To Appear in Proceedings of the SPDP'96 Workshop on Program Visualization and Instrumentation – October 1996

From Interactive High Performance Programs to Distributed Laboratories: A Research Agenda

Greg Eisenhauer* Beth Schroeder Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

1 Interactive Computational Instruments

Many scientific problems require highly complex and difficult computations. If the individuals who study these problems can interact with the computational tools while the computations are progressing, the analysis of the problem could be much more efficient and effective. This article introduces the potential increases in functionality and performance gained by the online interaction of end users with high performance *computational instruments* on single and on networked parallel machines (also see [17]). We consider systems in which users interact with computations as if they were physically accessible laboratory instruments and in which entire *distributed laboratories* are constructed from sets of such computational instruments. Within this context, our intent is to facilitate both online interactions with single computational instruments and interactions among multiple scientists and multiple instruments located at physically distributed sites where scientists may have dissimilar areas of expertise.

With our research and with the larger-scale *Distributed Laboratories* project at the Georgia Institute of Technology[7], we aim to improve the state of the art of interactive high performance computing for parallel and distributed applications on the variety of heterogeneous platforms now in common use by HPC users and researchers. Particularly, our goal is to develop a general framework for enhancing the interactivity of high performance applications. This framework:

- contains general interactivity (i.e., online monitoring and steering) mechanisms with which high performance applications may be inspected and/or steered at runtime by algorithms, human users, or both;
- supports interchangeable visualizations run across heterogeneous and distributed hardware platforms, using a robust and portable data meta-format for transporting visualization content;
- supports the simultaneous interaction of multiple scientists with single large-scale computations; and
- permits collaboration via multiple computational instruments among scientists working in separate locations.

In this article we introduce one of the parallel and distributed scientific applications used in our research. We then explore interactions with a single computational instrument during its execution, typically referred to as *interactive program steering*†. Finally we discuss the use of multiple computational instruments by sets of end users, thereby moving from issues addressed in previous work toward interesting topics for future research.

2 A Sample Application

In order to more clearly illustrate the challenges posed by distributed laboratory environments, we consider a specific application. In particular, consider a large scientific simulation running on a set of computational resources, such as the global climate model being developed for networked parallel machines in our research[11]. This model uses assimilated windfields (derived from satellite observational data) for its

*Email addresses for the authors are: eisen@cc.gatech.edu, beths@cc.gatech.edu, and schwan@cc.gatech.edu

†Interactive steering may be defined as the online configuration of a program by algorithms or by human users, where the purpose of such configuration is to affect the program's execution behavior.

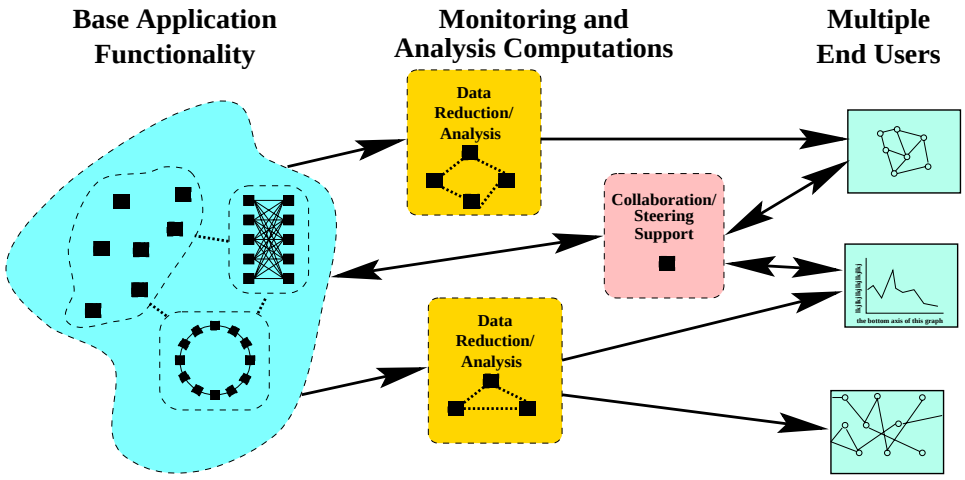

Figure 1: Computational elements and communications in the distributed laboratory.

