Federated Grid Computing with Interactive Service-oriented Programing

Michael Sobolewski and Raymond M. Kolonay

Concurrent Engineering 2006; 14; 55
DOI: 10.1177/1063293X06064148

CONCURRENT ENGINEERING: Research and Applications

Federated Grid Computing with Interactive Service-oriented Programing

Michael Sobolewski[1,*] and Raymond M. Kolonay[2]

[1]*Computer Science Department, Box 43104, Texas Tech University, Lubbock, TX 79409-3104, USA*
[2]*AFRL/VASD, 2130 8th Street, Suite 1, Building 146 Room 220, Wright-Patterson AFB, OH 45433-7542, USA*

Abstract: Improvements in distributed computing, and the technologies that enable them, have led to significant advancements in middleware functionality and quality, mainly through networking and protocols. However, the distributed programing style has changed little over the years. Most programs are still written line per line of code in languages such as 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 programing 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 programing methodologies. The Grid interactive service-oriented (GISO) methodology presented provides a programming environment with development tools that permit true interactive grid programming. The GISO approach permits the different elements of programming to be stored, reused, aggregated, and executed with concurrency and a grid-level control strategy not achievable in the conventional programming languages.

Key Words: concurrent engineering, engineering analysis, grid programing, service-oriented computing, object-oriented computing, service orchestration.

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 the information and knowledge across heterogeneous systems. As ARPANET [1,2] began through the involvement of the NSF [3,4] 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 the distributed systems [5]. The latest in these ideas is the grid [6–8] to be used by a wide variety of different users in a non-hierarchical manner to provide access to powerful aggregates of resources [9,10]. Grids, in the ideal, are intended to be accessed for computation, data storage and distribution, 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 [11,12].

The reality at present, however, is that grid resources are still very difficult to access for most of the users, and that detailed programing 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 programing technology. By inspection of the afore mentioned 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 [13–21], a $21.5 million program funded 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 programing 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 concurrency and grid-level control strategy not achievable in the conventional programing languages.

*Author to whom correspondence should be addressed.
E-mail: sobol@cs.ttu.edu
Figures 1–13 appear in color online: http://cer.sagepub.com

1063-293X/06/01 0055–12 $10.00/0 DOI: 10.1177/1063293X06064148

The presented GISO programing approach [15] is characterized as follows:

> Service-oriented grid programing is achieved by applying the object-oriented concepts directly to the grid as a repository of network objects (method and context providers).
> Service-oriented execution infrastructure enabling dynamic federations of grid providers to execute service-oriented programs.
> Provisioning and deploying grid objects with an autonomic behavior, enabling grid objects to be instantiated and managed on computing resources available through the grid using an adaptive quality of service model.
> 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.

The presented approach addresses a number of gaps which exist in the grid technology. The technology gaps and approach to solve these gaps are articulated in Table 1.

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, is adopted in this work. 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 as a job federation. This federation is also called as a job space. While this sounds similar to the object-oriented paradigm, it is not. 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 is a paradigm shift. 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, one can 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 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 [21]. The jobs and tasks are persisted for later reuse. This feature allows the user to quickly create new applications or programs on the fly in terms of existing tasks and jobs.

The GISO/FIPER supports three centricities and deploys three neutralities. GISO's three centricities are network centricity, service centricity, and web centricity. A GISO federation is composed of various service providers any of these can come and go, and the system can respond to changes in its environment in a reliable way (network centricity). Services in GISO can discover lookup services and join the grid or lookup for relevant services in order to cooperate in a grid federation (service centricity). Users can request to use multiple services and check the status of their submissions in different locations through an HTTP portal with thin web clients (web centricity). The three neutralities that the GISO deploys are location neutrality, protocol neutrality, and implementation neutrality. With location neutrality, services need not be collocated; lookup services are discovered and used to find a particular service, which simplifies management of the entire grid environment. With protocol neutrality, the way in which clients communicate with a service provider is not important. Clients are not aware of what protocols

Table 1. Technology gaps vs the GISO approach.

Technology gap	GISO Solution
1. Protocol-based grid middleware is difficult to use	Develop object-oriented middleware components
2. Current grid middleware is transaction, data, and host centric	Provide autonomic, dynamic, QoS, network centric middleware components
3. No grid-oriented programing methodologies to utilize grid resources and middleware services	Provide point-and-click interactive grid programing
4. Moving executable code and data over grid to compute resources is inefficient	Provide reusable method and data services as service-to-service grid providers
5. Access to grid resources is not user friendly	GISO easy-to-use web-based end-user-agents
6. No grid high-level programing and development tools	Develop interactive grid-programing and development tools

are used or where the implementations reside. With implementation neutrality, the clients who use the GISO services do not need to know what languages are used or how a service is implemented.

In all, GISO development tools provide (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.

3. GISO Execution Environment

The peer-to-peer (P2P) service-oriented framework presented here targets the 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. The service providers do not have mutual associations prior to the transaction. They come together (federate) for a specific transaction. Each provider in the federation performs its services according to a job's control strategy that defines a transaction. Once the transaction is complete the federation dissolves and the providers disperse and seek other transactions to join. Different combinations of providers may come together for any given type of transaction at different times. The following characteristics define such transactions that are supported by the GISO execution environment:

1. Multiple tasks/jobs need to be executed in order to complete the transaction;
2. Service providers are interchangeable (i.e., any provider that implements the same interface for a service can be selected to perform the service);
3. Same service provider can perform multiple tasks in the transaction;
4. Tasks in a transaction (federation) need to share data and resources with each other; and
5. A nested transaction is coordinated by a service broker – jobber – with a set of depended sub-jobs of the top-level job.

