Building a Computational Grid: Design Issues

Dana Petcu[1,2]

[1] Computer Science Department, Western University of Timişoara,
[2] Institute e-Austria Timişoara
B-dul Vasile Pârvan 4, RO-300233 Timişoara,
tel/fax: +40-256-244834
petcu@info.uvt.ro

Abstract. Building problem solving environments in a computational grid infrastructure is a current challenging task. We describe the architecture of CompGrid, a computational grid dedicated to solve problems described in a mathematical form. Two approaches to include existing software in CompGrid are further discussed: constructing a wrapper for a commercial software and rewriting numerical codes to make them grid-aware.

Keywords: distributed high-performance computing, computational grids.

1 Introduction

Grid computing enables the development of large scientific applications on an international scale. Grid-aware applications make use of coupled computational resources that cannot be replicated at a single site. Solving larger or new problems is possible by pooling together resources that could not be coupled easily before grids.

In this context we try to define the architecture of a computational grid, namely CompGrid, dedicated to solve problems described in mathematical terms. CompGrid will rely on a pool of mathematical software codes and on a collection of personal computers and dedicated clusters of workstations.

The next section discuss the requirements and the available technologies for computational grids. CompGrid architecture is presented in Section 3. Two different approaches are presented in details in the following sections: the grid-wrapper for a commercial software and the splitting procedure of a monolithic solver to adapt it to the grid environment.

2 Towards computational grids

The majority of the scientific and engineering applications today are monolithic [9]. A monolithic application is typically the same as a single executable program that does not rely on outside resources and cannot access or offer services to other applications in a dynamic and cooperative manner. It is typically written using just one programming language. A new style of application development based on components has become more popular nowadays. In this style the programmers build new applications by reusing existing of the shelf components and applications which may be distributed across a wide area network. Components are defined by the public interfaces that specify the functions

as well as the protocols that they may use to communicate with other components. An application program in this model becomes a dynamic network of communicating objects.

There are several reasons for programming applications on a Grid, for example: to exploit the inherent distributed nature of an application, to decrease the response time of a huge application, to allow the execution of an application which is outside the capabilities of a single (sequential or parallel) architecture, and to exploit affinities between an application component and grid resources with specific functionalities.

Grids comprise an infrastructure enabling scientists to use a diverse set of distributed software, services, and components that access a variety of dispersed resources as part of complex scientific problem-solving [17]. This infrastructure includes the use of compute resources such as personal computers, workstations, and supercomputers; access to information resources such as directory services and large-scale data bases; and access to knowledge resources such as collaboration with colleagues.

Grid applications are distinguished from traditional client-server applications by their simultaneous use of large numbers of resources, dynamic resource requirements, use of resources from multiple administrative domains, complex communication structures, and stringent performance requirements, among others [5].

Five major classes of grid applications were identified in [6]: distributed supercomputing (very large problems needing lots of CPU, memory), on demand (remote resources integrated with local computation, often for bounded amount of time), high throughput (harness many otherwise idle resources to increase aggregate throughput), data intensive (synthesis of new information from many or large data sources), or collaborative (support communication or collaborative work between multiple participants). We are interested in the first and second type of applications.

The grid is a set of additional protocols and services that build on Internet protocols and services to support the creation and use of computation- and data-enriched environments. Any resource that is on the Grid is also, by definition, on the Net [6].

Accessing advanced Grid services, such as authentication, remote access to computers, resource management, and directory services, is usually not a simple matter for problem solving environment developers. The Commodity Grid project is working to overcome this difficulty by creating Commodity Grid Toolkits (CoG Kits) that define mappings and interfaces between the Grid and particular commodity frameworks familiar to problem solving environment developers [18]. In this context Java-based Commodity Grid Toolkit (Java CoG Kit) defines and implements a set of general components mapping Grid functionality into the Java framework.

Distributed simulation can be achieved by using MPICH-G [7], an implementation of the Message Passing Interface standard that uses mechanisms provided by the Globus grid toolkit to enable wide area execution.

A portal is a community service with a single point of entry to an integrated system providing access to information, data, applications, and services. A web portal is an entry point or starting site for the WWW, combining a mixture of content and services that attempts to provide a personalized home base for its audience. A convenient way of interfacing with a computational grid is to design portals for a scientific domain or a particular

problem strategy. A grid portal is a specialized portal providing an entry point to the grid to access applications, services, information, and data available within a grid. In contrast to the web portals, grid portals may not be restricted to simple browser technologies but may use specialized plug-ins or executables to handle the data visualization requirements for example. Because of the diversified use of a computational grid portal, the architecture of such an environment must be flexible. Thus, it is not feasible to develop a point solution for a single problem. It is needed instead a portal toolkit that includes a set of services exposed via APIs that can be used to assemble a point solution for a problem.