transport calculations, and known chemical concentrations also derived from observational data for its chemistry calculations. Models like these are important tools for answering scientific questions concerning the stratospheric-tropospheric exchange mechanism or the distribution of such atmospheric agents as chlorofluorocarbons (CFCs), hydrochlorofluorocarbons (HCFCs) and ozone. This global model might interact with more detailed local climate, atmospheric or pollution models[13] in order to enhance the accuracy of both the local and global models. These models may also run concurrently on a variety of parallel and distributed computing resources, or ideally, any one of the global or local models should be replaceable at runtime with a historical database containing information from previous model runs. In addition, model outputs may be processed with a variety of computational instruments, such as tools which perform time series analysis, trajectory analysis and residual circulation analysis. Similarly, model inputs may be provided via other 'instruments' that either utilize live satellite feeds[1] or access stored satellite data on remote machines (e.g., at NASA storage sites).

Into this application scenario we introduce observers. Given the scale of such an application and the speed and complexity of its execution platform, it is easy to imagine that the simulation's progress and results are monitored by multiple scientists in distributed locations. The scientists' interests may vary from wishing to see the "big picture," to investigating in detail subsets of the simulation's output, to collaborating with each other via the computational instruments these models implement. Toward this end, the distributed application has additional components that assemble the information needed to drive various interactive displays, by gathering data from the distributed simulation and performing the analyses and reductions required for these displays. Some of these components may themselves have substantial computational or storage needs and require dedicated, additional resources of their own. In addition, they may require access to additional information, as is the case for the atmospheric model's display with which end users compare observational (satellite) data with model outputs in order to assess model validity or fidelity. Moreover, since end users control the set of computational instruments, input and output components, and the displays, the current set of interacting 'computational instruments' will change dynamically, driven by end users' needs or by the current needs of the running simulation. Figure 1 depicts computational elements and communications which might exist as multiple users examined and interacted with an application in the distributed laboratory environment.

The previous paragraph establishes the dynamic nature of computational instruments and their connections, which may be further amplified by these applications' need to react to changes in underlying

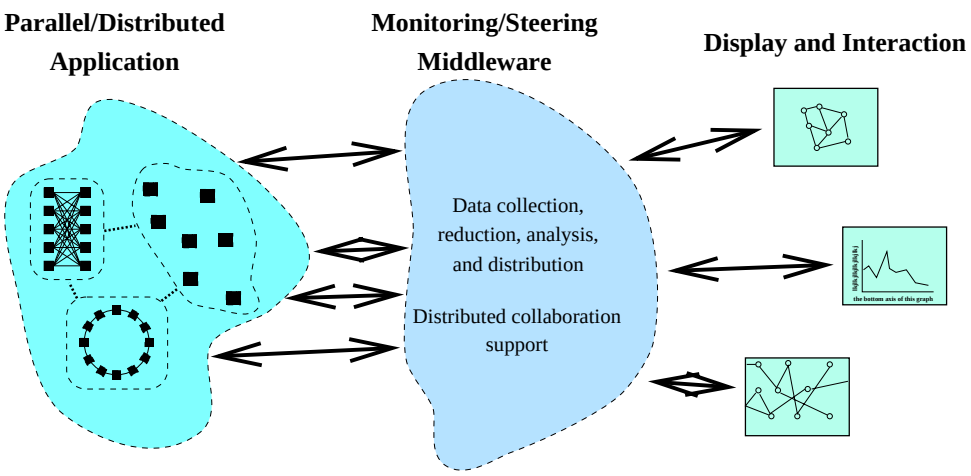

Figure 2: General Interactivity Framework

computing platforms (e.g., changes in available computational or communication capacity in the underlying hardware resources). While it is clear that end users frequently interact with displays and with the analysis components used prior to display, it has become more common for users to also interact with the actual simulations while they are running, perhaps to *steer* them toward more appropriate data domains, to enable or disable certain ancillary computations, or to experiment with alternative settings for simulation parameters. The benefits of such dynamic interactions of end users with large-scale simulations are well known for discrete event simulations (e.g., battlefield management[14]) and have recently been found to offer substantial benefits to scientific and engineering users, as well. In addition, it is well known that significant performance improvements may be derived for high performance applications from the runtime configuration of selected application attributes[4], from the online adjustment of their execution environment (e.g., process migration[2]), and from the adaptation of underlying operating system or communication facilities[12].