The GISO/FIPER middleware is service-based in which a service is defined as an independent self-sustaining entity performing a specific task or job. Each service is defined by a public interface. The service grid is dynamic in which new services can enter the grid and existing services can leave the network at any instance. Services advertise themselves and can be found and selected based on the type (interface) and other attributes that they exhibit.

The GISO environment defines all decentralized distributed components in the system to be equal. These components might be devices, repositories, processes, or objects on the network. In this environment, peers are network objects of the same type. Each peer implements well defined, top-level public and common interfaces and may implement multiple custom interfaces that are published when the peer joins the environment. All methods of the custom interfaces have the same format: an input and an output parameter of any provider method is just a *service context* – the generic data structure for GISO programing [22].

By its interface (type) and optional attributes (e.g., provider name), the network object can be dynamically found on the network without a host name and port required. The custom interfaces and their implementations might change, as they are specific to particular service providers. Thus peers should not expose their specific interface explicitly at the infrastructure level. The common, top-level peer interface is called *servicer*. All peers implement this interface and their equality is defined as being service providers or *servicers*. A service is an act of requesting a service (Exertion) operation from a service provider as explained in Figure 2. The *exertion* describes a distributed activity in object-oriented terms.

In the GISO environment two types of basic exertions are defined: tasks and jobs. A task is the atomic exertion that is defined by its context model (data), and by its

Figure 1. GISO layered architecture.

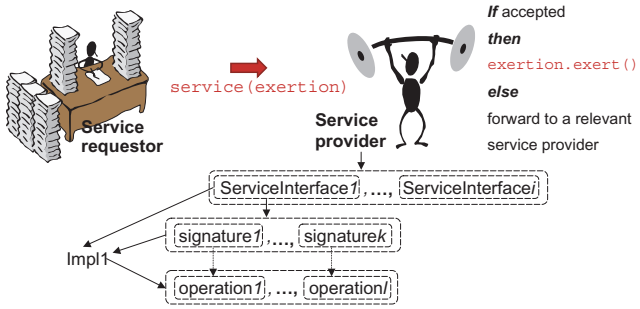

Figure 2. Service requestor and service provider relationship.

method. An exertion method defines a service provider (grid object) to be bound to at runtime. This network object provides the business logic to be applied to the exertion context model (data). The computing framework based on the concepts: context model, method, and exertion are called the CME framework.

A method is primarily defined by a provider type (interface) and selector (operation name) in the provider's interface. Optionally, additional attributes might be associated with the method, for example a provider's name or provider's identifier. The information included in the exertion method allows the GISO program to bind dynamically (at runtime) the exertion to the network object and process the exertion's context by one of its custom operations, which is defined by its published interface. This type of service provider is called a method provider. Another type of service provider is a context provider that provides shared data to the grid via the observer-observable paradigm. Thus, both context and method providers represent data and compute grid and operations respectively to be used in GISO programs.

Service providers are equal since they define a common top-level interface, but the interface might be implemented differently by different groups of network objects. Rather than the currently prevalent client/server model, in which all communication passes through and is controlled by a central server (e.g., web, FTP, mail, and application servers), in service-to-service (S2S), the communication goes directly from one grid's object to another grid's object. Because accessing these decentralized object nodes means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes operate outside the domain name system (DNS) and have significant or total autonomy from central servers.

Sun's Jini™ Network Connection Technology [23,24] is used to implement the functionality specified above. The discovery and lookup protocols that Jini™ defines allow for a dynamic federation of services to be created.

A simplified UML-diagram showing the service-oriented framework of GISO is illustrated in Figure 3. The core of the framework consists of the four middleware services: jobber (job coordination broker), cataloger (service catalog), spacer (exertion shared space – [25]), and provisioner (Rio provisioning service – [26]). The jobber coordinates exertion execution within the GISO job. It interprets the job control context supplied by the end-user-client or any of the service providers and coordinates the exertion execution accordingly to a defined control strategy.

There are two different ways in which the jobber can submit requests to the service providers. For an explicit access to the service provider the jobber can either use discovery to find a lookup service or use a service catalog (used by *CatalogExertionDispatcher*) for selecting a service from dynamically formed peer groups. A cataloger is a service-grid cache that periodically polls all relevant lookup services and maintains a cache of all the provider proxies that are registered with the lookup services for a particular peer group. The jobber can either discover lookup services (service registries) each time it needs to use a service or can find a cataloger that in turn performs continuous service discovery for the jobber. Weather the jobber finds an available service using a lookup registry or catalog, a proxy for the service is downloaded on to the jobber which invokes the service. Alternately, the jobber submits the service request implicitly into an exertion space that is a shared GISO exertion repository for executing asynchronously grid services. The exertion spaces provider, called the spacer (used by *SpaceExertionDispatcher*), holds the

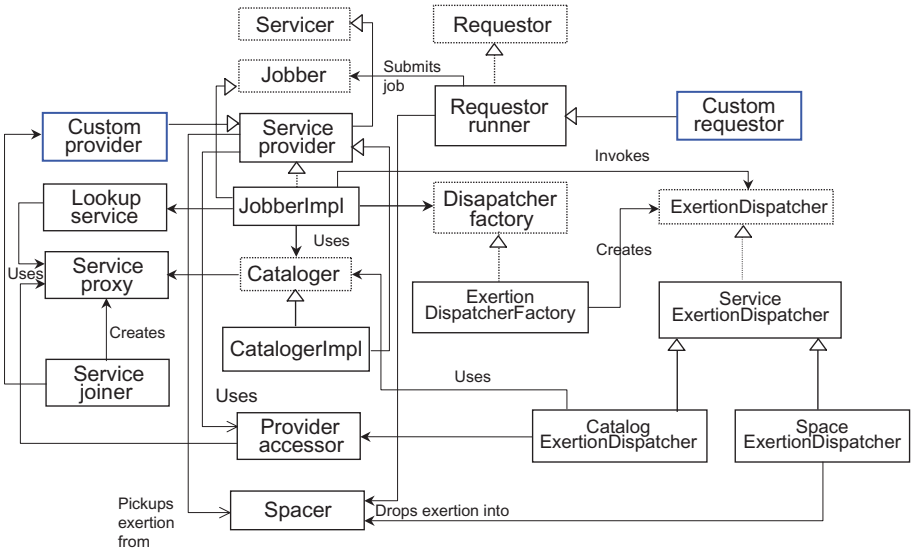

Figure 3. Job execution of GISO programs.

request and waits for a relevant service provider to acquire the request from the exertion space. This is essential so that the job does not have to abort due to non-availability of a service.

