Application Experiences with the Globus Toolkit

Sharon Brunett[1] Karl Czajkowski[2] Steven Fitzgerald[2] Ian Foster[3]
Andrew Johnson[4] Carl Kesselman[2] Jason Leigh[4] Steven Tuecke[3]

[1] Center for Advanced Computing Research
California Institute of Technology
Pasadena, CA 91125

[2] Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292

[3] Mathematics and Computer Science
Argonne National Laboratory
Argonne, IL 60439

[4] Electronic Visualization Lab
University of Illinois
Chicago, IL 60637

Abstract

The development of applications and tools for high-performance "computational grids" is complicated by the heterogeneity and frequently dynamic behavior of the underlying resources; by the complexity of the applications themselves, which often combine aspects of supercomputing and distributed computing; and by the need to achieve high levels of performance. The Globus toolkit has been developed with the goal of simplifying this application development task, by providing implementations of various core services deemed essential for high-performance distributed computing. In this paper, we describe two large applications developed with this toolkit: a distributed interactive simulation and a teleimmersion system. We describe the process used to develop the applications, review lessons learned, and draw conclusions regarding the effectiveness of the toolkit approach.

1. Introduction

The Globus grid toolkit is a collection of software components designed to support the development of applications for high-performance distributed computing environments, or "computational grids" [11, 12]. The Globus toolkit is an implementation of a "bag of services" architecture, which provides application and tool developers not with a monolithic system but rather with a set of standalone services. (Other candidate grid architectures include the use of object-based technologies [16, 15], web technologies [14, 27], and CORBA [17].) Each Globus component provides a basic service, such as authentication, resource allocation, information, communication, fault detection, and remote data access. Different applications and tools can combine these services in different ways to construct "grid-enabled" systems.

Various components of the Globus toolkit are described in detail in other papers [11, 7, 9, 13]. Briefly, the toolkit comprises the core services listed in Table 1, plus a selection of higher-level services defined in terms of these core services. Each core service defines an application program interface (API) that provides a uniform interface to a local service. For example, the Globus Resource Allocation Manager (GRAM) service provides an API for requesting that computations be started on a computational resource, and for managing those computations once they are started [7]. Higher-level services use core services to implement more complex global functionality. For example, resource brokers and co-allocators use services provided by GRAMs and by the Globus information service (the Metacomputing Directory Service, or MDS [9]) to locate available resources and to start computations across computations of resources, respectively. Application-specific scheduling techniques [2] can also be used.

The Globus toolkit has been used to construct the Globus Ubiquitous Supercomputing Testbed, or GUSTO: a large-scale testbed that spans over 20 sites and includes over 3,000 compute nodes for a total compute power of over 2 TFLOPS. Over the past six months, we and others have used this testbed to conduct a variety of application experiments, in such areas as multiuser collaborative environments (teleim-

Table 1. Core Globus services. As of early 1998, these include only those services deemed essential for an evaluation of the Globus design philosophy on realistic applications and in medium-scale grid environments.

Service	Name	Description
Resource Management	GRAM	Resource allocation & process management
Communication	Nexus	Unicast & multicast communication services
Information	MDS	Distributed access to structure & state information
Security	GSI	Authentication & related security services
Health & Status	HBM	Monitoring of health & status of components
Remote Data Access	GASS	Remote access to data via sequential & parallel interfaces
Executable Management	GEM	Construction, caching, & location of executables

mersion), computational steering, distributed supercomputing, and high-throughput computing.

The goal of this paper is to review what has been learned from these experiments regarding the effectiveness of the toolkit approach. To this end, we describe two of the application experiments in detail, noting what worked well and what worked less well. These two applications are a distributed supercomputing application, SF-Express, in which multiple supercomputers are harnessed to perform large distributed interactive simulations; and a teleimmersion application, CAVERNsoft, in which the focus is on connecting multiple people to a distributed simulated world.