Taken as a whole, this application scenario has requirements for instrumentation, monitoring, visualization, and general data management and exchange that go well beyond those of a traditional parallel or distributed application. The remainder of this paper will explore those requirements.

3 Interactivity Framework

Figure 2 depicts a general interactivity framework for a complex high performance application. The Distributed Laboratories project includes support for all aspects of such applications, including visualizations, steering interfaces, data communication and analysis middleware, collaboration support, and application monitoring and steering support. Before discussing the general issues in distributed collaborative applications we focus more narrowly on the framework required to support a single user interacting effectively with a single computational instrument. Specifically, we consider:

- the support required to efficiently monitor system- and application-level behavior in parallel and distributed programs;
- frameworks and support for steering such a program though external interfaces; and
- visualization interfaces for both monitoring and steering.

3.1 Monitoring

Interacting with computational instruments in terms meaningful to end users requires online monitoring – the dynamic gathering of application-specific information from an instrument as it executes. Falcon [9] is a set of tools and libraries supporting online, application-specific, event-based monitoring of parallel and distributed applications. Falcon consists of a sensor specification language and compiler for generating application sensors, and one or more local agents for online information capture, collection, filtering and

analysis of event data. In addition, Falcon uses intermediate monitoring/steering middleware to disseminate monitoring events to the potentially large number of clients that wish to interact.

Falcon's implementation is based on a local agent, usually residing on the monitored program's machine, to capture event data. On shared memory architectures, this local agent may be an additional thread operating in the instrument's address space. Local agents, due to their proximity to the computational instrument, can gather monitoring data quickly while minimizing interference with the instrument's operation. For physically distributed instruments, multiple local agents are employed and each of the instrument components is treated as a separately monitored entity.

3.2 Interactive Steering

A steering system used by external agents must have the following functionality: (1) receiving the computational instrument's runtime information from the online monitoring system, (2) analyzing and then displaying the information to the end user or submitting it to a steering agent, (3) accepting steering commands from the user or agent, and (4) enacting those commands to affect the application's execution. In Falcon's implementation of computational steering, at least one local agent runs on the target machine with the application. This local steering agent performs any steering actions requested by external agents. External agents are driven by monitoring data and may request steering actions directly based on this data or they may support graphical interfaces interfaces and request steering actions in response to user manipulations.

One insight from our experience with interactive steering is that computational instruments differ in terms of the ease with which certain steering actions may be implemented. An instrument's implementation may permit some of its internal variables to be inspected and steered continuously, with little additional instrumentation of the code. In general, however, potential improvements in performance or functionality by the addition of steering depend largely on an instrument's implementation and on the steering and monitoring actions required.

Another insight from our experiences is that program steering must consider overheads not relevant to performance monitoring, which tends to focus on the effects of program perturbation. These overheads are: (1) the perturbation to the application due to instrumentation for monitoring and steering, (2) the latency of the monitoring to enactment feedback loop, and (3) the costs of decision making part of this latency. Specifically, for steering, the end-to-end latency of the monitoring to enactment feedback loop is a critical performance constraint when steering actions cannot be based on 'stale' monitoring data, or when such actions become inappropriate after some future program state has been passed.

3.3 Application-specific Visualization/Steering Interfaces

While monitoring and steering support are essential for efficient, low-impact interactivity, it is often the final visual interface that determines the success and effectiveness of interactivity. The interactive 3D visualization of atmospheric modeling data shown in Figure 3 is constructed and used with a set of modules from the GlyphMaker display and steering system [16] (originally constructed with the SGI Explorer environment, and now being supported in the OpenInventor framework). These modules implement the functionality required by certain application-specific displays, in this case modules that convert monitoring data extracted from the computational instrument from its spectral form to a gridded form more suitable for visualization, and a module acting as a reader for converting the data being displayed to be printed on high the high resolution output devices used by atmospheric scientists (using the PV-Wave visualization system).