A computational grid is dedicated to high-performance applications using widely dispersed computational resources. The goal of a computational grid as it is defined in [4] is to aggregate ensembles of shared, heterogeneous, and distributed resources (potentially controlled by separate organizations) to provide computational power to an application program.

3 CompGrid: architecture overview

The main issue in constructing a computational grid is how to aggregate the available software and hardware resources. Figure 1 suggests the vision that the user must have about the CompGrid: a pool of specialized mathematical software products working together to solve its problem.

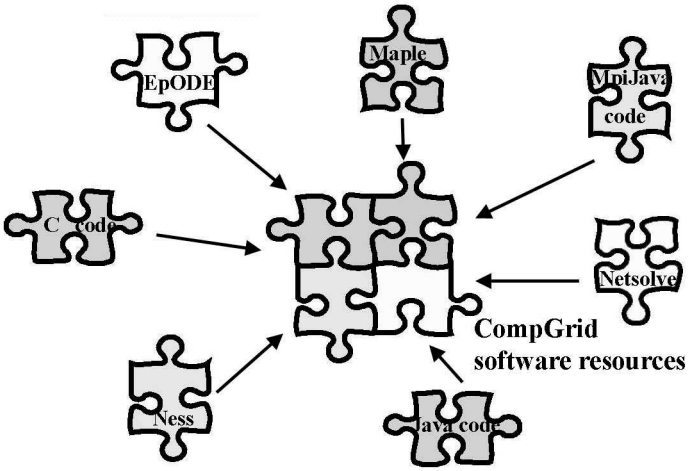

Fig. 1. User point of view: CompGrid must bring together different software resources

The CompGrid system defines a set of appropriately layered abstractions such that irrelevant complexities are hidden from higher layers, while performance-critical features are revealed.

At the highest level, the *access level*, we have the grid-aware application. For the user, the abstraction presented is a numerical or symbolic solver interface. The user controls the behaviour of the solver by specifying the problem, solution resolution etc.

Table 1. CompGrid portal components

Components	Function
Science components	Applications provided by CompGrid
Compute resources	Adaptive pool of compute resources, schedules jobs, job submission (using GRAM)
Security components	Authenticates users (using GSI)
Monitoring components	Monitors the state of CompGrid (using MDS)
Display components	Display information
Administrative components	Install software on client

The *management level* is represented by the CompGrid portal, with several components described in Table 1. This grid portal is based on the current web technologies and the Globus [8] main services:

- MDS, the information service which enables uniform access to information about the structure and state of Grid resources;
- GSI, the authentication and authorization service which provides mechanisms for establishing, identifying, and creating delegatable credentials;
- GRAM, the uniform job submission service across distributed scheduling systems.

The relations between these services and the portal components are depicted in Figure 2.

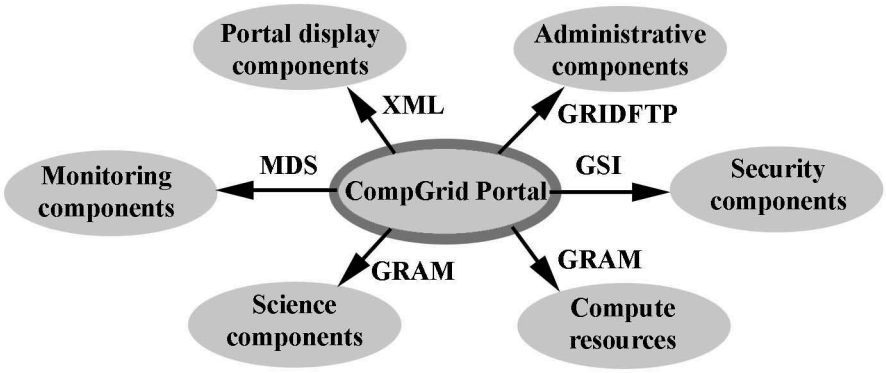

Fig. 2. ComGrid portal interface to the grid components through Java CoG

One of the main service a grid portal must provide is the job submission to remote resources. The CompGrid job management is based on the GRAM service, which follows the paths depicted by Figure 3.

