Communication Services for Advanced Network Applications

John Bresnahan, Ian Foster, Joseph Insley, Brian Toonen, Steven Tuecke
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL, U.S.A.

Abstract *Advanced network applications such as remote instrument control, collaborative environments, and remote I/O are distinguished from "traditional" applications such as videoconferencing by their need to create multiple, heterogeneous flows with different characteristics. For example, a single application may require remote I/O for raw datasets, shared controls for a collaborative analysis system, streaming video for image rendering data, and audio for collaboration. Furthermore, each flow can have different requirements in terms of reliability, network quality of service, security, etc. We argue that new approaches to communication services, protocols, and network architecture are required both to provide high-level abstractions for common flow types and to support user-level management of flow creation and quality. We describe experiences with the development of such applications and communication services.*

Keywords: Network applications, communication libraries, Nexus, Collaboratory Interoperability Framework (CIF)

1 Introduction

Advanced network applications such as remote instrument control, collaborative environments, and remote I/O are distinguished from "traditional" networked applications such as videoconferencing by their need to maintain multiple, heterogeneous flows with different characteristics. For example, a single application may require remote I/O for raw datasets, shared controls for a collaborative analysis system, streaming video for image rendering data, and audio for collaboration. Furthermore, each flow can have different requirements in terms of reliability, network quality of service, security, and so on. For example, in a tele-immersive collaborative environment, tracking information need not be propagated reliably but can often benefit from multicast, while database updates require reliable communication but cannot always use multicast capabilities. Mechanisms are required that allow both convenient specification of these applications and efficient execution in a variety of environments.

Historically, such applications either have used a single low-level communication protocol for all flows (e.g., TCP/IP [1, 2, 3]) or have used a mixture of different, often specialized APIs for different flows [4, 5, 6]. Neither approach is ideal. The first approach leads to a protocol that is good for some purposes but less ideal for others; in the second, program complexity is increased and portability is hard to achieve. In both cases, a variety of issues relating to the coordination of multiple flows (e.g., synchronization of audio and video, prioritization of different flows) have typically not been addressed at all.

We believe that such applications require more sophisticated communication services with, ideally, the following characteristics:

- A uniform API allows both high-level specification of communication structure and independent specification of communication mechanisms.

- A variety of flow types and interaction models are supported, including various types of streaming data, shared controls, and database updates.
- Support is provided for automatic and user-managed manipulation of flow characteristics, such as privacy, integrity, compression, and network quality-of-service.
- Support is provided for the coordination and management of ensembles of flows, enabling, for example, programmer-controlled prioritization, synchronization, and aggregation of flows in various types of network.
- Integrated instrumentation allows user-level monitoring of flow quality and notification, for the purpose of adaptation, of violations in performance contracts.

Our views on these topics have been shaped by our experiences developing both advanced networked applications and communication libraries designed to support such applications. In this paper, we review these experiences, focusing on one particularly demanding application and three different communication libraries.

2 Motivating Example

We use a single example application to motivate some of the discussion that follows. Our chosen application is typical of an emerging class of so-called "Grid" applications that couple geographically distributed resources of various types to create virtual devices with unique capabilities [7]. In this case, the resources in question are a specialized scientific instrument, the Advanced Photon Source (APS) at Argonne National Laboratory, used to probe the interior structure of materials at very small scales using, in this case, a technique called computed microtomography (CMT); a supercomputer, used to reconstruct 3-D material densities from the sequence of 2-D raw data "slices" provided by the instrument; and a number of both high-end and low-end display devices, used to support collaborative analysis of the reconstructed data. These resources work in concert to enable quasi-real-time reconstruction and collaborative analysis of APS data, so that users at remote sites can be manipulating and discussing three-dimensional image data just minutes after data collection begins [8].

2.1 Visualization Capabilities

The visualization system uses a specialized graphics utility, the SGI Volumizer library, to produce high-quality, 3-D volume-rendered images of the dataset. As illustrated in Figures 1 and 2, this data can be displayed in multiple ways, depending on the capabilities of the user's visualization environment:

- **Virtual reality display:** CAVE or ImmersaDesk (Idesk) immersive display devices support high-quality, 3-D stereo display. The user can control the display through the use of a control panel provided within the virtual environment. This control panel allows operations such as rotation, volume cropping, and assignment of color and opacity to dataset voxels.
- **Desktop display:** Rather than rewriting the volume renderer to run on less capable desktop hardware, we use remotely rendered video for the desktop display. The high-quality images produced by the SGI-based volume rendering hardware and software are captured, compressed, and sent over the network for display using standard network video display tools. A Java control panel supports desktop control of the rendering process.
- **Web display:** The same software can also capture individual images from the scene and put them on a Web page, for a very low-end solution that provides high-quality images.