We believe that the results of these experiments indicate that the Globus toolkit architecture is effective, at least for the applications considered to date. Large, complex applications can be either adapted to execute in a grid environment (e.g., SF-Express) or developed from scratch for grid computing (e.g., CAVERNsoft) without unusual difficulty, and with a saving in cost, complexity, and usability relative to similar codes developed without our toolkit. The experiments also point to areas in which further work is required: for example, code and data management, and fault tolerance.

2. SF-Express

The first application that we consider, SF-Express, is a distributed interactive simulation (DIS) application that harnesses multiple supercomputers to meet the computational demands of large-scale network-based simulation environments [24]. A large simulation may involve many tens of thousands of entities and requires thousands of processors. Globus services can be used to locate, assemble, and manage those resources. For example, in one experiment in March 1998, SF-Express was run on 1352 processors distributed over 13 supercomputers at nine sites [5] (Figure 2). This experiment involved over 100,000 entities, setting a new world record for simulation and meeting a performance goal that was not expected to be achieved until 2002.

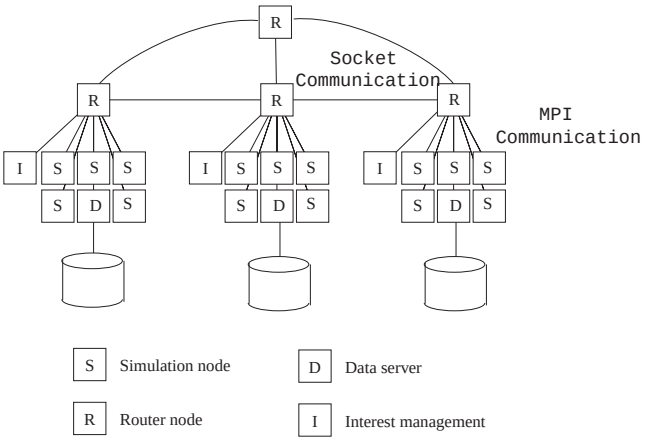

Figure 1. SF-Express architecture

An SF-Express computation is distributed across a large number of simulation nodes, where each node is responsible for simulating the behavior of entities assigned to it. During the simulation, entity state information, such as the position and velocity, is exchanged. Current DIS implementations broadcast state update information, an approach that is not practical for the large-scale simulations. SF-Express uses a technique called interest management to reduce the amount of communication required. As illustrated in Figure 1, simulation nodes are organized into groups, and a router node is associated with each group. Routers keep track of which simulation nodes contain entities that can potentially interact with one another, and entity state updates are sent only between those routers responsible for interacting simulation nodes. Entities are assigned to nodes in a way that preserves physical locality to further reduce communication requirements.

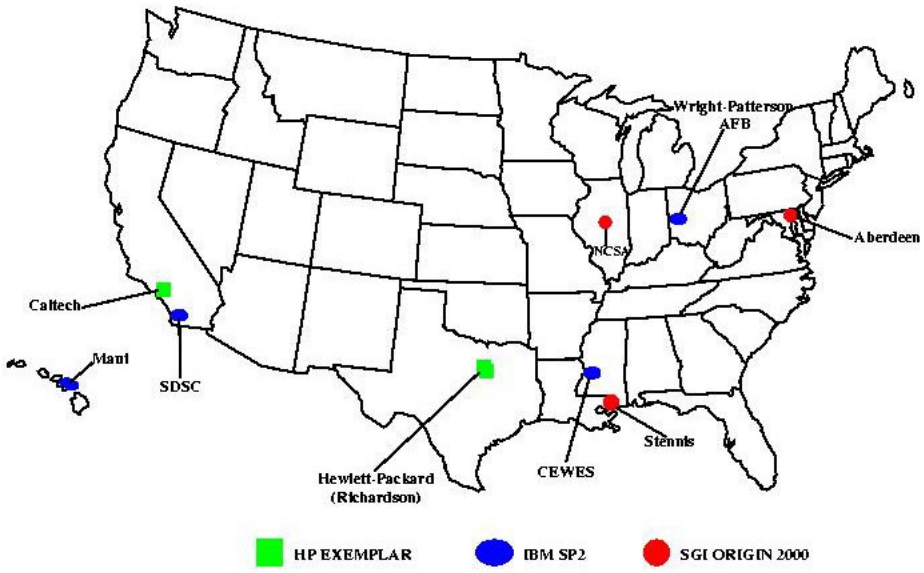

Figure 2. The nine sites involved in the record-setting distributed interactive simulation