From the technical point of view, CompGrid must bring more than only software resources: it must rely on a large hardware infrastructure. The physical support of the *computing level* of CompGrid must be a various collection of workstations, clusters of workstations, and parallel computers (Figure 4).

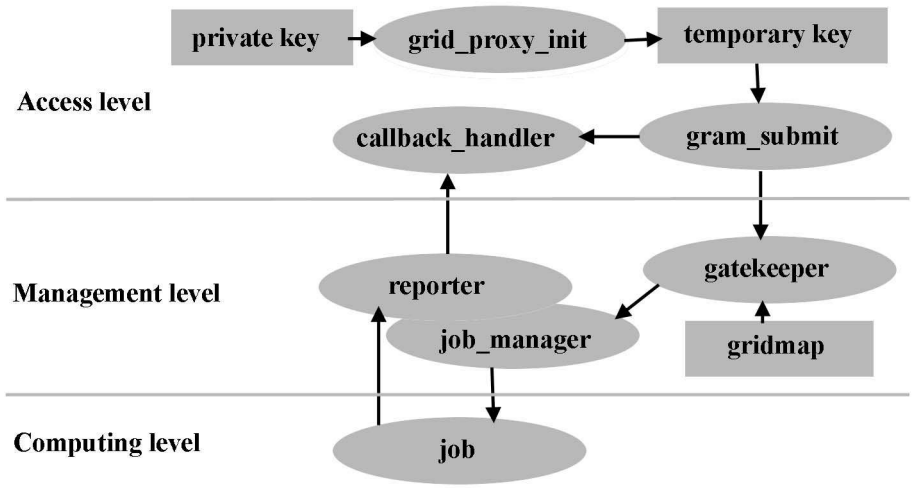

Fig. 3. GRAM - job submission

At this starting point of the CompGrid project, the main constitutive pieces are the following ones:

- hardware: a cluster of IBM-PCs connected by a fast local net (2Gbs), a departmental network, two web servers, two computers from outside country from two different security domains;
- software: free codes (like those from Netlib), previous project codes (like the two expert systems for solving nonlinear and differential equations, i.e. NESS [11] and EpODE), computer algebra systems (like Maple and Matlab);
- middleware: Globus, Java CoG, MPICH-G, mpiJava, PVM.

A first attempt to build the CompGrid portal on a small scale was reported in [14]: it is based on the IBM-PC cluster as the underlying hardware infrastructure. We envision

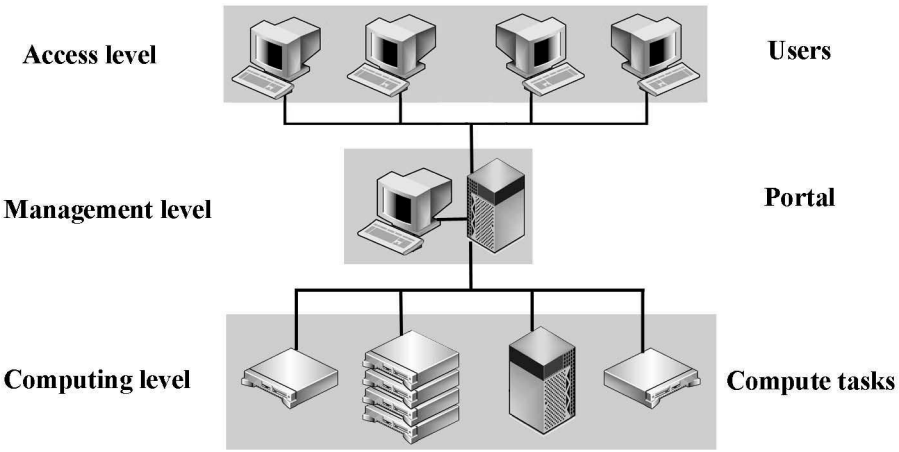

Fig. 4. Hardware and software in CompGrid

to reach with CompGrid at least the level of the ASC portal [1] facilities in which the above mentioned cluster is included as hardware resource.

In the next two sections we describe the small steps that have been already taken to bring together different software resources into CompGrid.

4 Case study: Maple2g, a grid-wrapper for Maple

One of the developments that can lead to a wider usage of grids technologies is grid-enabling of computer algebra systems (CASs). These systems are frequently used by mathematicians or engineers in performing complicated calculations and often it is desirable to be able to augment their facilities with functionality from external pieces of software.

