Grid interactive service-oriented programming environment

R.M. Kolonay
Air Force Research Laboratory, WPAFB OH

M. Sobolewski
Texas Tech University, Lubbock TX

ABSTRACT: Improvements in distributed computing, and the technologies that enable them, have led to significant improvements in middleware functionality and quality, mainly through networking and protocols. However, the distributed programming style has changed little over the years. Most programs are still written line per line of code in languages like C, C++, and Java. These conventional programs that can provide grid operations and grid data can be considered as common grid resources and shared by research and education communities worldwide. However, there are no relevant programming methodologies to utilize efficiently these shared service providers as a potentially vast grid repository, except through the manual writing of code. Realization of the potential of grid computing requires significant improvements in grid programming methodologies. The Grid Interactive Service-Oriented (GISO) methodology is presented that provides a programming environment with development tools that permit interactive (point-and-click), true grid programming, thus permitting the different elements of programming to be stored, reused, aggregated, and executed with a level of concurrency and grid-level control strategy not achievable in the conventional programming languages.

1 INTRODUCTION

From the beginning of networked computing, the desire has existed to develop protocols and methods that facilitate the ability of people and automatic processes across different computers to share information and knowledge across heterogeneous systems. As ARPANET (Postel and Sunshine 1981) began through the involvement of the NSF (Postel & Reynolds 1987, Lynch & Rose 1992) to evolve into the Internet for general use, the steady stream of ideas became a flood of techniques to submit, control, and schedule jobs across distributed systems (Lee 1992). The latest in these ideas is the *grid* (Foster 2002, Kesselman et al. 2002, Tuecke et al. 2002, Foster et al. 2001), to be used by a wide variety of different users in a non-hierarchical manner to provide access to powerful aggregates of resources (Foster & Kesselman 1999), Grimshaw, & Wulf 1997). Grids, in the ideal, are intended to be accessed for computation, data storage and distribution, and visualization and display, among other applications without consideration for the specific nature of the hardware and underlying operating systems on the resources on which these jobs are carried out (Smarr 1997, NRC 1993).

The reality at present, however, is that grid resources are still very difficult for most users to access, and that detailed programming must be carried out by the user through command line and script execution to carefully tailor jobs on each end to the resources on which they will run, or for the data structure that they will access. This produces frustration on the part of the user, delays in adoption of grid techniques, and a multiplicity of specialized "grid-aware" tools that are not, in fact, aware of each other that defeat the basic purpose of the grid.

The need for further improvements in grid computing is clear, and requires significant further improvements in grid programming technology. By inspection of the above paradigm, it is clear that incremental improvements in the scripts and submission techniques will not suffice. A new *grid interactive service-oriented (GISO)* integrated development environment (IDE) that is based on evolution of the concepts and lessons learned in the FIPER project (Sobolewski 2002, Lapinski & Sobolewski 2002), Röhl et al. 2000), a $21.5 million program founded by the United States National Institute of Standards and Technology (NIST), is presented. It provides an environment that will permit true interactive click-and-drag grid programming through the manipulation of graphical elements that represent object-oriented grid resources, thus permitting the different elements of grid program to store, reuse, aggregate, and execute with a level of concur-

rency and grid-level control strategy not achievable in the conventional programming languages.

The presented GISO programming approach is characterized as follows:

1. Service-oriented grid programming is achieved by applying the object-oriented concepts directly to the grid as a repository of network objects (method and context providers)
2. Service-oriented execution infrastructure enabling dynamic federations of grid providers to execute service-oriented programs
3. Provisioning and deploying grid objects with an autonomic behavior, enabling grid objects to be instantiated and managed on compute resources available through the grid using an adaptive quality of service model
4. An open, web-based environment in which existing proprietary applications and analytical packages are integrated through Java-based wrappers that handle grid processes and data distributed across different locations.