Central to the execution of the GISO/FIPER job is the *Exertion Dispatcher* that can dispatch exertions either by the lookup service, the cataloger, or the spacer. A factory pattern is used whereby the right dispatcher is called based on the number of exertions, control strategy (sequential or parallel), and a service access type (lockup service, cataloger, or spacer) all defined in the job control context. The framework supports autonomic provisioning whereby, the jobber can create or activate services on demand using a provisioning provider (provisioner). Also the GISO environment defines a common service provider interface and a set of utilities to create, startup, and register service providers as service peers. A service joiner is used for the static bootstrapping service providers and also for maintaining leases on registered proxies with the Jini lookup services.

The GISO programing environment consists of basic GISO/FIPER middleware providers: Jobber, Spacer, Cataloger, Provisioner, and Persister (as depicted in the GISO functional architecture in Figure 4). The Persister is a data store service that allows users to store and retrieve GISO data (service contexts) and programs (tasks and jobs). As described in Section 4, GISO user agents extensively use Persister functionality.

Domain specific providers used by GISO programs (jobs) can be developed using GISO development tools (Figure 2). All layers depend on the FIPER CME framework. Web clients and stand-alone requestors submit jobs to be executed by GISO/FIPER middleware services (infrastructure providers) in concert with the domain specific providers that complement the GISO programing.

4. GISO Programing and Development Tools

As previously stated the P2P service-oriented framework presented targets multiparty grid transactions. When performing a nested transaction, be it either a banking transaction or an engineering analysis, there are three basic components that can be identified. They 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 GISO/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 GISO/FIPER programing environment is an exertion. Each exertion contains a FiperMethod and a FiperContext object. The FiperMethod specifies what action that is to be taken in a given step in the process. The FiperContext contains all the data the FiperMethod 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 5. It is worth noting that recursion of FiperJobs is supported. That is any of the FiperTasks within a FiperJob can be a FiperJob itself.

The relationship between the GISO/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.

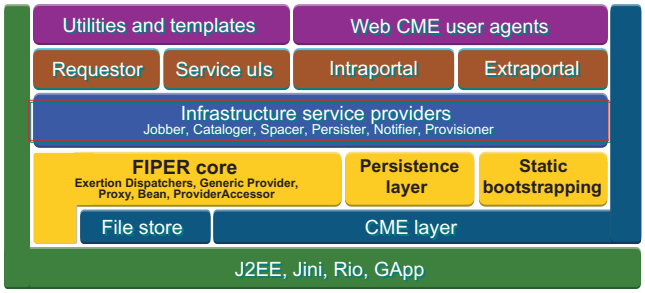

Figure 4. The GISO functional architecture.

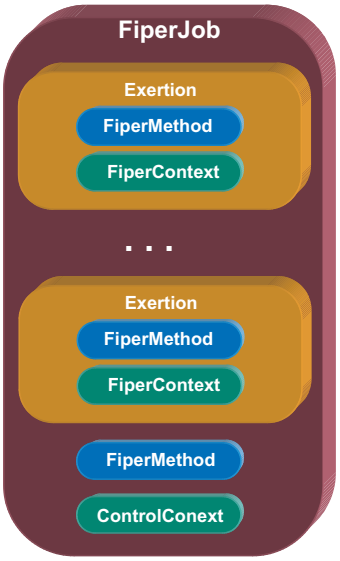

Figure 5. Program object hierarchy.

As an example of a nested transaction in the GISO/FIPER Environment consider the following engineering application, the mechanical analysis of a gas turbine component. The component, a turbine blade is shown in Figure 6. The process of performing a mechanical analysis consists of the following actions; generate solid geometry, discretize 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 7. Also depicted in Figure 7 are the associations between the three components of a nested transaction and the GISO/FIPER program objects.

To create the necessary program objects (FiperContext, FiperMethod, FiperTask, and

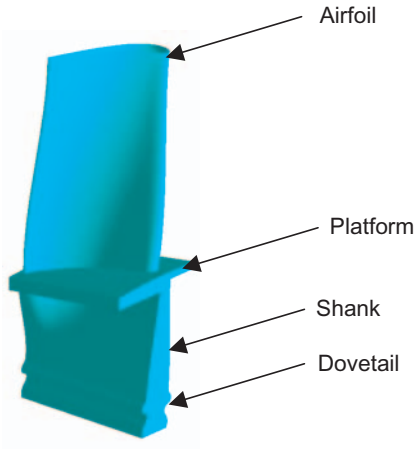

Figure 6. Turbine blade geometry.