2.1. SF-Express Experiences

SF-Express is interesting as a test case for the Globus toolkit because a distributed implementation existed before work started on the use of Globus services. As illustrated in Table 2 and explained in the following, a Globus-based implementation was constructed (and continues to be constructed) by incorporating Globus services incrementally to improve existing functionality or add new functionality. This incremental process made it possible to study the impact on complexity and usability of incorporating Globus components.

Resource allocation and security. Prior to the use of Globus services, simply starting SF-Express on multiple supercomputers was a painful task. The user had to log in to each site in turn and recall the arcane commands needed to allocate resources and start a program. This obstacle to the use of distributed resources was overcome by encoding resource allocation requests in terms of the GRAM API. GRAM and associated GSI services could then be used to handle authentication, resource allocation, and process creation at each site.

Co-allocation. SF-Express requires that the allocation of the resources used by its simulation nodes and routers, and the starting of the relevant processes, occur simultaneously. We used a Globus co-allocation service called DUROC to coordinate multiple GRAM requests, construct an unified job from individual GRAM requests, or subjobs, and prevent application components from starting until all the required resources have been obtained. As shown in Figure 3, the changes to SF-Express required to interface to GRAM and the co-allocation service are minimal: The co-allocator library function `ca_barrier` is added to ensure that the simulation proceeds only when all resources have been obtained, and the function `sfe_exchange_info` uses GRAM and co-allocator functions to gather and broadcast the IP addresses and port numbers of all router nodes.

Because of the sheer size of the computation, the 100,000-vehicle run of SF-Express required exclusive access to the 13 supercomputers used by the simulation. This exclusive access was achieved by manually coordinating dedicated time on all machines. Globus was then used to start the computation, assured that it could obtain the resources requested. In general, dedicated time is hard to come by on a single supercomputer, let alone thirteen. Thus, such manual scheduling is the exception, not the norm.

During the test runs leading up to the large simulation, Globus was used to dynamically allocate resources via the normal scheduling queues on the desired machines. In this usage mode, it was possible for a request for a set of nodes on a specific machine could be arbi-

```
main(int argc, char **argv) {
  router_info my_router;
  router_info routers[MAX_ROUTERS];
  /* Wait for rest of nodes to start */
  ca_barrier();
  /* Connect up routers *
  my_router.addr = router_addr();
  my_router.port = router_port();
  /* Get locations of other routers */
  sfe_exchange_info(&my_router,routers);

  /* Start MPI in local set of nodes */
  MPI_Init(argc,argv);
  /* Rest of code */
```

Figure 3. The Globus-enhanced SF-Express startup code extends the code used on a single parallel computer with two new calls, as described in the text

trarily delayed. The dynamic editing capabilities of DUROC proved extremely helpful in these situations, enabling dynamic editing of the resource set until an appropriate collection of machines was obtained in a timely fashion.

Fault detection. The nondeterministic nature of the DIS simulation algorithm means that a simulation can sometimes continue in a useful fashion even if a component has failed. For this reason, we use the Globus heartbeat monitor (HBM) to provide fault detection and notification. This Globus service provides a wide-area mechanism for monitoring the state of the components of a computation, notifying a status monitor of failure.