2 GISO CONCEPTUAL FRAMEWORK

Building on the object-oriented paradigm the *service-oriented* paradigm, in which the objects are distributed, or more precisely they are *network objects* and play some predefined roles. A *service provider* is an object that accepts messages from *service requestors* to execute an item of work – a *task*. The task object is a service request – a kind of elementary grid instruction executed by a service provider. A service *jobber* is a specialized service provider that executes a *job* – a compound request in terms of tasks and other jobs. The job object is a *service-oriented program* that is dynamically bound to all relevant and currently available service providers on the grid. This collection of grid providers dynamically identified by a jobber is called a *job federation*. This federation is also called a *job space*. While this sounds similar to the object-oriented paradigm, it really isn't. In the object-oriented paradigm the object space is a program itself; here the job space is the *execution environment* for the job itself and the job is a service-oriented program. This changes the game completely. In the former case the object space is a *virtual computer*, but in the latter case the job space is the *virtual network*. This virtual network or *grid federation* is the jobs' execution environment and the *job object* is a service-oriented program. In other words, we apply the object-oriented concepts directly to the grid in the service-oriented manner.

The GISO framework is built on the top of the FIPER Technology (Kolonay et al. 2002) middleware. The GISO environment provides the means to create interactive service-oriented programs and execute them without writing a line of source code. Jobs and tasks are created using web-based user interfaces. Also via web-based interfaces the user can execute and monitor the execution of jobs or tasks. The jobs and tasks are persisted for later reuse. This feature allows the user quickly to create new applications or programs on the fly in terms of existing tasks and jobs.

In all, GISO development tools provide (see Figure 1) accessibility through web-centric architecture; self-manageability using federated grids, scalability via network centricity, and adaptability with the power of mobile code inserted for execution through service providers.

Figure 1. GISO layered architecture

In this paper the focus is on the GISO programming and developed tools identified in Figure 1.

3 GISO PROGRAMMING AND DEVELOPMENT TOOLS

The peer-to-peer (P2P) service-oriented framework presented targets multiparty grid transactions. A collection of all registered service providers (active and inactive) is called a service grid. A nested transaction is composed of a federation of providers that come together for completing a transaction. A transaction consists of a set of tasks with specific precedence relationships. When performing a nested transaction, be it either a banking transaction or an engineering analysis, there are three basic components that can be identified. These are; the *process* or series of steps that must be executed to complete the transaction, a specification of the *action* to be taken in each step of the process, and the *information/data* associated with each step in the process (both input and output). Within FIPER the program objects that represent the components of a nested transaction are FiperExertions (FiperJob and FiperTask), FiperMethod, and FiperContext. The basic work unit within the FIPER programming environment is an exertion. Each exertion contains a FiperMethod and a FiperContext object. The Fiper-

Method specifies what *action* that is to be taken in a given step in the process. The FiperContext contains all the *data* the FiperMetod operates on or generates. The FiperContext also holds attributes for the data much like MIME types that identify the application(s) the data
is associated with, its format (text, binary etc.), and other user defined modifiers. A FiperJob defines the *process*. It consists of one or more exertions, the execution strategy for the process (sequential, parallel, looping and conditionals), and the mapping/relationship of data between exertions. The hierarchy of these classes is shown in Figure 2. It is worth nothing that recursion of FiperJobs is supported. That is any of the FiperTasks within a Fiper Job can be a FiperJob itself.

The relationship between the FIPER program objects and the general description of a nested transaction is as follows; a FiperJob represents the process, the *FiperMethod represents the action*, and a FiperContext represents the data/information. The FiperTask acts as a container holding the FiperMethod and FiperContext creating the basic unit of work that is passed between various service providers.