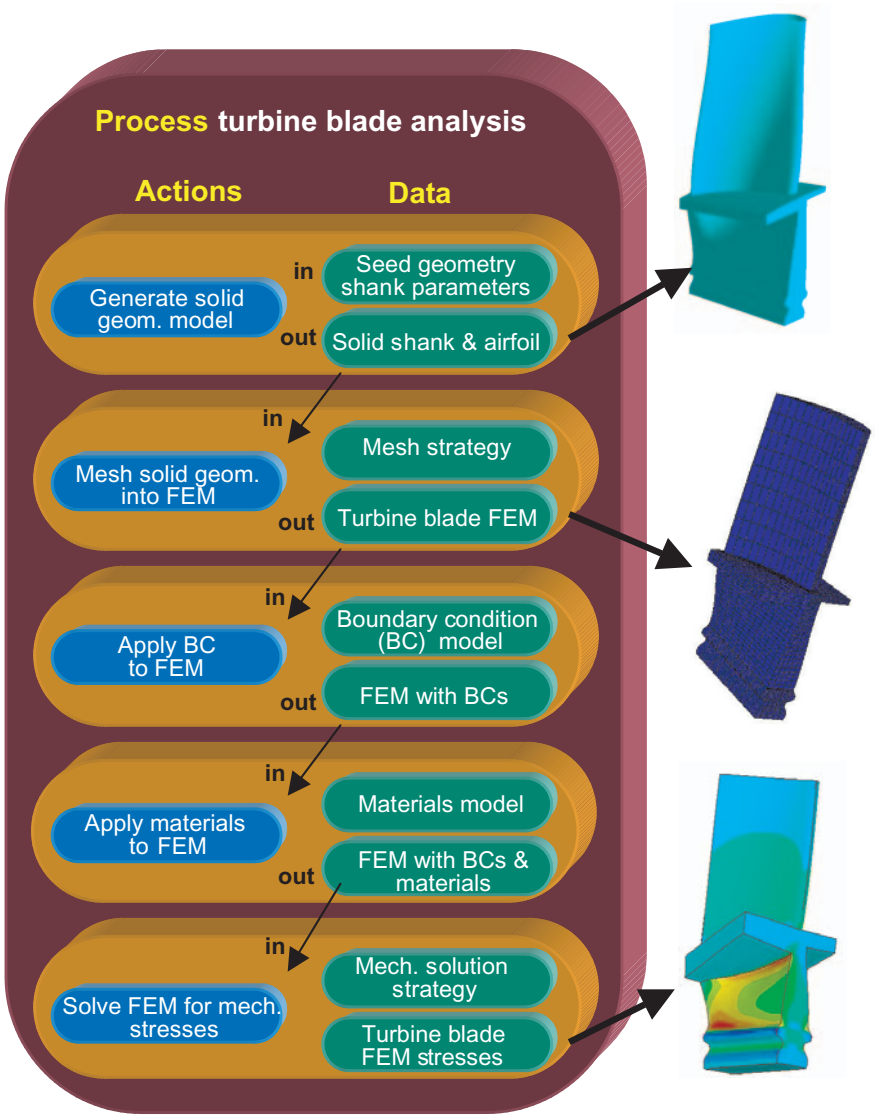

Figure 7. Process for the mechanical analysis of a turbine blade.

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 objects 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 8 shows the Fiper launcher page once logged into the Fiper environment. It can be seen that there are separate selections for the previously described program objects, FiperContext, FiperTask, and FiperJob. The FiperMethod object is created within the FiperTask menu selection.

4.1 Context Editor

The Context Editor allows the end-user to specify the data or references to the data along with the 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 9 shows the Context Editor along with the context for the first action or task in the Turbine Mechanical Analysis Job represented in Figure 7.

Figure 9 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.

4.2 FiperTask Editor

From the Fiper launcher in Figure 8 the end-user selects 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 GISO/FIPER Environment which contains the action and data for a nested transaction (Figure 7), the Methods field