Remote data access. SF-Express generates two types of file I/O: it reads a variety of databases and configuration files and writes an error log. We use the Global Access to Secondary Storage (GASS) service to simplify access to error logs. Each machine participating in SF-Express generates a log file indicating the status of the simulation on that machine. Prior to using Globus, an SF-Express user had to log into the machine running a piece of the simulation to examine the contents of the log. Using GASS, which supports append-mode file writes, we can write logfile entries to a remote location. GASS also supports automatic fetching of files when they are opened, in a similar fashion to UFO [1], and program-controlled prestaging of files. We plan to use this facility to provide access to the read-only files accessed by SF-Express.

2.2. SF-Express Lessons Learned

It is difficult to quantify the impact of Globus on SF-Express. However, we can offer strong anecdotal evidence as to its usefulness. One major benefit was a significant decrease in the amount of time it took to start up and shut down a simulation. This facilitated both the debugging of the application and the agility with which the configuration could be changed during the actual large-scale run. The second major benefit of Globus was the simplification of application configuration. Prior to using Globus, a configuration file was used to list the IP addresses and ports at which simulation components running on different machines could be contacted in order to connect the pieces of the simulation to one another. Globus eliminated the need for this static information, since dynamic Globus startup mechanisms supported the distribution of this information. This reduced the complexity of startup and eliminated a major source of error, increasing our confidence that the simulation would run if last-minute changes in computational resources had to be made.

Experience with SF-Express also revealed areas in which existing Globus services could be improved and suggested additional opportunities for use of Globus services in SF-Express. The primary Globus deficiency revealed by this work relates to the implementation of the co-allocation service. The co-allocation service initially designed for Globus provided a static co-allocation model in which a request failed if any component of the request failed. While this model vastly simplified SF-Express startup, it meant that a startup problem on any one computer required that we terminate the healthy components of the job and restart the computation. Yet in practice, individual GUSTO components failed frequently. Interestingly, software problems rather than hardware and network failures were the leading cause of difficulty. Examples of failure states that we observed include system paralysis due to generation of a large core file; failure of local scheduling systems; intermittent application crashes (due to bugs in the original SF-Express code); and operator error. These problems were especially troublesome because SF-Express has a startup time of over 15 minutes. Building on this experience, we have designed a more flexible, dynamic co-allocation model in which the contents of a co-allocation request can be modified until the program starts to execute.

Experience suggested three additional areas in which Globus components could be used in an SF-Express

Table 2. A grid-aware version of SF-Express is being constructed incrementally: Globus services are incorporated one by one to improve functionality and reduce application complexity. The Status field indicates code status as of early 1998: techniques are in use (Y), are experimental or partial use (y), or remain to be applied in the future (blank).

Services	How Used	Benefits	Status
GRAM, GSI	Start SF-Express on supercomputers	Avoid need to log into and schedule each system	Y
+ Co-allocator	Distributed startup and management	Avoid application-level check-in and shutdown	Y
+ MDS	Use MDS information to configure computation	Performance, portability	y
+ Resource Broker	Use broker to locate appropriate computers	Code reuse, portability	y
+ Nexus	Encode communication as Nexus RSRs	Uniformity of interface, access to unreliable comms	y
+ HBM	Routers check in with application-level monitor	Provide degree of fault tolerance	Y
+ GASS	Use to access terrain database files, etc.	Avoid need to prestage data files	y
+ GEM	Use to generate and stage executables	Avoid configuration problems	

implementation: configuration (using MDS information for autoconfiguration, hence improving portability and performance); resource brokering (providing an SF-Express-specific resource broker, hence reducing the need for human involvement in the resource selection process); and communication (using Globus communication services to access multicast and quality of service mechanisms, hence improving scalability and simulation performance).