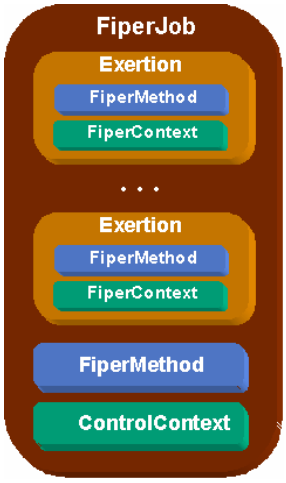


Figure 2.
Program Object Hierarchy

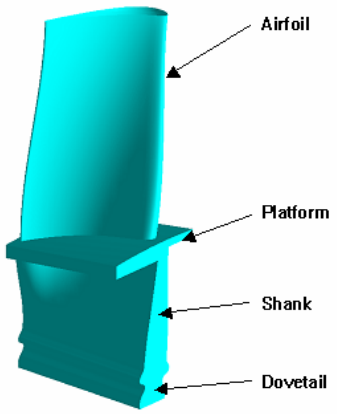
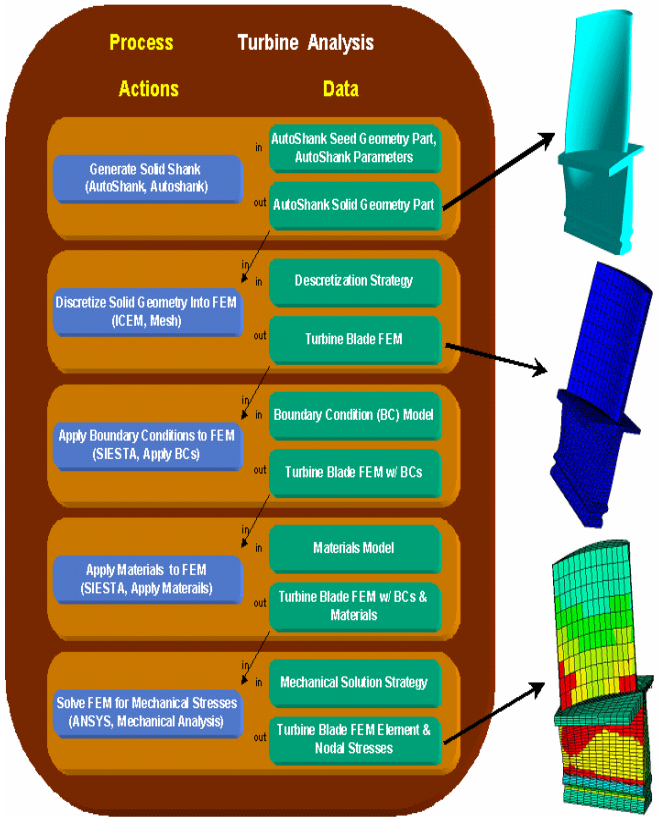


Figure 3.
Turbine Blade Geometry

As an example of a nested transaction in the FIPER Environment consider the following engineering application, the mechanical analysis of a gas turbine component. The component, a turbine blade is shown in Figure 3. The *process* of performing a mechanical analysis consists of the following *actions*; generate solid geometry, descretize the geometry into a finite element model (FEM), apply boundary conditions to FEM, apply materials to FEM, and solve the FEM for structural stresses. The necessary input data for each action and the resulting output data are shown in Figure 4. Also depicted in Figure 4 is the association between the three components of a nested transaction and the FIPER program objects.

To create the necessary program objects (FiperContext, FiperMethod, FiperTask, and FiperJob) for a nested transaction in the FIPER environment a collection of web browser user agents has been developed. It is not necessary to use these user agents for the development and execution of a FiperJob. Any standalone application can perform programmatically the same steps to create the necessary ob-



jects and act as a service requestor to submit the FiperJob for execution. The following sections illustrate the usage of the web user agents to create and execute the necessary FIPER program objects to perform the mechanical analysis of the turbine blade. Figure 5 shows the Fiper launcher page once logged into the Fiper environment. Here it can be

