and M. Bubak, “The Polder Computing Environment: a system for interactive distributed simulation”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, 1313-1336, 2002.

25) Chris Johnson , Steve Parker and David Weinstein, “Component-Based Problem Solving Environments for Large-Scale Scientific Computing”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1337-1350, 2002.

26) M. Thomas, M. Dahan, K. Mueller, S. Mock, C. Mills,and R. Regno, “Application Portals: Practice and Experience”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1427-1444, 2002.

27) S. Mock, M. Dahan, M. Thomas and G. von Lazewski, “The Perl Commodity Grid Toolkit”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1085-1096, 2002.

28) Manzur Murshed, Rajkumar Buyya, and David Abramson, “GridSim: A Toolkit for the Modeling and Simulation of distributed resource management and scheduling for Grid Computing”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1175-1220, 2002.

29) S. Krishnan, R. Bramley, M. Govindaraju, R. Indurkar, A. Slominski, D. Gannon, J. Alameda and D. Alkaire ``The XCAT Science Portal,'', Proceedings SC2001, Nov. 2001, Denver.

30) Ninf network server project http://ninf.apgrid.org/

31) Cactus Grid Computational Toolkit http://www.cactuscode.org

32) The Globus Grid Project http://www.globus.org

33) Apache Jetspeed Portal http://jakarta.apache.org/jetspeed/site/index.html

34) Common Component Architecture http://www.cca-forum.org/

35) JXTA Peer-to-Peer Environment http://www.jxta.org

36) G.C. Fox, D. Gannon and M. Thomas, “Editorial: A Summary of Grid Computing environments”, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1035-1044, 2002.

37) Fran Berman, Geoffrey Fox and Tony Hey, ‘Grid Computing: Making the Global Infrastructure a Reality’, ISBN 0-470-85319-0, John Wiley & Sons Ltd, Chichester, 2003. See http://www.grid2002.org

38) Geoffrey Fox, Dennis Gannon, and Mary Thomas, ‘Overview of Grid computing environments’, Chapter 20, pages 543-554 in Ref. [37].

39) Craig Lee and Domenico Talia, ”Grid Programming Models: Current Tools, Issues and Directions”, Chapter 21, pages 555-578 in Ref. [37].

40) Henri Casanova and Fran Berman, "Parameter Sweeps on the Grid with APST", Chapter 33, pages 773-788 in Ref. [37].
41) Hidemoto Nakada, Yoshio Tanaka, Satoshi Matsuoka and Staoshi Sekiguchi, "Ninf-G: a GridRPC system on the Globus Toolkit", Chapter 25, pages 625-638 in Ref. [37].
42) Gabrielle Allen, Tom Goodale, Michael Russell, Edward Seidel and John Shalf, "Classifying and enabling Grid applications", Chapter 23, pages 601-614 in Ref. [37].
43) Andrew S. Grimshaw, Anand Natrajan, Marty A. Humphrey, Michael J. Lewis, Anh Nguyen-Tuong, John F. Karpovich, Mark M. Morgan and Adam J. Ferrari, "From Legion to Avaki: The Persistence of Vision", Chapter 10, pages 265-298 in Ref. [37].
44) Gregor von Laszewski, Jarek Gawor, Sriram Krishnan and Keith Jackson, "Commodity Grid Kits - Middleware for Building Grid Computing Environments", Chapter 26, pages 639-658 in Ref. [37].
45) Tomasz Haupt and Marlon E. Pierce, "Distributed object-based grid computing environments", Chapter 30, pages 713-728 in Ref. [37].
46) Sudesh Agrawal, Jack Dongarra, Keith Seymour, and Sathish Vadhiyar, "NetSolve: Past, Present, and Future; A Look at a Grid Enabled Server", Chapter 24, pages 615-624 in Ref. [37].
47) William A. Watson, Ying Chen, Jie Chen and Walt Akers, "Storage Manager and File Transfer Web Services", Chapter 34, pages 789-804 in Ref. [37].
48) V. Mann and M. Parashar, "DISCOVER: A Computational Collaboratory for Interactive Grid Applications", Chapter 31, pages 729-746 in Ref. [37].
49) Geoffrey Fox and Shrideep Pallickara, "NaradaBrokering: An Event Based Infrastructure for Building Scaleable Durable Peer-to-Peer Grids", Chapter 22, pages 579-600 in Ref. [37].
50) David Snelling, "Unicore and the Open Grid Services Architecture", Chapter 29, pages 701-712 in Ref. [37].
51) Mary Thomas and Jay Boisseau, "Building Grid Computing Portals: The NPACI Grid Portal Toolkit", Chapter 28, pages 675-700 in Ref. [37].
52) Rich Wolski, John Brevik, James S. Plank, and Todd Bryan, "Grid resource allocation and control using computational economies", Chapter 28, pages 747-772 in Ref. [37].
53) Dennis Gannon, Rachana Ananthakrishnan, Sriram Krishnan, Madhusudhan Govindaraju, Lavanya Ramakrishnan, and Aleksander Slominski, "Grid Web services and application factories", Chapter 9, pages 251-264 in Ref. [37].
54) Concurrency and Computation: Practice and Experience http://www3.interscience.wiley.com/cgi-bin/issuetoc?ID=102522447
55) GCE research group of the Global Grid Forum, http://www.gridforum.org/7_APM/GCE.htm.
56) Open Grid Services Architecture (OGSA) http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf
57) GrADS (Grid Application Development Software Project http://www.hipersoft.rice.edu/grads/
58) AVS http://www.avs.com/
59) Khoros http://www.khoral.com/
60) BPEL4WS Business Process Execution Language for Web Services http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
61) WSCL Web Services Conversation Language http://www.w3.org/TR/wscl10/
62) ICENI project (http://www.lesc.ic.ac.uk/iceni/) of the UK e-Science Program.