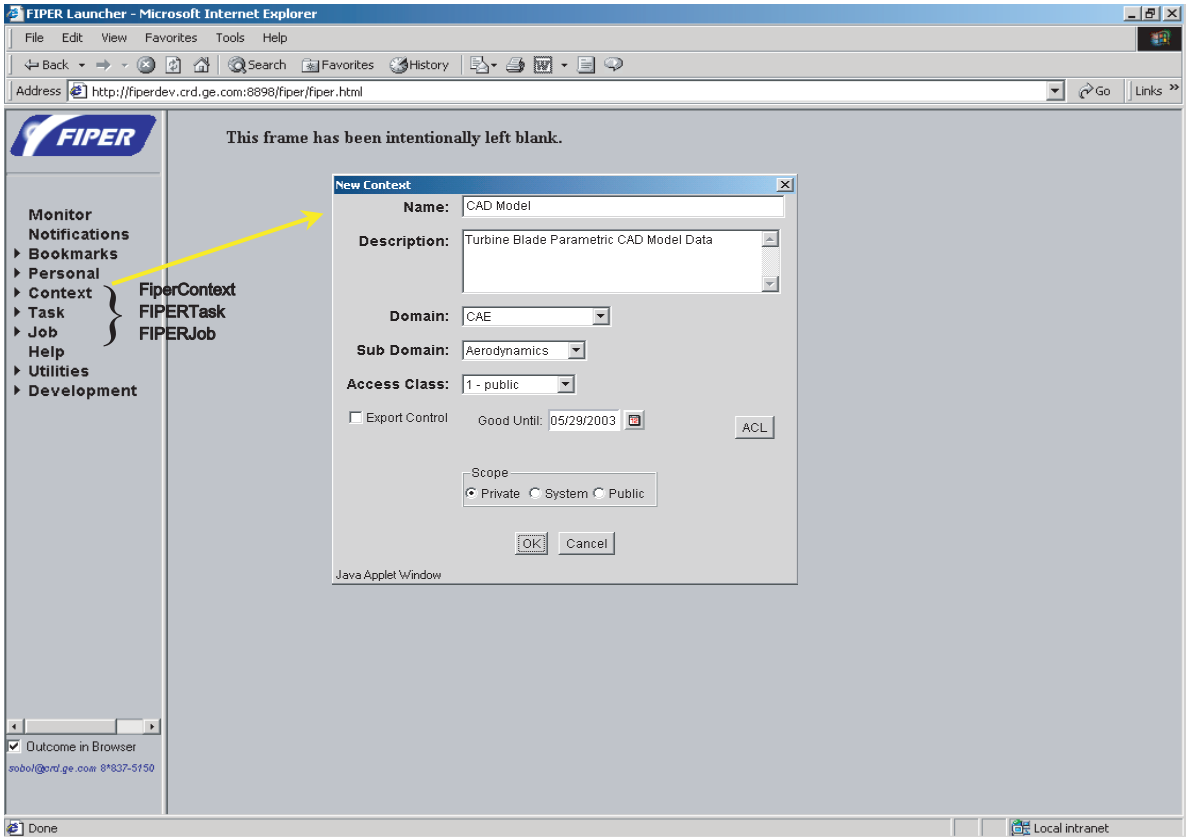

Figure 8. FIPER launcher and new Context dialog.

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 (Figure 10). Figure 10 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 9. Once all the actions/FiperTasks have been defined for a given process/FiperJob the FiperJob itself can then be constructed.

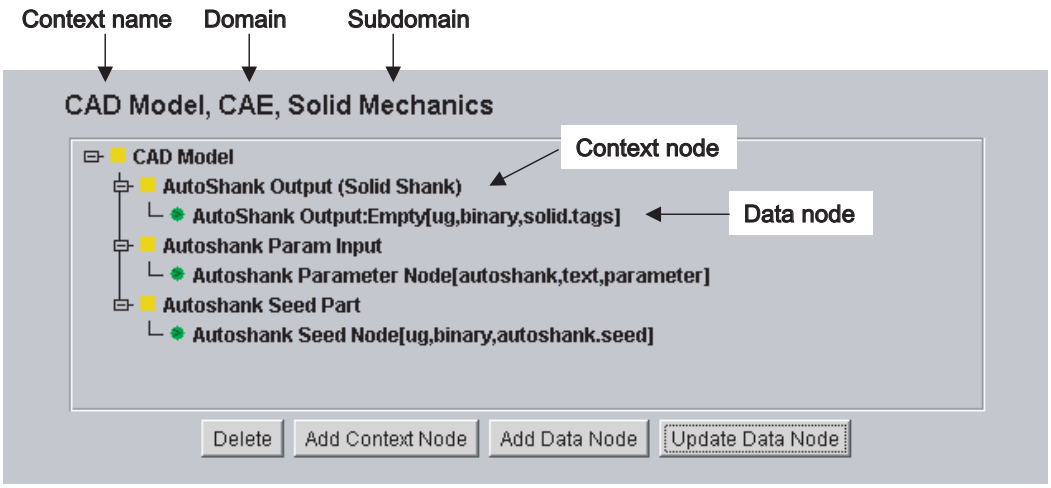

Figure 9. FiperContext Editor.