Using the 3D visualization shown in Figure 3, the atmospheric model is steered by first positioning the latitudinal and longitudinal planes, sizing and moving a rectangle to intersect a plane, then entering a specific desired concentration increment/decrement. The resulting set of new concentration values is forwarded from the visualization interface to the computational instrument via the aforementioned monitoring/steering infrastructure. The new concentration values are used as part of the next timestep taken by the model to result (hopefully) in improved model behavior.

In contrast to the 3D visualizations providing excellent overviews of model behavior, a complementary 2D steering interface operating with subsets of the atmospheric modeling application's data is shown in Figure 4. This interface's display presents the distribution of C^{14} at the single latitude of 2.8° N. It has two logical parts: one for showing both the computed and the observed concentration values of C^{14} atoms in air to the end user, and the other for accepting steering requests from the user. The computed results of the C^{14} distribution are represented by the blue plotted curve from atmospheric layer 0 to 37, which is updated

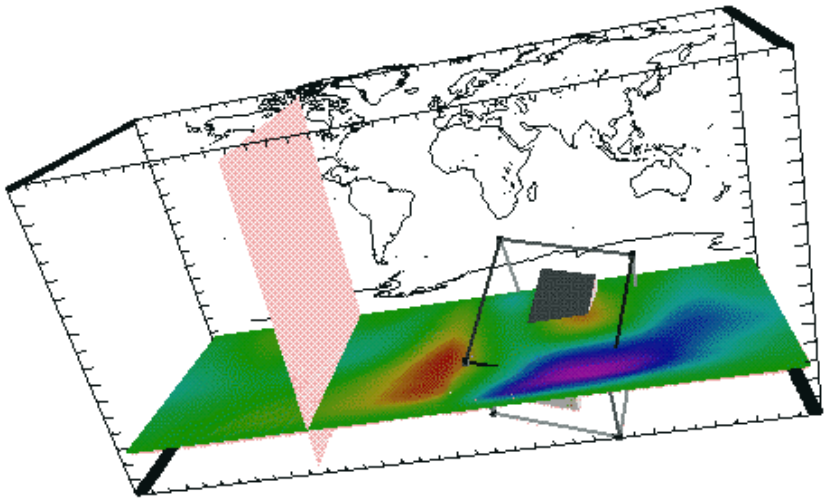

Figure 3: Three-dimensional visualization of atmospheric modeling data.

for every model time step. The concentration of C^{14} actually observed at this point is represented by the red plot. When noticeable discrepancies between the calculated values and the observed values are detected, the user can dynamically modify the application execution to 'correct' the computations.

4 Toward Distributed Laboratories

Distributed laboratories are environments where scientists and engineers working in geographically separated locations share access to interactive visualization tools and large-scale simulation computations, share information generated by such instruments, and collaborate across time and space to evaluate and discuss their results. The intent is to permit scientists, engineers, and managers at geographically distinct locations (including individuals telecommuting from home) to combine their expertise in solving shared problems by allowing them to simultaneously view, interact with, and steer sophisticated computation instruments executing on high performance distributed platforms.

The goals of the Distributed Laboratories project are to develop the underlying, enabling technologies and software tools to support the distributed laboratories vision and to develop working prototypes for use in research projects in high performance scientific computations such as the atmospheric science model and virtual telecommunication networks.

Distributed laboratories have several essential and novel characteristics which stand in contrast to facilities like the World Wide Web used mostly for information access and browsing:

- The information generated in a distributed laboratory must not only be accessible to all users, but users must also be given interactive interfaces for the online control of laboratory instruments and for control of information generation, analysis, storage, and sharing between sites and physical and virtual instruments.
- Interactive laboratory access should be provided with little regard to physical locations of experimenters and instruments, including home locations with limited computational and storage capabilities, access bandwidths, and increased access latencies.
- In contrast with current collaboration environments (e.g., supporting direct remote audio and video), a shared laboratory must also support communication arising through shared manipulation of visualizations, interactive experimentation with running simulations, and remote control of instruments.