3. CAVERNsoft

The second application that we consider is CAVERNsoft [20], a software infrastructure designed to support the rapid development of teleimmersive applications. In teleimmersion, immersive virtual reality environments are used over networks to provide shared access to simulated virtual spaces for design, collaboration, entertainment, education, and so forth [20, 8] The producers and consumers of the virtual environment, as well as the datasets and simulations on which the virtual space is based, are frequently geographically distributed, placing heavy demands on distributed computing support. For example CAVERNsoft is the underlying data distribution architecture for VisualEyes, a collaborative design review system used by General Motors (Figure 4); CAVE5D, a collaborative system for visualizing multidimensional weather and environmental hydrology data; Virtual Temporal Bone, a collaborative learning environment to teach medical students the structure and function of the inner ear; and TIDE (Teleimmersive Data-mining Environment), a collaborative system for viewing decision-trees generated by data-mining classifiers.

CAVERNsoft supports teleimmersive application development by providing runtime support for the definition, update, and access of shared virtual worlds. Its layered architecture has at its core an information resource broker (IRB) that supports the maintenance of shared databases, and above this libraries for the manipulation of avatars and manipulation of audio and video streams.

3.1. CAVERNsoft Experiences

The initial version of the IRB makes extensive use of the Globus toolkit's communication service, and hence we focus our discussion on this aspect of the system. We also note opportunities that we have identified for the use of other services.

Communication in teleimmersive applications is complicated by the variety of flows that need to be handled. DeFanti and Stevens [8] identify nine distinct types of flow (control, text, audio, video, tracking,

Figure 4. VisualEyes, a distributed design review system from General Motors. The design of a car interior is projected on the walls of the CAVE. Remote participants are able to enter the space and see and critique the design from various points of views and make minor adjustments.

database, simulation, haptics, rendering), each with distinctive requirements in terms of both performance and the mechanisms that can be used to implement the flows [18]. For example, tracking information need not be propagated reliably but can almost always benefit from multicast, while database updates require reliable communication but cannot always use multicast capabilities.

Historically, teleimmersion systems (and similar applications) either have used a single low-level communication protocol for all flows (e.g., TCP/IP [26, 3, 6]) or have used a mixture of different, often specialized APIs for different flows [23, 25, 22]. Neither approach is ideal. We believe that a better approach is to code to a single API that allows both high-level specification of communication structure and independent specification of the mechanisms used to achieve that communication. Nexus, the communication component of the Globus toolkit, meets this requirement.

The Nexus communication library allows applications to define communication links over which can be performed asynchronous remote procedure calls called remote service requests (RSRs). Associated operations allow us to select the underlying communication protocol used for a particular RSR according to when, where, and what is being communicated [10]. Hence, if two components of a CAVERNsoft application are located on different nodes of a parallel computer, Nexus operations can be mapped onto efficient local communication methods, such as MPI; if the components are located on different computers, Nexus communication operations can be mapped into unreliable, wide-area communication protocols. More important, this flexibility means that CAVERNsoft can specify all communication operations in terms of a single abstraction (and API) and then vary the method used according to the type of flow that the communication is associated with. For example, tracking events can be performed with an unreliable multicast protocol, while database updates are propagated with reliable unicast or multicast.

Nexus also allows quality of service (QoS) specifications [18] to be associated with communication links. These specifications can then be translated into an RSVP [4] or similar reservation if the underlying network supports this capability. MDS information can be used to determine the capabilities and utilization of the underlying networks, and hence to evaluate trade-offs between different protocols.

3.2. CAVERNsoft Lessons Learned

In teleimmersion the major criteria that decide the suitability of a networking infrastructure are its portability across multiple collaborating platforms, the manner in which its communications model matches the data dissemination model of teleimmersion, its ability to support a wide variety of networking protocols and QoS, and the overhead imposed by using a higher-level library.