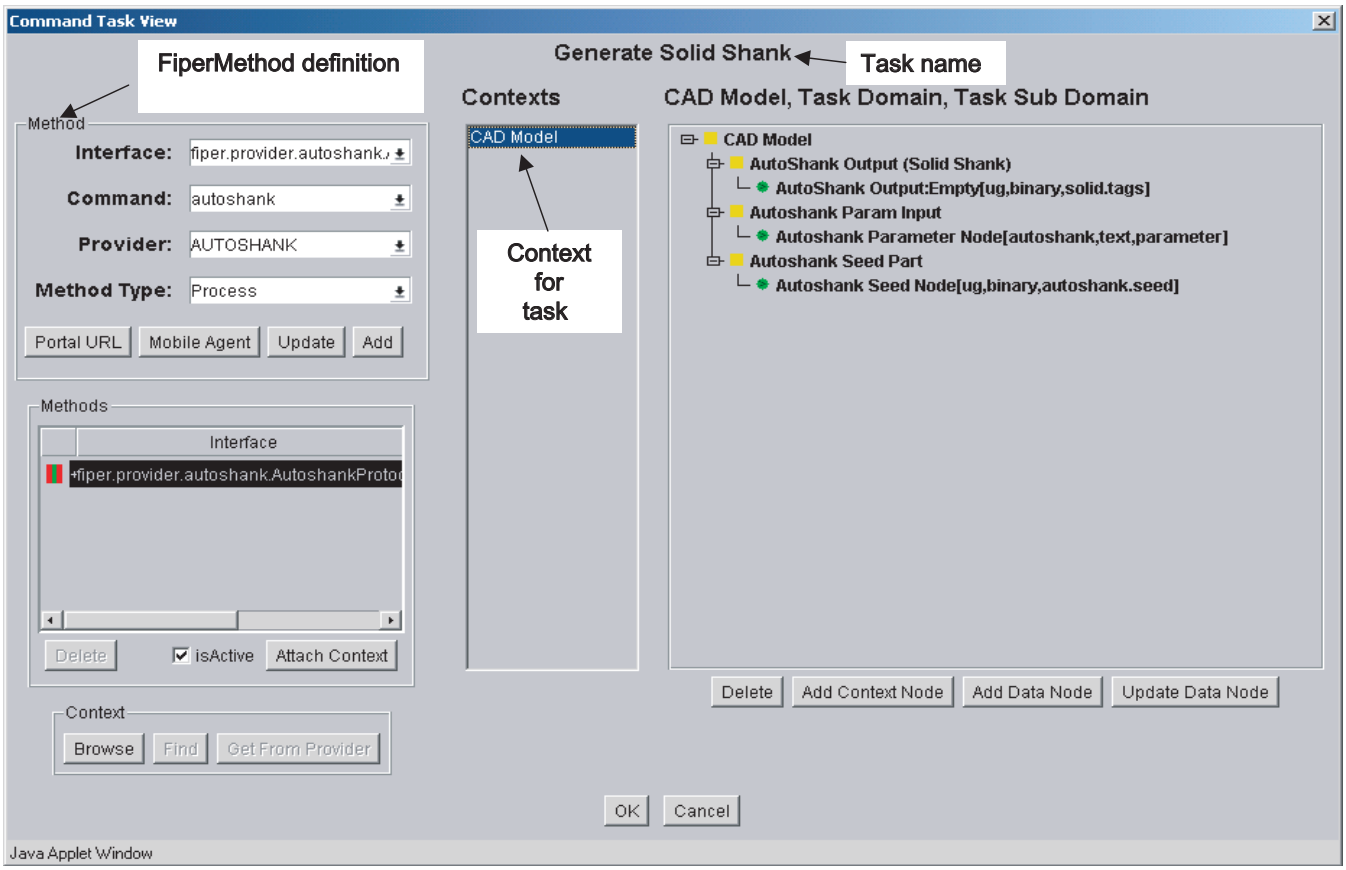

Figure 10. FiperTask, FiperMethod, and FiperContext Editor.

4.3 FiperJob Editor

Figure 11 illustrates the creation of the FiperJob represented in Figure 7. It contains all the tasks; Generate Solid Shank, Mesh Shank, Apply Boundary Conditions, Apply Materials, and Perform Stress Analysis.

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 the 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 11.

The FiperJob Context dialog for the Turbine Analysis Job is shown in Figure 12. 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 12 this has occurred by dragging the AutoShank Output Solid Shank Node contained in Task [0] onto the Solid Shank unnamed Fiper DataNode in Task [1].

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 10). 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.

4.4 FiperJob Monitor

The most critical capability that GISO/FIPER programing needs 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 requires 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 within which application the failure occurred. 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 applications to perform a nested

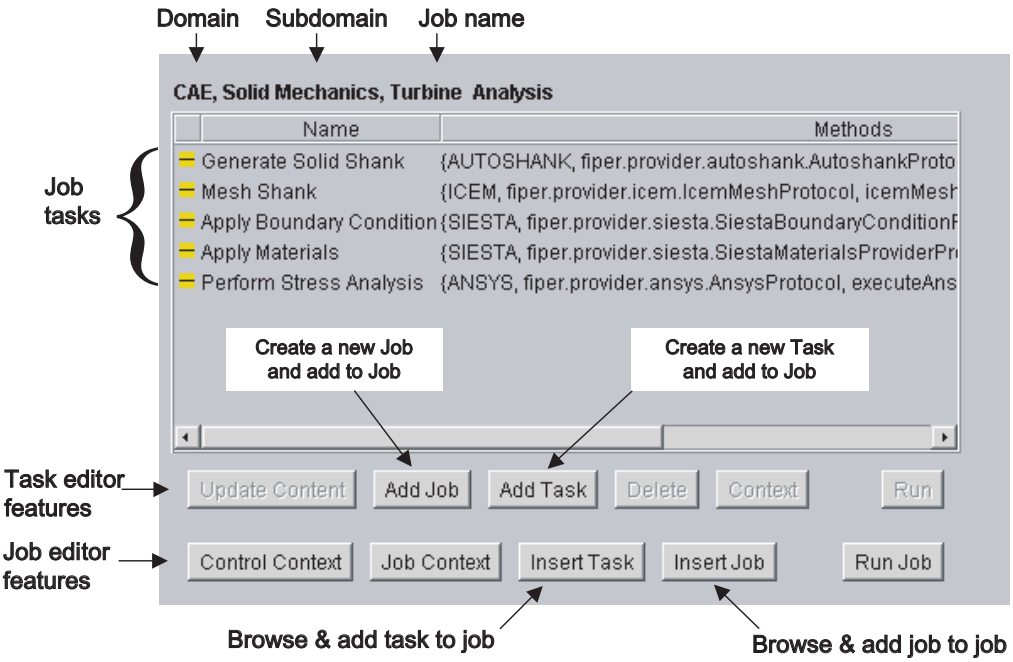

Figure 11. FiperJob Editor.