The underlying, enabling technologies needed to support distributed laboratories research include:

- dynamic monitoring, adaption, and interactive steering of high performance computations;
- interconnectivity and data exchange infrastructure; and
- collaboration and shared visualization technologies.

Our work in program monitoring, adaption and interactive steering is documented elsewhere[9, 10]. Work

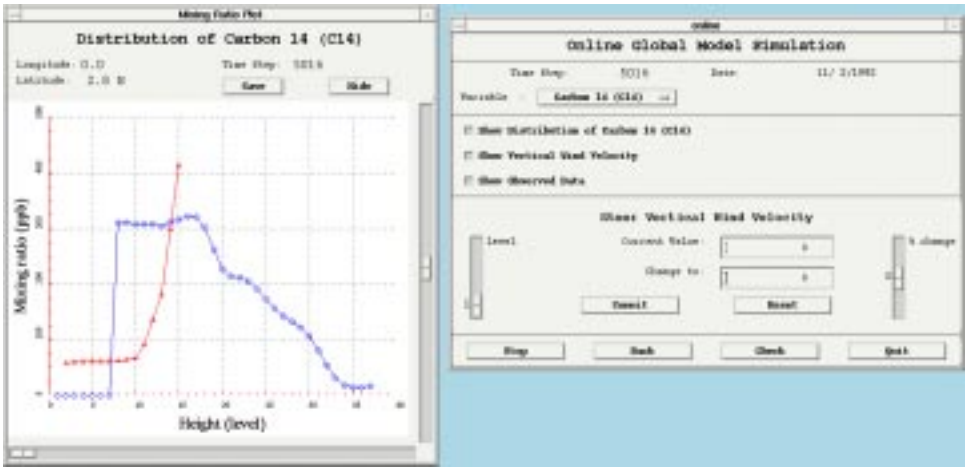

Figure 4: An application-specific display for online control of the atmospheric modeling code.

on collaboration and shared visualizations is in progress at Georgia Tech and in a variety of other research groups. However, the appropriate communication infrastructure to support this type of research is less well studied. Our work on a novel communications and data analysis middleware facilitating online monitoring and interactive steering in a multi-scientist/multi-instrument environment is discussed in the next sections.

4.1 Communication Requirements of Distributed Laboratory Applications

The applications and environment described in the previous sections yield some specific requirements which a communications infrastructure for the distributed laboratory must satisfy.

High-bandwidth Flexible Data Transfer

The need for the fastest possible transfer in the distributed environment is common to a wide range of distributed applications and those in the distributed laboratory will be no exception. Besides the normal demands of the running distributed application, the speed of the data transfer driving the observers' displays is critical for the performance of the overall system. While few communications systems have the luxury of imposing significant overheads, the important role of communications in the overall performance and usability of a distributed laboratory imposes additional constraints. For example, the distributed laboratory communication scheme cannot rely on inefficient techniques, such as ASCII transmission of data, to handle such issues as heterogeneity in the execution environment. Also, the communications system must provide mechanisms for the number of observers of a computation to scale without significantly affecting the execution of that computation.

Operation in a Diverse Environment

The application environment described in Section 2 above is clearly heterogeneous, but it can perhaps be more strongly described as *diverse*. By this we mean that in addition to operating in an environment where machine-level representations of elementary types may differ, the various components of the application may be developed in a much less tightly coupled manner than a traditional parallel or distributed program. That is, they may not all be produced by a single group or organization and they may include "generic" components designed to operate in many situations.

One implication of this situation is that it is not generally practical for all of the components operating in the distributed laboratory to agree at compile-time on the formats of all data records to be exchanged during execution. Even if agreeing on a common and consistent format for the data were possible, systems depending upon such agreement are very fragile and unsuitable for a dynamic environment. This is a problem that has not been addressed in the past by high-performance communication systems. The most obvious approach to operating in an environment this diverse is to transmit data in some

easily-interpreted free-form manner, such as encoded in ASCII, and to parse it upon receipt. This approach achieves the required flexibility, but at the cost of seriously compromising the efficiency of the communication system, and so it is not an option for a distributed laboratory.