- Portability. Teleimmersive applications typically interact with databases and simulations that are supported by hardware platforms that are dissimilar to the systems that drive the VR hardware. The availability of Nexus on all major platforms (Windows NT is the only significant exception and will be provided shortly) makes CAVERNsoft portable.
- Uniform Networking Interface. CAVERNsoft is a second-generation teleimmersion system, designed in response to lessons learned with two earlier applications (CALVIN, a collaborative architectural layout system, and NICE, a collaborative educational system) [25, 21]. CALVIN employed a distributed shared-memory system that used completely reliable protocols as the main mechanism for information distribution. Although the DSM model simplified programming, it had two significant limitations: its use of a reliable protocol resulted in significant latency during the delivery of tracking events that normally preferred

low-latency transmission; and its lack of an event-triggering mechanism meant that a program had to constantly poll for any changes in the DSM. In NICE, however, a fully message-passing system was implemented that used both reliable and unreliable network protocols. In addition, an event-triggering mechanism was implemented so that new messages would trigger the proper update of the virtual environment. NICE's main limitation was that its use of multiple networking protocols (multicast, UDP, and TCP) required each portion of the program that used these protocols to be managed separately.

CAVERNsoft combined a DSM with a message-passing system. Moreover, through Nexus's uniform networking interface, CAVERNsoft provided support for reliable TCP, unreliable UDP, and unreliable multicasting. In the future, other protocols such as RTP and reliable multicast will be possible. Furthermore, since Nexus will also provide networking QoS capabilities, these capabilities will also become a standard part of CAVERNsoft.

- Half a Remote Procedure Call. The Nexus remote service request (RSR) is essentially a remote procedure call (RCP) without a response that blocks the calling process. This asynchronous RPC is necessary for teleimmersive applications because these realtime applications cannot call a remote procedure and block to wait for a response. Responses must be managed asynchronously: that is, they are sent to handlers when an asynchronous event arises. This is exactly the model of operation employed by the Nexus RSR.

- Performance Overhead. A number of comparisons have been made between sending data via standard UNIX TCP calls and Nexus RSRs [19]. Our results showed the only significant overhead imposed by Nexus to be a single redundant memory copy of an application's data to Nexus transmission and receiving buffers. This redundant copy will significantly impact teleimmersive applications when large datasets are distributed; however, planned extensions to Nexus will provide a set of nonbuffered RSRs, hence eliminating this overhead.

Having completed construction of this initial CAVERNsoft prototype, we are considering a number of extensions. A first step is to use Globus security infrastructure and resource management mechanisms to handle authentication and resource allocation on distributed resources; currently, these tasks are handled in a rather ad hoc manner. A next task will be to use MDS to guide optimized configuration decisions for the IRB implementation. We also anticipate that the Globus instrumentation service will be of use.

4. Conclusions

We have used the Globus toolkit to implement a variety of distributed computing applications, two of which we have described here. Each application typically uses a different set of grid services. Nevertheless, all have in common that an existing application code or application structure was modified for grid execution fairly easily by introducing appropriate components chosen from the Globus bag of services. This means that the application did not have to be entirely rewritten before it could operate in a grid environment: services could be introduced into an application incrementally, with functionality increasing at each step. In this respect, we believe that our initial experiments with the toolkit have been a success and suggest that the approach should be pushed further.

Our experiments also teach us lessons about the grid environment, most notably the importance of fault tolerance. While detecting and dealing with failure are known to be critical issues in distributed systems, we have been astonished by the range of error conditions that we have encountered. Fortunately, we find that the availability of relatively simple fault detection techniques can render applications significantly more robust. In this respect, the integrated, network-accessible information service provided by Globus (MDS) proved valuable as a mechanism for detecting and recovering from failure. MDS information allowed us develop a range of general and application-specific high-level tools such as resource brokers and status monitors.

Future work will focus on further refining the current Globus services by studying their use in additional applications. We are continuing to work with both SF-Express and CAVERNsoft to make them more grid-aware. We are also working to extend the Globus toolkit to incorporate additional services, notably in the area of executable management. Our goal is to make these and other applications robust and simple enough so that the use of computational grids becomes commonplace.

Acknowledgments

We gratefully acknowledge the contributions of other members of the Globus team, in particular, Joe Bester, Joe Insley, Nick Karonis, Gregor von

Laszewski, Stuart Martin, Warren Smith, and Brian Toonen at Argonne National Laboratory; Mei-Hui Su and Marcus Thiebaux at USC/ISI; and Craig Lee and Paul Stelling at the Aerospace Corporation. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; and by the National Science Foundation.