Figure 4. Process for the Mechanical Analysis of a Turbine Blade

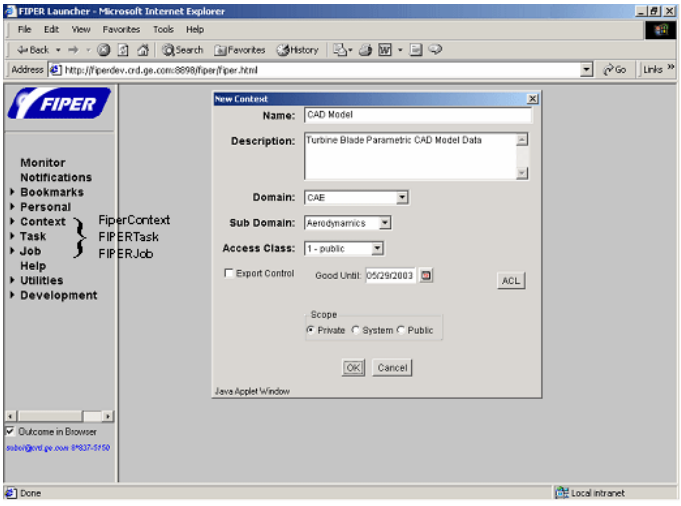


Figure 5. FIPER Launcher and New Context Dialogue

seen that there are separate selections for the above described program objects, FiperContext, FiperTask, and FiperJob. The FiperMethod object is created within the FiperTask menu selection.

3.1 *Context Editor*

The Context Editor allows the end-user to specify the data or references to the data along with attributes associated with the data. When creating a new context the end-user is presented with the dialog that requires the following fields. The Name and Description fields are user defined, the Domain and Subdomain are selected from a drop down menu. The Access field is a company internal access classification and the Export Control box indicates if the data is export controlled. The ACL button produces an Access Control List (ACL) dialogue that allows the end-user to assign read, write, and execute permissions on this program object based on userid or role. Once the end-user completes the New Context Dialogue and selects OK the Context Editor then appears. Figure 6 shows the Context Editor along with the context for the first action or task in the Turbine Mechanical Analysis Job represented in

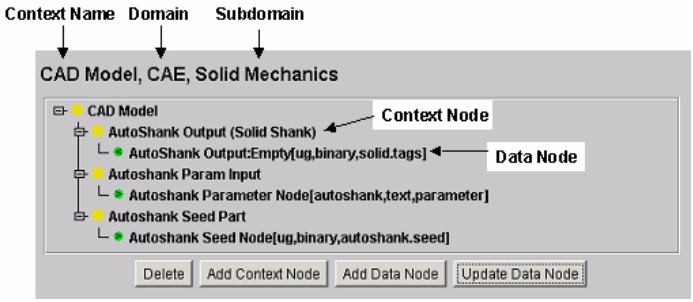


Figure 4.

Figure 6 also illustrates that the FiperContext is a tree structure with Context Nodes and Data Nodes. The Data Nodes are further identified as either input ">" or output "<". The editor allows the end-user the ability to create, edit, or delete Context Nodes and Data Nodes in the FiperContext.

3.2 *FiperTask Editor*

From the Fiper launcher in Figure 5 the end-user se-

Figure 6. FiperContext Editor

lects Task, New, and completes the New Task Dialog to gain access to the Task Editor shown .

Recalling that the FiperTask is the fundamental building block or work unit in the FIPER Environment which contains the *action* and *data* for a nested transaction (reference Figure 4), the Methods field represents the *action* and the Context field represents the *data*. To view/edit more detail on theses fields the end user selects "Update Content" which produces an editor (see Figure 7). Figure 7 shows the definition of the FiperMethod and the Context that is used for the selected task, Generate Solid Shank. The fields Interface, Command, Provider, and Method Type define the Method. The Interface and the Provider are used as the attributes to locate a service within the environment with the current implementation. The context for this task is the CAD Model Context presented in Figure 6. Once all the actions/FiperTasks have been defined for a given process/FiperJob the FiperJob itself can then be constructed.