There are several running projects aiming to link CAS with grids. NetSolve [2] is an example of a grid-based server that supports Matlab and Mathematica as native clients for grid computing; it automates the process of looking for computational resources on a network, choosing the best one available, solving a problem, and returning the answer to the user. MathGridLink [16] permits the full access to any available web service deployed on the grid, regardless of the language the service is written in, within Mathematica; with MathGridLink it becomes possible not only to integrate any existing grid web service in Mathematica's framework, but also to write and deploy new web services entirely from Mathematica, such that the Mathematica programmer does not need to have detailed knowledge about the grid. The Geodise toolkit [3] is a suite of tools for grid-enabled codes within the Matlab environment; these grid services are presented to the design engineer as Matlab functions that conform to the usual syntax of Matlab, enabling the programmer to access Java classes, to create objects, and to call methods of these objects within the Matlab environment.

To transform a component system's interface type and syntax in order for it to become part of a conglomerate system, like a computational grid, a piece of software known as a wrapper must be implemented. A wrapper [15] presents the desired interface to the operating environment and translate external interactions to and from the native interface and syntax of the component system. The construction of wrappers is probably the most difficult part of building conglomerate systems, especially when the native interface of a system is a user interface, with all the attendant ambiguities of interpretation of output. Automatic wrapper generators for legacy codes that can operate at varying degrees of granularity, and can wrap the entire code or subroutines within codes automatically, are still not available.

Maple2g is a the grid-wrapper for Maple which was recently constructed in the frame of CompGrid project. A shortly description can be found in [13]. The main reason for choosing Maple in the attempt to integrate a computer algebra system in a computational grid, is that we were not able to locate efforts to link Maple with grids, despite its robustness and ease of use. Furthermore, Maple has already a sockets library for communicating over the Internet, and a library for parsing XML. These built-in capabilities

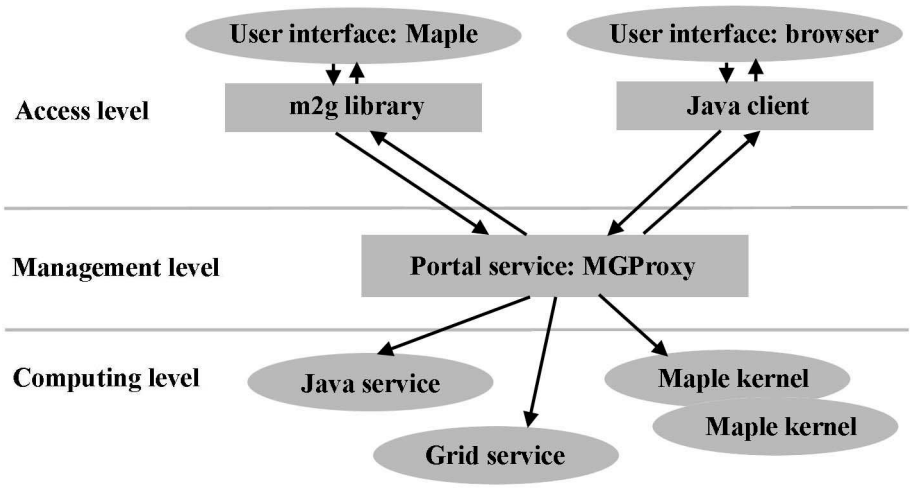

Fig. 5. Maple2g as CompGrid component

match very well with the goal of making Maple a client for an external computational service.

Figure 5 projects the Maple2g components on the CompGrid levels.

Maple2g consists of two components, *MGProxy* middleware, a package of Java classes, the interface between Maple and the grid environment, and *m2g*, a Maple library of functions allowing the Maple user to interact with the grid middleware. MGProxy has three operating modes:

1. *user mode*: activated from inside the Maple environment, receives the user command from the user's Maple interface via a socket interface, contacts the grid services (including also other MGProxy processes), queries the user requests to the contacted services, and sends to the main Maple interface results of queries.
2. *server mode*: activates a Maple twin process, acts as a server waiting for external calls, interprets the requests, sends the authentications requests to the Maple twin process, waits for Maple results, and sends them back to the user.
3. *parallel mode*: activated from user's Maple interface with several other MGProxy copies; the copy with the rank 0 enters in user mode and normally runs in the user environment, while the others enter in server mode; the communication between different MGProxy copies is done via a standard message passing interface (mpiJava [10]).