References

[1] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. Extending the operating system at the user level: The UFO global file system. In *1997 Annual Technical Conference on UNIX and Advanced Computing Systems (USENIX'97)*, Jan. 1997.

[2] F. Berman. High-performance schedulers. In *[12]*.

[3] K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, 1993.

[4] R. Braden, L. Zhang, D. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 functional specification. Internet Draft, Internet Engineering Task Force, 1996.

[5] S. Brunett, D. Davis, T. Gottschalk, P. Messina, and C. Kesselman. Implementing distributed synthetic forces simulations in metacomputing environments. In *Proceedings of the Heterogeneous Computing Workshop*, pages 29–42. IEEE Computer Society Press, 1998.

[6] C. Carlsson and O. Hagsand. DIVE - a multi-user virtual reality system. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*. 1993.

[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[8] T. DeFanti and R. Stevens. Tele-immersion. In *[12]*.

[9] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 365–375. IEEE Computer Society Press, 1997.

[10] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke. Managing multiple communication methods in high-performance networked computing systems. *Journal of Parallel and Distributed Computing*, 40:35–48, 1997.

[11] I. Foster and C. Kesselman. The Globus project: A progress report. In *Proceedings of the Heterogeneous Computing Workshop*, 1998. to appear.

[12] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[13] I. Foster, C. Kesselman, and S. Tuecke. The Nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

[14] G. Fox and W. Furmanski. High-performance commodity computing. In *[12]*.

[15] D. Gannon and A. Grimshaw. Object-based approaches. In *[12]*.

[16] A. S. Grimshaw, W. A. Wulf, and the Legion team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), Jan. 1997.

[17] O. M. Group and X/Open. Common object request broker: Architecture and specification, 1991.

[18] R. Guérin and H. Schulzrinne. Network quality of service. In *[12]*.

[19] J. Leigh. *CAVERN and a Unified Approach to Support Realtime Networking and Persistence in TeleImmersion*. PhD thesis, University of Illinois at Chicago, Dec 1997.

[20] J. Leigh, A. Johnson, and T. A. DeFanti. CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Virtual Reality: Research, Development and Applications*, 2(2):217–237, December 1997.

[21] J. Leigh, A. E. Johnson, C. A. Vasilakis, and T. A. DeFanti. Multi-perspective collaborative design in persistent networked virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium '96*, pages 253–260, Apr. 1996.

[22] M. R. Macedonia and M. J. Zyda. A taxonomy for networked virtual environments. In *Proceedings of the 1995 Workshop on Networked Realities*. 1995.

[23] J. Mandeville, J. Furness, and T. Kawahata. Greenspace: Creating a distributed virtual environment for global applications. In *Proceedings of IEEE Networked Virtual Reality Workshop*. IEEE Computer Society Press, 1995.

[24] P. Messina, S. Brunett, D. Davis, T. Gottschalk, D. Curkendall, L. Ekroot, and H. Siegel. Distributed interactive simulation for synthetic forces. In *Proceedings of the 11th International Parallel Processing Symposium*, 1997.

[25] M. Roussos, A. Johnson, J. Leigh, C. Valsilakis, C. Barnes, and T. Moher. NICE: Combining constructionism, narrative, and collaborati on in a virtual learning environment. *Computer Graphics*, 31(3):62–63, August 1997.

[26] C. Shaw and M. Green. The MR toolkit peers package and environment. In *Proceedings of the Virtual Reality Annual International Symposium*. IEEE Computer Society Press, 1993.

[27] A. Vahdat, E. Belani, P. Eastham, C. Yoshikawa, T. Anderson, D. Culler, and M. Dahlin. WebOS: Operating system services for wide area applications. In *7th Symposium on High Performance Distributed Computing, to appear*, July 1998.