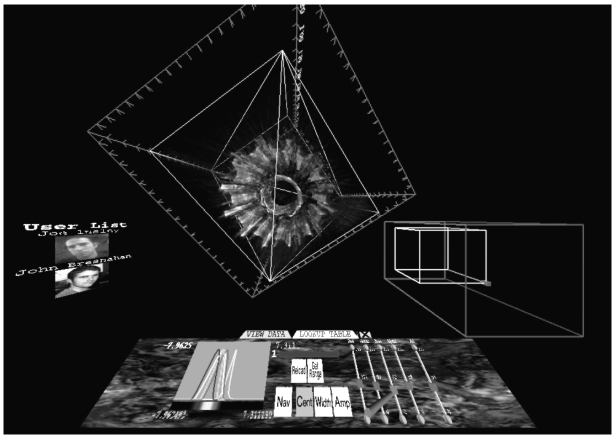

Figure 1: *A screen shot of the ImmersaDesk taken during a collaborative session with two users.*

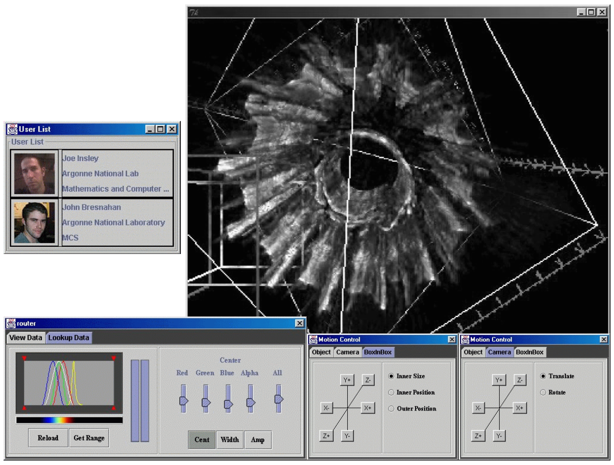

Figure 2: *A screen shot of a low-resolution graphics workstation taken during a collaborative session with two users.*

Shared-state mechanisms are used to link the virtual reality and desktop displays, so that users at different locations and on different systems can cooperate in the steering of the volume-rendering process.

2.2 Communication Requirements

The application is typical of advanced Grid applications in its simultaneous use of many underlying communication structures:

- The transfer of 2-D images from the APS to the supercomputer, and of 3-D datasets from the supercomputer to the visualization system, requires high-bandwidth (10s of Mb/s today, Gb/s or more in the future), unicast communication.
- Communication within the parallel 3-D reconstruction program requires high bandwidth and low-latency communication, as is typically available on parallel supercomputers through the Message Passing Interface (MPI) or shared-memory libraries.
- The video stream uses standard, unreliable IP multicast protocols (e.g., RTP and RTCP). We commonly used 800x600 pixel H.261 video, which requires approximately 300 kbps of bandwidth when the image is rapidly changing; significantly higher resolution is desirable.
- Audio streams between the collaborators can also use standard, unreliable IP multicast protocols. Audio requires less bandwidth than video but is more susceptible to quality degradation due to lost packets.
- The communication between the control panels of the collaborators uses both reliable and unreliable multicast protocols. Unreliable protocols can be used for incremental updates of the panels, for example while a user is dragging a slider on the panel. Reliable protocols are used to ensure that all participants are synchronized, for example when the user releases the slider on the panel to set a final value.

Hence, even in this relatively simple application we see a need for tens of flows (if multiple collaborators are participating) with widely varying characteristics. Other applications can place yet more complex demands on a communications infrastructure. For example, DeFanti and Stevens identify nine flow types in collaborative design applications [9].

3 Nexus

The preceding section outlines the wide variety of communication modalities that must be simultaneously supported in an advanced network application such as the CMT collaborative analysis and visualization system. In general, we observe that *the low-level method used to achieve a communication can vary according to where communication is being performed, what is being communicated, or when communication is performed* [10].

Currently, developers of such applications must program to a variety of APIs for these various flows (e.g., TCP sockets, IP multicast, reliable multicast libraries, MPI) and must know myriad details about each API in order to achieve good performance (e.g., TCP socket buffer sizes). This burden will only increase as these applications add such features as security and network quality of service.