3.3 *FiperJob Editor*

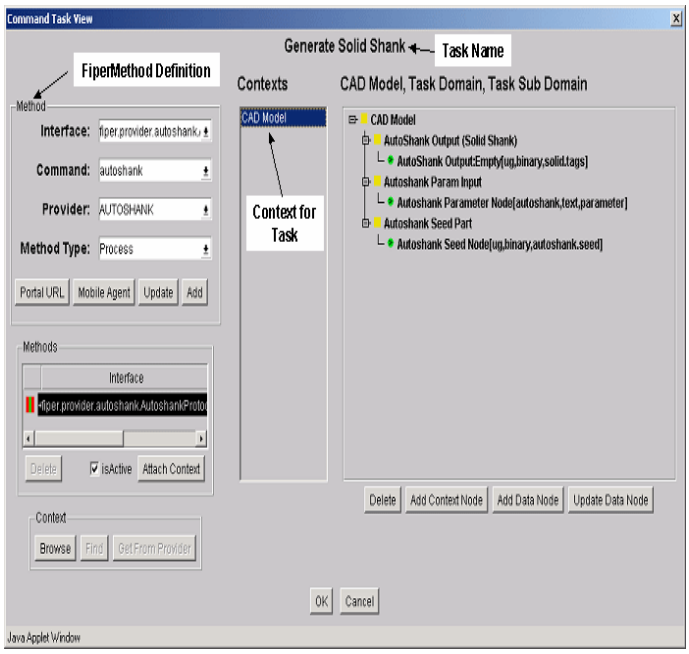
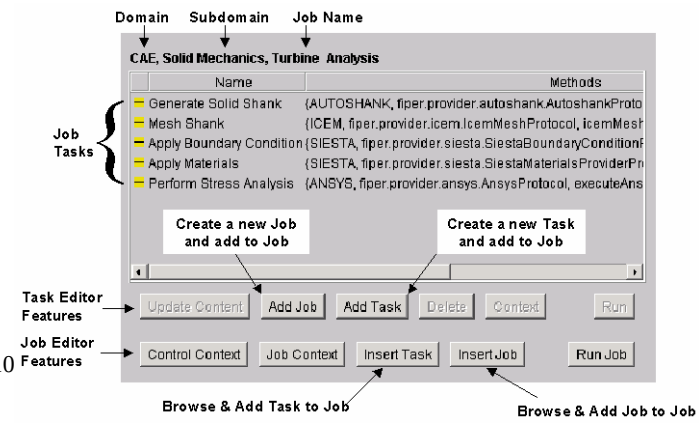


Figure 7. FiperTask, FiperMethod and FiperContext Editor

Figure 8 illustrates the creation of the FiperJob represented in Figure 4. It contains all the tasks, Generate Solid Shank, Mesh Shank, Apply Boundary Conditions, Apply Materials, and Perform Stress Analysis.

Figure 8. FiperJob Editor

The Job Editor lists all FiperTasks associated with the job along with the FiperTask's Name and FiperMethod Attribute information (Provider Name and requested provider's type - interface). The Task and Job Editor features allow the end user to add additional FiperTasks or FiperJobs by either browsing existing program objects or creating new



objects on the fly. The Job Editor features also enable the specification of the Control Context and the JobContext. The ControlContext specifies the flow and method of execution of the FiperJob. The final step before a FiperJob can be executed is to define the flow of data between tasks in the job. This is done using the JobContext dialog, which can be invoked from the Job Editor features on the Job Editor Dialog in Figure 8.

The FiperJob Context dialog for the Turbine Analysis Job is shown in Figure 9. Here the Job is shown with each task and the context for each task in a hierarchical tree structure. The data flow from one task to the other is defined by dragging one Fiper DataNode onto another Fiper DataNode. In Figure 16 this has occurred by dragging the AutoShank Output Solid Shank Node contained in Task0 onto the Solid Shank unnamed Fiper DataNode in Task1.