Dynamic Connection of Clients

In the distributed laboratory environment computational agents may be very long-running and physical instruments may always have data available. In this environment it is natural to expect displays, their associated data processing agents and other components of the execution environment to come and go as computations proceed. To accomplish this, external clients must be able to locate and connect to data sources dynamically. This functionality tends to be lacking in communications tools created with relatively static topologies in mind (such as MPI [6]). Even those which have dynamic client creation may not support external connections or have adequate client/server location facilities (such as PVM [8]).

Data Flow Management

The potential for dynamic connection of clients means that the flow of data in the distributed laboratory is itself very dynamic. In particular, it implies that senders may not have an a priori knowledge of the identity of all possible receivers. This is a problem for communications systems which require destinations to be explicitly specified by the sender, such as traditional send/receive and RPC-style communication mechanisms.

Information Filtering and Analysis

While the demands of the distributed laboratory environment eliminate some possible solutions, the richness of the environment and the dynamic nature of traffic create additional opportunities. For example, a capable communications system may take advantage of external knowledge of the application and reduce overall communication requirements through the use of filtering, analysis and compression.

4.2 DataExchange and PBIO

We have developed a communications infrastructure specifically designed to the needs of the distributed laboratory environment described above. The infrastructure is composed of two cooperating libraries, DataExchange[5] and PBIO (Portable Binary Input Output)[3]. Their relevant characteristics are described below.

4.2.1 PBIO

PBIO is the lower layer of the distributed laboratory communications infrastructure and provides basic support for handling heterogeneity and operation in a diverse environment. Essentially, PBIO is a data meta-representation. Users register the structure of the data that they wish to transmit/store or receive/read and PBIO transparently masks the differences. In particular, PBIO handles differences in the sizes, locations and even basic types of the fields in the records to be exchanged. Field matching is by field names provided by the application. Records are C-style structures consisting of fields which can be any of the usual atomic data types or NULL-terminated strings. A previously registered structure may also serve as the basic type of a field, allowing the creation of complex nested record types. PBIO also supports fields which are statically sized one or two dimensional arrays or dynamically sized one dimensional arrays of any of the above datatypes. Record meta-information is transmitted once, when record formats are registered. Thereafter, transmission occurs in the writer's native format and the PBIO library on the receiver transparently handles discrepancies between the writer's format and the format required by the reader. In the case of transfers between homogeneous machines, the only additional overhead imposed by PBIO is the transmission of a 4-byte format ID. Between heterogeneous machines, the extent of the overhead depends upon the degree to which the record formats and atomic type representations differ between the two machines.

PBIO's data meta-representation and the support for type matching by application-level field names, rather than by field layout or ordering, facilitates the creation of applications which can operate in the diverse environment of the distributed laboratory. In particular, PBIO allows applications with no compiled-in knowledge of the record formats involved to locate information based on record and field names. This is of

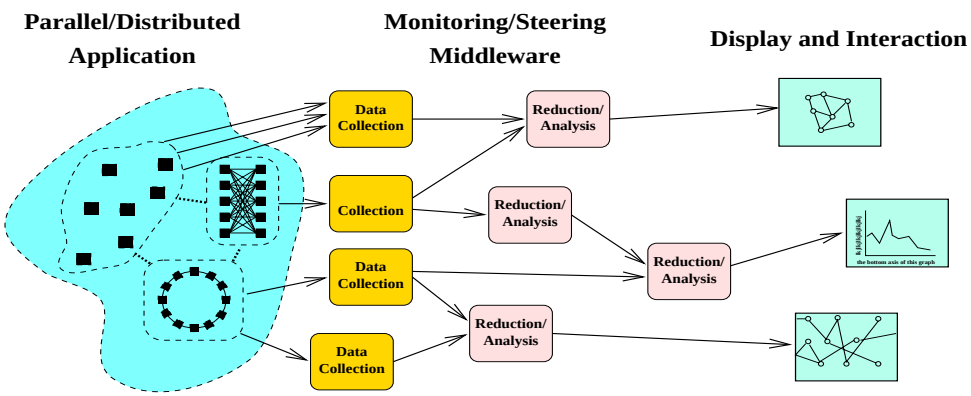

Figure 5: Application, data processing agents and visualizations in the distributed laboratory.