We believe that the solution to this problem is to allow for the separate specification of the communication structure of an application and the methods used to achieve that communication. The Nexus communication library [11, 10] represents both an ambitious experiment in this regard and a substantial software system that has been used in many tool development and application projects, ranging from parallel language compilers to high-level communication libraries and distributed performance profiling systems. Nexus also serves as the communication component of the Globus toolkit.

Nexus provides simple, general ways for expressing communication, based on the abstractions of startpoints, endpoints, communication links, and remote service requests. These abstractions are able to express the wide variety of communication modalities described above. The Nexus implementation maps these abstractions onto a wide variety of underlying communication methods.

Nexus programs bind communication startpoints and endpoints to form communication links. If multiple startpoints are bound to an endpoint, incoming communications are interleaved, in the same manner as messages sent to the same node in a message passing system. If a startpoint is bound to multiple endpoints, communication results in a multicast operation. A startpoint can be copied between processors, causing new communication links to be created that mirror the links associated with the original startpoint. Hence, startpoints can be used as global names for objects that can be communicated and used anywhere in a distributed system.

A communication link supports a single communication operation: an asynchronous *remote service request* (RSR). An RSR is applied to a startpoint by providing a procedure name and a data buffer. For each endpoint linked to the startpoint, the RSR transfers the data buffer to the address space in which the endpoint is located and remotely invokes the specified procedure, passing the endpoint and the data buffer as arguments. A local address can be associated with an endpoint, in which case startpoints associated with the endpoint can be thought of as "global pointers" to that address.

An advantage of the startpoint construct in a distributed computing environment is that the startpoint can be used to encapsulate not only information about *where* communication should be performed, but also *how* to communicate. Different communication methods can be associated with different communication links, with selection being either automatic or user guided. The communication methods currently supported by Nexus are listed in Table 1.

In addition, a message transform, or filter, can be applied to each communication link. This feature allows operations such as compression, encryption, and profiling to be specified and performed on a per-link basis.

Our experience is that the Nexus abstractions capture nicely numerous communication structures and map cleanly onto a variety of underlying protocols and capabilities (e.g., security and quality of service). The one limitation of which we are aware relates to support for multicast communication. The Nexus API for creating startpoints and endpoints is currently better suited for the creation of unicast

Table 1: Communication methods supported by Nexus

Name	Description
Local	Reliable ordered unicast within a single process
SysV	Reliable ordered unicast between processes on the same computer, via System V shared memory
MPI/MPL/INX	Reliable ordered unicast between processes on different nodes of a single distributed-memory computer, via low-level communication libraries
TCP	Reliable ordered unicast
UDP	Unreliable, unordered or ordered unicast
IP multicast	Unreliable, unordered or ordered multicast
XTP	Reliable, source-ordered multicast
Totem	Reliable, totally ordered multicast

communication than for multicast communication. In particular, there is currently no way to directly bind a startpoint to multicast group. Instead, one must first create an endpoint that is bound to the multicast group, and then bind a startpoint to that endpoint. This can be annoying for processes that only want to send to a particular multicast group. This problem can be corrected by adding the communication link management to the API and then allowing startpoints and endpoints to directly bind to the communication link. Therefore, multicast communication would be set up by creating a communication link with multicast properties and by binding one or more startpoints and endpoints to that communication link.

4 CIF Comm Library

While Nexus demonstrates that a uniform interface can be constructed for a variety of protocols and messaging libraries, this interface (which was originally designed for use by compilers) is too low level for all but the most expert programmer. Hence, in a more recent project we have developed a higher-level interface that makes the same protocols available in a more convenient form. This interface, developed as part of the DOE2000 Collaboratory Interoperability Framework (CIF) project, is termed CIF Comm.

The CIF Comm design employs object-oriented concepts as a means of encapsulating protocol details. The interface consists of three core classes: abstract connection and listener classes, and a factory class to instantiate them.

The abstract connection class provides a simple interface for sending and receiving messages. It is from this class that all protocol-specific connection classes are derived. As the class name and capabilities imply, each of the protocol-specific implementations provide a connection-oriented, message-passing style view of the communication irrespective of the underlying protocol. Hence, applications can switch between different protocols simply by instantiating a different class.