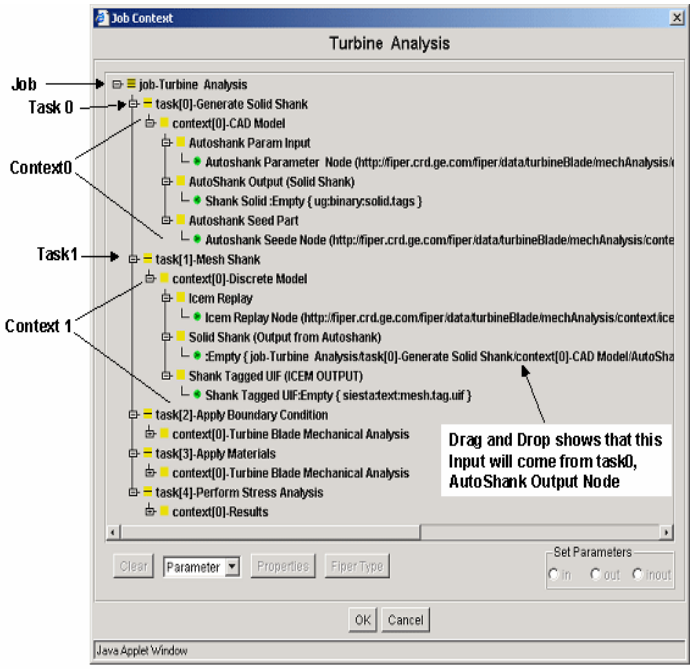


Figure 9. Fiper JobContext Dialog

Once the data flow has been defined in the JobContext the FiperJob is now ready for execution. To submit the job to the Fiper Environment the Run Job button is selected in the Job Editor (Figure 8). A typical engineering analysis or design job could take anywhere from a few hours up to several days or even weeks. With jobs running this long it is critical that the end-user have access to the status of the job and control over the job as it executes. This is the function of the Job Monitor.

3.4 *FiperJob Monitor*

The most critical capability that GISO programming will need from an end-users perspective is the ability to interact with the process/FiperJob once it has been submitted to the environment. Using a GISO IDE will require a cultural change within the end-user community. Today's state of practice is that typical designers and analysts execute single standalone applications either on their desktop or submit the runs to a major shared resource (MSR) computing environment. In either case the end-user is executing applications individually and if a failure occurs they know at least which application the failure occurred within. Also, when running locally or in a MSR the end-user usually has some or all control over the running application and can closely monitor the progress of the execution by monitoring log files and or output files from the application. In the GISO IDE the end-user is now combining many application to perform a nested transaction and submitting the execution of the nested transaction to the network, which could easily take days or weeks to complete. In the GISO IDE the end-user may have no idea where the execution is taking place and worse will have no feedback as to the state of progress of the process. In the GISO IDE the end-user surrenders all control to the environment, a precarious proposition for a designer who is accustomed to having complete control of the applications they are running. With these facts in mind a few essential functionalities are identified for GISO programming that are necessary for end-user to accept such a working environment. The end-user must be able to monitor the progress of the process and obtain intermediate results from a given task. The end-user must be able to control the process once it is submitted to the environment by stopping, suspending, or terminating the process. For a suspended GISO program the end-user must be able to edit not only the data within the process but also the process itself by adding or deleting tasks. After any edits to the data or process the end-user must be able to resume the process from any task within the process not necessarily the task the process was suspended at. If the process fails the end-user must obtain meaningful information that specifies where the failure occurred and what action needs to be taken to correct the problem. This last requirement puts a significant burden on the service provider developers to properly trap exceptions and translate them into meaningful information for the end-user.

In the FIPER Environment the monitoring/client process interaction is done using the Job Monitor. Figure 10 shows the Turbine Analysis Job running in the Job Monitor. The Job Monitor can be viewed as an "interactive debugger for program objects or services on the network". The Job Monitor shows the progress of the process (green complete, green/yellow running, red failed, yellow suspended). It also displays intermediate information from a task (by viewing the job context) if the provider returns such information. The client is also able to stop,

suspend, step and resume a running job. In addition, for a given suspended or completed job, the client has access to a drop down menu that allows full edit capability of the data in the job or the job/process itself. Data can be changed, tasks can be edited/added/deleted and the job resumed from any task.