great value in constructing generic visualization and data analysis and reduction tools which can be employed in a variety of roles in the distributed laboratory. Similarly, PBIO has extensive support for applications which are only interested in examining or operating on specific fields in records to the exclusion of other fields. Through these facilities, PBIO application can be robust to changes that need not affect them, such as the addition of new fields or reorganization of existing fields in a record. This is a significant advantage in the distributed laboratory where a widely-used fixed record format might otherwise be a significant obstacle to the evolution of an application.

For efficiency and speed, PBIO takes care to avoid unnecessary overhead, such as multiple data copies or conversion of all data to a standard byte-order. While PBIO is most commonly used across TCP/IP sockets, PBIO also supports storing data in files and the use of alternative communications protocols such as UDP broadcast. This too supports the need for fast and flexible data transfer because in appropriate situations, such as when communicating to large number of local clients and when reliability is not a concern, the use of alternative communication transport layers can be a significant performance advantage.

4.2.2 DataExchange

While PBIO supports efficiently and robustly exchanging data between two clients, the DataExchange library[5], which is layered on top of PBIO, addresses the remainder of the needs outlined in Section 4.1. In particular, DataExchange provides support for establishing communication between agents, resolving differences between data formats used by multiple agents, forwarding data from agent to agent, and processing data within an agent.

In order to address dynamic client connection and data flow management, DataExchange provides support for a type of generative, or publish/subscribe, communications model within the distributed laboratory. This stands in contrast to the send/receive or RPC-style communication styles more traditionally provided by libraries designed for use with parallel applications. However, other researchers are advocating the use of generative models for the similarly-dynamic World Wide Web environment [18] and as technique to make systems more robust to changes in their environment[15].

In understanding DataExchange behavior, it is useful to think of DataExchange as analogous to the hardware that makes up a telephone exchange. Like a node in the telephone network, each DataExchange instance provides services and behavior that contributes to the collective behavior of a set of connected instances. The DataExchange library provides a simple name service to assist in connecting independent DataExchange instances into a functional network. Once the instances are connected, the library provides facilities for connected instances to select the types of messages that they receive, to block messages they are not interested in, and to control the forwarding of messages to any other instances to which it is connected. To support data processing and analysis, DataExchange also allows application handler functions to be bound to message events.

This functionality makes it easy to use the DataExchange library to construct networks of cooperating agents such as the one depicted in Figure 5. Data sources, such as the local monitoring agents in Falcon or persistent instruments in the Distributed Laboratory, register their availability with the DataExchange name server. Principal data consumers, such as data collection and collection agents and end-user displays connect to existing data sources and the non-display agents that forward data may then register themselves as data sources. This establishes the main application→analysis→display data flow in the distributed laboratory. Program steering, however, entails sending steering commands from the end-user displays to the application. While this communication is presumably more of a one-to-one nature and less suitable for a generative communications model it can be handled with the same functionality that creates the application→display dataflow. Future version of DataExchange may include more direct support for one-to-one communication.

5 Conclusions

The Distributed Laboratories project is constructing an infrastructure with which future scientists and engineers can interact with each other and with their computational instruments as if they were physically co-located in a single laboratory. Several research efforts being undertaken at Georgia Tech address the topic of interactive high performance programs:

- The Falcon steering and monitoring tools and infrastructure are used in the online observation and manipulation of scientific computations.
- The visualization support provided by the GlyphMaker tool permits the definition of appropriate visual abstractions and their efficient representation on 3D graphical displays.
- Collaboration infrastructure and abstractions are provided using the OpenInventor graphical display framework.

This paper describes those efforts, considers their communication demands and also describes the common communication infrastructure being developed to support their needs.

References

[1] National Aeronautics and Space Administration. Mission to Planet Earth. http://www.hq.nasa.gov/office/pao/NASA/mtpe.html.

[2] F. Douglis and J. Ousterhout. Process migration in the Sprite operating system. In *Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 18–25, Berlin, West Germany, September 1987. IEEE.