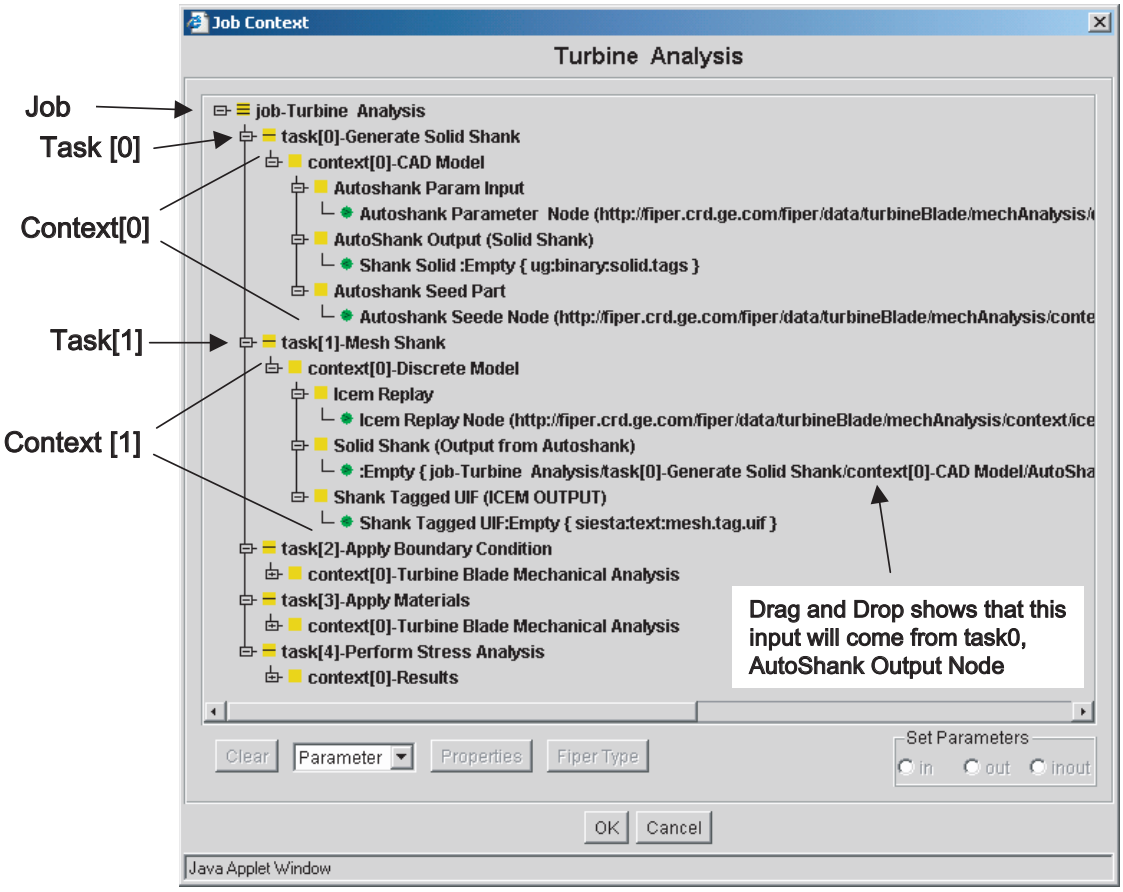

Figure 12. Fiper JobContext dialog.

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 programing that are necessary for the 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 GISO/FIPER Environment the monitoring/client process interaction is done using the Job Monitor. Figure 13 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.

5. Conclusions

In the GISO approach, object-oriented concepts are applied to the network and grid-oriented programs.

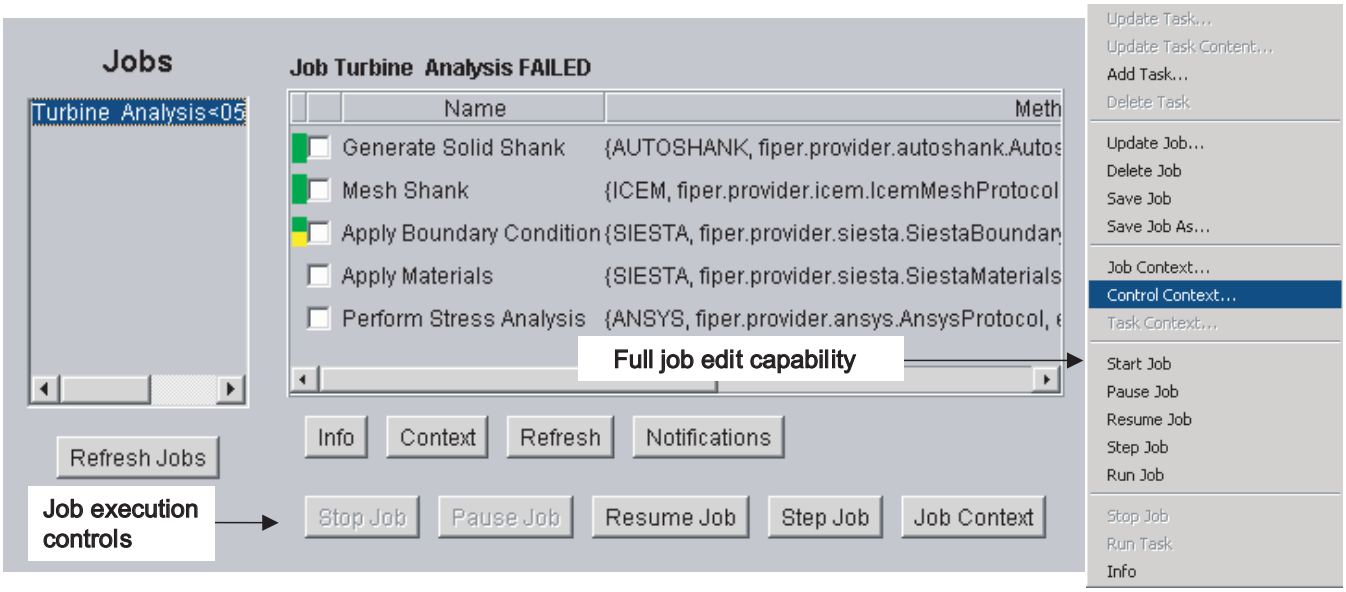

Figure 13. FiperJob monitor.

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 one 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 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 the General Electric Global Research Center and at General Electric's Aircraft Engines. At the research center approximately 10 developers were working within in the environment creating and running jobs. These jobs ranged from two tasks up to approximately 20 tasks with run times varying from minuets to days. The services were distributed across a half a dozen computing platforms. At the Aircraft Engine site approximately six design engineers were using the environment. Jobs for these applications contained on average six tasks and run times were on the order of minutes and hours. It was found that the ability to monitor the jobs and return usable exception information when jobs failed was critical to the successful transition of the technology.