A short example of using the m2g library (user mode) is given in what follows:

```
> with(m2g);
  [m2g_connect, m2g_getservice, m2g_jobstop, m2g_jobsubmit, m2g_maple,
   m2g_MGProxy_end, m2g_MGProxy_start, m2g_rank, m2g_recv, m2g_results,
   m2g_send,m2g_size]
> m2g_MGProxy_start(); m2g_connect();
  Grid connection established
> m2g_getservice("factor",‘service_location‘);
```

```
    ["&(resourceManagerContact="myri1.info.uvt.ro")(count=1) (label=
     "subjob 0")(directory=/home/Dana)(executable=/home/Dana/factor)"]
  > m2g_jobsubmit(1,cat(service_location[1],"(args=903957729051191)")
      job submitted
  > m2g_results(1);
      29*71*113*173*229*281*349
  > m2g_MGProxy_end();
    Grid connection closed
```

The grid service named `factor` is appealed from inside Maple in order to decompose a large number. Vice versa, a thin web interface can activate a Java client which communicate with MGProxy (server mode) appealing to the internal Maple function `ifactor` in order to obtain the decomposition of the same long number. If a large list of integers waiting to be factorized is given several Maple kernels can be activated on different processors of a cluster of workstations available as CompGrid resources.

5 Case study: grid-enabled version of a numerical solver

EpODE, expert system for the numerical solution of initial value problems for systems of ordinary differential equations [12], was designed in a monolithic version. It detects the problem properties and can automatically select an appropriate numerical solver. A friendly user interface allows the extension of the solver database and a solver properties detector informs the expert about the new solving alternatives. Moreover, several solver can be launch in cluster computing environments.

Several ideas from EpODE can be extended towards CompGrid. The management level of CompGrid must include a detector of user problem properties, a mechanism to bring together the problem and the possible solvers, an automated expert or a user controlled scheme to select from many solving variants, an variable database of solvers.

To build such an basic environment for CompGrid we start by reconstructing EpODE in a grid-enabled version. EpODE 1.0 can be wrapped to be seen by CompGrid user as grid software resource (a wrapper is needed). On other hand, in order to construct a grid-aware version of EpODE, we must add to it at least the following facilities:

1. multiple threads for different external solvers;
2. search for hardware resources and code transfer;
3. links to proprietary and free codes;

Since different components of EpODE can be used not only for solving initial value problems, the codes are rewritten now in Java instead C++, allowing an easier use of Java CoG and Globus toolkit. Different components of EpODE are projected on different levels of CompGrid architecture, as Figure 6 illustrates.

The parallel ODE solvers from EpODE 1.0 have been designed for dedicated homogeneous cluster environments. In order to use heterogeneous grid resources a dynamic load balancing scheme must be added. Moreover, the older message passing interface PVM must be replaced with the MPICH-G which works both on cluster and grid environments.

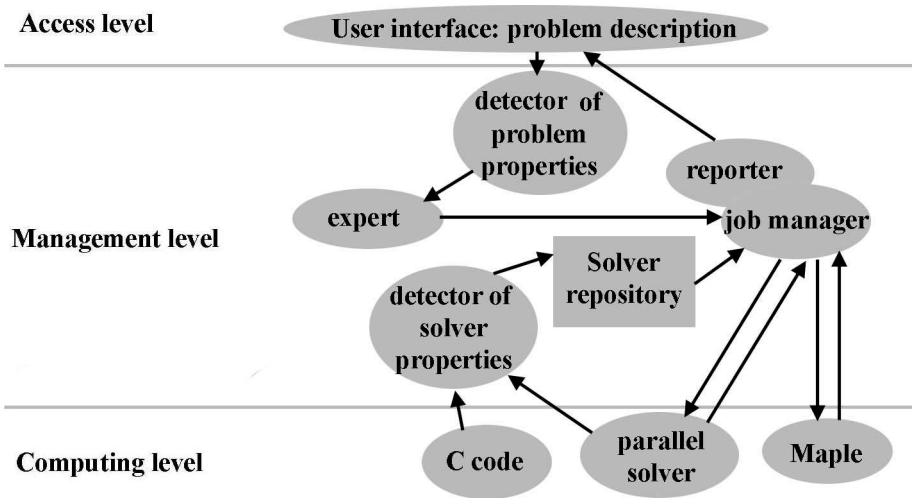

Fig. 6. EpODE parts projected into CompGrid components

The future steps will be:

- build a web interface for mathematical problem description which can provide elementary checks;
- build detectors for problem properties for linear or nonlinear equation, partial derivative equations, integral equations etc;
- populate the code repository with numerical or symbolic solvers;
- build automated experts for each type of mathematical problem;
- build or adapt the solvers to be used in cluster environments;
- build or adapt parallel solvers to be used in grid environments.