[3] Greg Eisenhauer. Portable self-describing binary data streams. Technical Report GIT-CC-94-45, College of Computing, Georgia Institute of Technology, 1994. *http://www.cc.gatech.edu/tech_reports* or *http://www.cc.gatech.edu/systems/projects/PBIO*.

[4] Greg Eisenhauer, Weiming Gu, Karsten Schwan, and Niru Mallavarupu. Falcon – toward interactive parallel programs: The on-line steering of a molecular dynamics application. In *Proceedings of The Third International Symposium on High-Performance Distributed Computing (HPDC-3)*. IEEE, IEEE Computer Society, August 1994. Also available as Georgia Institute of Technology Tech Report GIT-CC-94-08.

[5] Greg Eisenhauer and Beth Schroeder. The DataExchange library. Technical Report GIT-CC-96-17, Georgia Institute of Technology, 1996. *http://www.cc.gatech.edu/tech_reports* or *http://www.cc.gatech.edu/systems/projects/DataExchange*.

[6] MPI Forum. MPI: A Message-Passing Interface. Technical Report CS/E 94-013, Department of Computer Science, Oregon Graduate Institute, March 94.

[7] Richard Fujimoto, Karsten Schwan, Mustaq Ahamad, Scott Hudson, and John Limb. Distributed laboratories: A research proposal. Technical Report GIT-CC-96-13, Georgia Institute of Technology, 1996. http://www.cc.gatech.edu/tech_reports.

[8] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM 3 Users Guide and Reference manual.* Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, May 94.

[9] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, J. Vetter, and N. Mallavarupu. Falcon: On-line monitoring and steering of large-scale parallel programs. Report GIT-CC-94-21, College of Computing, Georgia Institute of Technology, 1994. http://www.cc.gatech.edu/tech_reports.

[10] Weiming Gu, Greg Eisenhauer, Eileen Kraemer, Karsten Schwan, John Stasko, Jeffrey Vetter, and Nirupama Mallavarupu. Falcon: On-line monitoring and steering of large-scale parallel programs. In *Proceedings of FRONTIERS'95*, February 1995. Also available as Technical Report GIT-CC-94-21, College of Computing, Georgia Institute of Technology.

[11] Thomas Kindler, Karsten Schwan, Dilma Silva, Mary Trauner, and Fred Alyea. Parallelization of spectral models for atmospheric transport processes. *Concurrency: Practice and Experience*, 1996.

[12] Robin Kravets, Ken Calvert, and Karsten Schwan. Dynamically configurable communication protocols and distributed applications: Motivation and experience. Technical Report GIT-CC-96-16, Georgia Institute of Technology, 1996. http://www.cc.gatech.edu/tech_reports.

[13] N. Kumar, M.T. Odman, and A.G. Russell. Multiscale air quality modeling: Application to southern california. *Journal of Geophys. Res.*, 99:5385–5397, 1994.

[14] Jim McDonald and Karsten Schwan. Ada dynamic load control mechanisms for distributed embedded battle management systems. In *First Workshop on Real-time Applications, New York*, pages 156–160. IEEE, May 1993.

[15] Brian Oki, Manfred PFluegl, Alex Siegel, and Dale Skeen. The information bus – an architecture for extensible distributed systems. In *Proceedings of the 14th Symposium on Operating System Principles.* Association of Computing Machinery, 1994.

[16] W. Ribarsky, E. Ayers, J. Eble, and S. Mukherjea. Glyphmaker: creating customized visualizations of complex data. *Computer*, 27(7):57–64, 1994.

[17] Beth Schroeder, Greg Eisenhauer, Karsten Schwan, Jeremy Heiner, Peter Highnam, Vernard Martin, and Jeffrey Vetter. From interactive applications to distributed laboratories. *submitted to IEEE Computational Science and Engineering*, June 1996.

[18] Mark D. Wood and Bradford B. Glade. Information servers: A scaleable communication paradigm for wide area networks and the information superhighway. In *Proceedings of the 7th SIGOPS European Workshop.* Association of Computing Machinery, 1996.