References

1. Hafner, K. and Lyon, M. (1996). *Where Wizards Stay Up Late (a history of Internet development)*, Simon and Schuster: New York, NY.
2. Postel, J., Sunshine, C. and Cohen, D. (1981). *The ARPA Internet Protocol Computer Networks*, Chapter 5, pp. 261–271.
3. Postel, J. and Reynolds, J. (1987). Request for Comments Reference Guide (RFC1000). Internet Engineering Task Force.
4. Lynch, D.L. and Rose, M.T. (1992) *Internet System Handbook*, Reading MA: Addison-Wesley.
5. Lee, J. (ed.) (1992). Time-Sharing and Interactive Computing at MIT, *IEEE Annals of the History of Computing*, **14**(1).
6. Foster, I. and Kesselman, C. (eds) (1999). *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kaufmann Publishers.
7. Foster, I., Kesselman, C. and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International J. Supercomputer Applications*, **15**(3).
8. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maguire, T., Sandholm, T., Snelling, D. and Vanderbilt, P. (2003). Open Grid Service Infrastructure (OGSI). Technical Report, Global Grid Forum (OGF), Version 1.0, June 2003. Available from http://www.ggf.org/documents/GWD-R/GFD-R.015.pdf
9. Foster, I. and Kesselman, 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.

10. Grimshaw, A.S. and Wulf, W.A. (1997). The Legion Vision of a Worldwide Virtual Computer, *Communications of the ACM*, **40**(1): 39–45.
11. Smarr, L. (1997). Computational Infrastructure: Toward the 21st Century, Special Issue on Plans for a National Technology Grid, Communications of the ACM 40, 11.
12. National Research Council. (1993). National *Collaboratories: Applying Information Technology for Scientific Research*, Washington DC: National Academy Press.
13. Goel, S. and Sobolewski, M. (2003). Trust and Security in Enterprise Grid Computing Environment, In: *Proceedings of the IASTED Intl. Conference on Communication, Network, and Information Security*, New York, NY, Dec 10–12.
14. Kolonay, R.M., Sobolewski, M., Tappeta, R., Paradis, M. and Burton, S. (2002). *Network-Centric MAO Environment*, The Society for Modeling and Simulation International, 2002 Western Multiconference, San Antonio, Texas.
15. Kolonay, R. and Sobolewski, M. (2004). Grid Interactive Service-oriented Programming Environment, *Concurrent Engineering: The Worldwide Engineering Grid*, Tsinghua Press and Springer Verlag, ISBN 7-302-08802-0, pp. 97–102.
16. Lapinski, M. and Sobolewski, M. (2003). Managing Notifications in a Federated S2S Environment, *International Journal of Concurrent Engineering: Research & Applications*, **11**: 17–25.
17. Röhl, P.J., Kolonay, R.M., Irani, R.K., Sobolewski, M. and Kao, K. (2000). 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.
18. Sobolewski, M. (2002a). FIPER: The Federated S2S Environment, *JavaOne, Sun's 2002 Worldwide Java Developer Conference*, (http://servlet.java.sun.com/javaone/sf2002/conf/sessions/display-2420.en.jsp)
19. Sobolewski, M. (2002b). Federated P2P Services in CE Environments, *Advances in Concurrent Engineering, A.A. Balkema Publishers*, 2002, ISBN 90 5809 502 9, pp. 13–22.
20. Sobolewski, M., Soorianarayanan, S. and Malladi-Venkata, R.K. (2003). Service-oriented File Sharing, In: *Proceedings of the IASTED Intl. Conference on Communications, Internet, and Information Technology*, Scottsdale, AZ, Nov 17–19, pp. 633–639.
21. Soorianarayanan, S. and Sobolewski, M. 2004, Monitoring Federated Services in CE, *Concurrent Engineering: The Worldwide Engineering Grid*, Tsinghua Press and Springer Verlag, pp. 89–95, ISBN 7-302-08802-0.
22. Zhao, S. and Sobolewski, M. (2001). Context Model Sharing in the FIPER Environment, In: *Proc. of the 8th Intl. Conference on Concurrent Engineering: Research and Applications*, Anaheim, CA.
23. Edwards, W.K. (2000). Core Jini, 2nd edn, Prentice Hall, ISBN: 0-13-089408.
24. Jini Architecture Specification. Available at URL: http://www.sun.com/jini/specs/jini1_1.pdf
25. Freeman, E., Hopfer, S. and Arnold, K. (1999). Javaspace™ Principles, Patterns, and Practice, Addison-Wesley, ISBN: 0-201-30955-6.
26. Project Rio, http://rio.jini.org/

Michael Sobolewski

Dr. M. Sobolewski joined, as a Professor, the Computer Science Department, Texas Tech University in September 2002. While at GE Global Research Center he was a chief architect of the FIPER project.

Prior to coming to the U.S., during his 18-year career with the Polish Academy of Sciences, Warsaw, Poland, he was the head of the Picture Recognition and Processing Department, the head of the Expert Systems Laboratory, and was doing research in the area of knowledge representation, knowledge-based systems, pattern recognition, image processing, neural networks, and graphical interfaces. He has served as visiting professor, lecturer and consultant in Sweden, Finland, Italy, Switzerland, Germany, Hungary, Czechoslovakia, Poland, Russia, and the USA.

Dr. Ray Kolonay

Dr. Ray Kolonay is currently the Assistant to the Chief Scientist in the Air Vehicles Directorate, Air Force Research Laboratory at Wright-Patterson Air Force Base Ohio.

He has over twenty years experience in the development, use, and support of mechanical analysis and automated design tools and methods. Recent research interests have been in the development of large scale distributed engineering analysis and design computing environments. While at the General Electric Global Research Center he was the team leader of the Federated Intelligent Product EnviRonment (FIPER) project – a 21.5 million dollar NIST Advanced Technology Program that focused on the development of a Network-Centric e-Engineering environment enabling global access and communication between engineering data and applications for mechanical analysis and design. His career has been focused on developing methods for the automated Multi-Disciplinary Analysis and Optimization (MDA/MDO) of flight vehicle structures ranging from complete airframes to turbine blade components. His areas of technical expertise include structures, structural dynamics, linear/nonlinear aeroelasticity, engineering sensitivity analysis, optimization, and distributed engineering computing environments.