6 Conclusions

CompGrid intends to be an environment for assisted solving of mathematical problems. Small steps have been already taken to construct a such environment: prototype of a grid portal, a grid-wrapper for a commercial code, and coding grid-aware numerical solvers. These dispersed efforts are the key components of the CompGrid. The future steps to be undertaken were also defined.

References

1. Astrophysics Simulation Collaboratory (ASC) Portal http://www.ascportal.org/
2. Casanova H. and Dongarra J.: NetSolve: a network server for solving computational science problems. Inter.J. Supercomputer Appls. & HPC 11, no. 3 (1997), pp. 212–223, http://icl.cs.utk.edu/netsolve/
3. Eres M. H., Pound G.E., Jiao Z., Wason J.L., Xu F., Keane A.J., and Cox J.S., Implementation of a grid-enabled problem solving environment in Matlab. In Procs. WCPSE03 (2003), in print, http://iccs03.cs.uwa.edu.au/program/WorkshopsICCS2003Page2.html, http://www.geodise.org
4. Foster I., Kesselman C., The Grid. Blueprint for a new computing infrastructure (1999), Morgan-Kaufmann.
5. Foster I., Kesselman C. Tsudik G., Tuecke S., A Security Architecture for Computational Grids. In Proc. of the 5th ACM Conference on Computer and Communication Security (1998), pp. 83-91.

6. Foster I., Kesselman C., Tuecke S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15, no. 3 (2001), pp. 200-222, http://www.globus.org/research/papers/anatomy.pdf.
7. Foster I., Geisler J., Gropp W., Karonis N., Lusk E., Thriruvathukal G., Tuecke S., A Wide-area Implementation of the Message Passing Interface. Parallel Computing 24, no. 12 (1998), pp. 1735-1749.
8. Globus Toolkit 3.0, http://www.globus.org.
9. Laforenza D., Grid programming: some indications where we are headed, Parallel Computing 28 (2002), pp. 1733-1752.
10. mpiJava, http://www.npac.syr.edu/projects/pcrc/HPJava/mpiJava.html
11. Negru V., Sandru C., Rotaru M., A multi-agent system for scientific problem solving, In Procs. of the 1st International Workshop of Central and Eastern Europe on Multi-Agent Systems, St. Peterburg (1999), pp. 172-180.
12. Petcu D., Drăgan M., Designing an ODE solving environment, LNCSE 10: Advances in Software Tools for Scientific Computing, eds. H.P. Langtangen, A.M. Bruaset, E. Quak, Springer-Verlag, Berlin (2000), pp. 319-338.
13. Petcu D., Dubu D., Paprzycki, M., Towards a Grid-aware Computer Algebra System. In Procs. ICCS 2004, LNCS 3036, in print.
14. Petcu D., Oprean A., Constructing a Grid Portal, In Procs. 5th International Workshop on Symbolic and Numerical Algorithms for Symbolic Computing (2003), Romania, pp. 324-328.
15. Solomon A.: Distributed computing for conglomerate mathematical systems, In Integration of Algebra and Geometry Software Systems, eds. M. Joswig, N. Takayama, http://www.illywhacker.net/papers/ webarch.ps
16. Tepeneu D. and Ida T., MathGridLink - A bridge between Mathematica and the Grid. In Procs. JSSST03 (2003), in print.
17. von Laszewski G., Blau E., Bletzinger M., Gawor J., Lane P., Martin S., Russell M., Software, Component, and Service Deployment in Computational Grids, In Procs. First International IFIP/ACM Working Conference on Component Deployment, Berlin (2002), http://www.mcs.anl.gov/~ laszewsk/papers/deploy-32.pdf.
18. von Laszewski G., Foster I., Gawor J., Lane P., Rehn N., Russell M., Designing Grid-based Problem Solving Environments and Portals, In Procs. 34th Hawaiian International Conference on System Science, Hawaii (2001), http://www.mcs.anl.gov/~ laszewsk/papers/cog-pse-final.pdf.