The abstract listener class allows traditional client-server applications to implement server-side functionality using CIF Comm. Once a class has been instantiated, the listener waits for connection requests from remote connection objects. These connection requests are transformed into local connection objects when the application requests the next incoming connection from the listener.

In reality, an application never instantiates a protocol-specific connection or listener class. Instead, it makes a request to the factory class, which performs the instantiation on its behalf. To facilitate protocol independence in the factory, all requests are made using URLs in which the first component specifies the protocol to be used. This protocol information is used to instantiate the correct connection or listener class, which is then passed the remainder of the URL.

At present, both C++ and Java bindings have been implemented for the CIF Comm interface, supporting TCP, UDP, IP multicast, and Totem. In addition, XTP is supported in the C++ implementation and will soon be supported in Java as well. With these protocols, the application has the full cross product of reliable/unreliable and unicast/multicast communication available to it.

To date, CIF Comm has been used in two applications: a multi-user camera controller system developed by Deb Agarwal at Lawrence Berkeley National Laboratory and the CIF Shared State library (described below), a fundamental piece of the CMT application.

5 CIF Shared State Library

Collaborative applications require mechanisms for maintaining and synchronizing updates to the shared data elements that represents the state of the world in which collaboration occurs. For example, in the CMT data analysis system this shared state includes the various controls for the remote visualization system: point of view, color map, and so forth. We have used CIF Comm to implement a shared-state abstraction library, CIF Shared State, which was then used to implement the CMT collaborative data analysis system.

The Shared State component of CIF allows for shared control of abstract states in collaborative space across multiple platforms. An initial impetus for the creation of the shared-state library was to allow for shared control of "widgets" across different computer architectures and languages. (Other systems, in particular NCSA's Habanero, support a shared-state abstraction, but only within a Java framework.) If shared control of sliders, buttons, and other arbitrary components could be established, a graphical program running on a high-end resource could be controlled remotely from a simpler, more accessible computer. The CMT collaborative visualization application uses the CIF Shared State Library to do just that.

The CIF Shared State library is an object-oriented API with both C++ and Java implementations that allows for shared control of abstract states. (A Java implementation is provided for portability and a C++ implementation for use on high-end platforms and in C-based applications; a common Nexus-based wire protocol allows for interoperability.) The abstract states can be implemented as GUI components (sliders, buttons, toggles) or more simply as arrays of data primitives (integers, floating point numbers, bytes). To create a shared state, the user needs only to provide a mechanism for packing and unpacking its current values into a CIF Shared State "Serial" object via convenient methods provided by the API.

The use of shared-state information rather than collective control functions as our basic primitive proved extremely effective in the CMT application. We were able to create unorthodox visual components that provided no control to the user but were used to display useful information, such as histogram graphs, color bandwidth filter curves, and images of all of the users currently participating in the collaborative session. This layer of abstraction between shared data and visual control also allowed us to couple different visual component packages with the messaging structure: a Java-based control GUI for desktop clients and a set of 3-D widgets for use in the CAVE.

6 Conclusions

Emerging networked applications involve multiple flows with different and time-varying requirements for low-level protocols, security, performance, and so on. We have argued that the communication services that we provide to support these applications need to recognize this fact and provide explicit support both for the separate specification of communication flow and communication method and for the management of ensembles of flows in an integrated fashion. We have described three software systems that we have developed to address the first of these concerns, namely, the Nexus communication library and the CIF Comm and CIF Shared State libraries. Application experiences with these systems indicate that the separate specification of communications structure and method is indeed desirable. In future work, we will address the association of quality-of-service attributes with flows and the management of flow ensembles.

Acknowledgments

We gratefully acknowledge the many colleagues who have contributed to the development of Nexus, the CIF libraries, and the CMT application, in particular Gregor von Laszewski and Steve Wang at Argonne; Carl Kesselman and Mei Su at USC/ISI; Deb Agarwal at LBNL; and Bruce Mah at SNL/CA. 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. DOE, under Contract W-31-109-Eng-38; by DARPA under contract N66001-96-C-8523; and by NSF.

References

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

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

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

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

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

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

[7] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[8] G. von Laszewski, I. Foster, J. Insley, J. Bresnahan, C. Kesselman M. Su, M. Thiebaux, M. Rivers, I. McNulty, B. Tieman, and S. Wang. Real-time analysis, visualization, and steering of microtomography experiments at photon sources. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, 1999.

[9] T. DeFanti and R. Stevens. Teleimmersion. In *[7]*, pages 131–156.

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