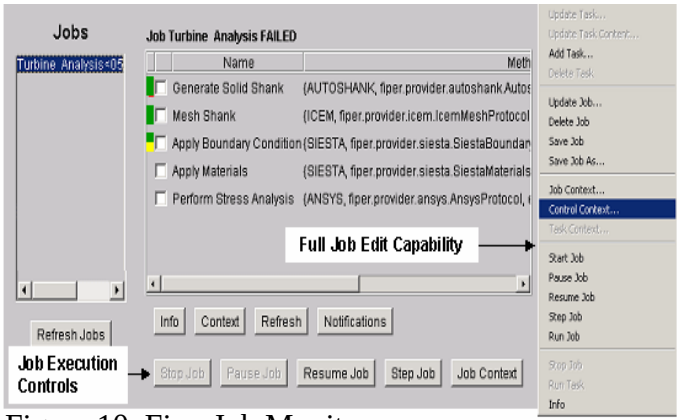


Figure 10. FiperJob Monitor

4 CONCLUDING REMARKS

In the GISO approach object-oriented concepts are applied to the network and grid-oriented programs. A job is a service-oriented program executed in a federated service-oriented environment across multiple virtual organizations. Jobs are created using friendly, interactive web-based graphical interfaces. Jini connection technology from Sun Microsystems enables federated, platform independent, real world grids. It allows us to create GISO programs that process a whole aircraft engine as a virtual object-oriented product control structure that can be manipulated by multidisciplinary teams as network-centric, active, evolving product. New shared programs and engineering applications can be assembled as needed on the fly by integrating new capabilities into existing workflows, systems, devices and applications. The presented web-centric GISO IDE reduces the costs of solving business problems as well as of establishing and maintaining online business relationships. Services are provided by shared low cost, easy to develop service providers and are integrated into the core business of an enterprise. An experimental version of presented approach was successfully deployed at GE Aircraft Engines.

REFERENCES

Foster, I. & Kesselman, C. eds.1999. "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann Publishers, San Francisco CA.

Foster, I., C. Kesselman, C., Tuecke, S. 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations.. International J. Supercomputer Applications, 15(3), 2001. Defines Grid computing and the associated research field, proposes a Grid architecture, and discusses the relationships between Grid technologies and other contemporary technologies.

Foster, I. & Kesseman, C. 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002. (extended version of Grid Services for Distributed System Integration).

Grimshaw, A. S. & Wulf W. A. 1997. "The Legion vision of a worldwide virtual computer", Communications of the ACM, 40(1), 39-45.

Hafner, K. & Lyon, M. 1996. "Where Wizards Stay Up Late," (a history of Internet development), Simon and Schuster, New York.

Kolonay, R.M., Sobolewski M., Tappeta, R., Paradis, M., Burton, S. 2002. *Network-Centric MAO Environment*, The Society for Modeling and Simulation International, 2002 Westrn Multiconference, San Antonio, Texas, Jan 27-31.

Lapinski M. & Sobolewski, M. 2002. "Managing Notifications in a Federated S2S Environment," International Journal of Concurrent Engineering: Research & Applications, December.

Lee, J. ed. 1992 "Time-Sharing and Interactive Computing at MIT," IEEE Annals of the History of Computing 14:1

Lynch, D.L. & Rose, M.T. 1992. "Internet System handbook," Addison-Wesley, reading, MA.

National Research Council 1993. Reports relevant to early grid research include the following: "National Collaboratories: Applying Information Technology for Scientific Research," National Academy Press, Washington D.C.

Postel, J., Sunshine, C. & Cohen, D. 1981. "The ARPA Internet Protocol," Computer Networks 5:261-271.

Postel, J. & Reynolds, J. 1987. "Request for Comments Reference Guide (RFC1000)," Internet Engineering Task Force.

Smarr L.1997. "Computational infrastructure: Toward the 21st century," Special issue on plans for a National Technology Grid, Communications of the ACM 40, 11

Sobolewski, M. 2002. FIPER: The Federated S2S Environment, JavaOne, Sun's 2002 Worldwide Java Developer Conference, San Francisco, 2002.

Röhl P. J., Kolonay, R.M., Irani, R.K., Sobolewski, M., Kao, K. 200. "A Federated Intelligent Product Environment, AIAA-2000-4902, 8th AIAA /USAF/ NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, CA, September 6-8.

Tuecke S., Czajkowski, Foster, I., Frey, J., Graham, S., Kesselman, C. 2002. Grid Service Specification. Open Grid Service Infrastructure WG, Global Grid Forum, Draft 2.