Realizing Distributed Computational Laboratories

Beth Plale Volker Elling Greg Eisenhauer
Karsten Schwan Davis King Vernard Martin

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
{beth,elling,eisen,schwan,kingd,vernard}@cc.gatech.edu

I. Introduction

The[1] Internet has created an opportunity for collaboration between scientists at unprecedented levels. Furthermore, in the late 80's, researchers developed general approaches to program monitoring[1], [2], [3], followed by advances in program adaptation[4] and steering[5]. Specifically, by coupling program monitoring and steering with online visualizations[6], [7] of scientific data, it has now become possible for scientists to interact with their simulations at runtime and in a variety of ways that do not depend on simulation designs or implementations. As a result, intermediate program values may be inspected at will, parameters may be changed 'on the fly', and data values may be selected to 'guide' applications into interesting data domains. Moreover, by extending the functionality of restart files, it has become possible to stop, rewind, and rerun applications when simulation output does not agree with observational data or when scientists deem ongoing runs uninteresting. In summary, such interactivity, also called 'program steering', removes the separation in time between the scientist and the computational tool being employed.

Program steering has the potential of improving end user productivity, yet its current realizations still confine multiple scientists to interact with their applications via single visualization clients. This situation is not satisfactory, especially considering the explosion in network bandwidth of the early 90s and the broadening diversity of resources with which a researcher can participate in scientific exploration, from high-end graphics workstations to home PCs with modem connections to the Internet. In response, research being conducted now aims to enable multiple scientists to interact with each other via shared complex scientific applications, from geographically distributed locations, using diverse computing and networking resources, and such that each scientist is able to interact with the application via operations and data suited to his/her expertise.

Recent directions in scientific computing have also been influenced by significant advances in compute power, making possible complex simulations that were heretofore unrealistic, including heterogeneous simulations simultaneously considering multiple, linked physical processes (e.g., atmospheric and oceanic modeling[8]) or simulations that consider multiple physical models at different time or length scales. For example, the atmospheric model discussed in Section IV couples a parallel spectral transport model with a grid-based chemical model. The environmental hydrology project [9] simulating the Chesapeake Bay integrates atmospheric models describing the physics of clouds and predicting rainfall, a river model predicting flow in stream channels, and a wind model describing surface flow patterns. Finally, our future work aims to couple a global atmospheric model of fairly low granularity with one or more regional models, perhaps attempting to understand the global effects of pollution in a certain metro area. Therefore, these models are not only heterogeneous, but they also operate at highly different scales (i.e., kilometers versus meters).

The opportunities in scientific collaboration presented by the Internet coupled with significant advances in compute power suggest the inadequacy of existing models of heterogeneous parallel computing, like PVM or MPI. This inadequacy has already been recognized by projects like Globus[10] for running large-scale, distributed scientific codes, interactively, across heterogeneous target systems. In addition, both the NCSA and PACI Supercomputer Centers in the U.S. are developing methods and tools for interactive use of large-scale data or simulations, and for scientific collaboration via both high end immersive systems and low end, browser-based interaction media.

Our work contributes to scientific computing by improving the scientists' freedom to interact online with their applications and with each other, from remote locations, their offices, and their laboratories, using interaction media suited to their locations and current needs. We call such research environments *Distributed Laboratories*; they are characterized by:

- multiple data sources,
- coupled models of varying levels of granularity,
- interactivity,
- multiple collaborating scientists, and

Invited paper in International Journal of Parallel and Distributed Systems and Networks, 1998
[1]Invited paper in International Journal of Parallel and Distributed Systems and Networks, 1998

- identifiable requirements of throughput and latency in interaction.

In this paper we discuss a software infrastructure that enables the construction and operation of distributed laboratories. We organize this discussion using sample computational laboratories (see Section 2) and a catalog of services (see Section III) to be provided by the infrastructure. Section III-B describes our novel solutions concerning these services, focusing on middleware components that are systematically introduced to the environment to increase the quality of service provided to end users. Since our work in distributed laboratories is based in part on the experiences gained working with atmospheric scientists at Georgia Tech in coupling two disparate atmospheric models, we next describe the integration results obtained from a recent port of the combined atmospheric model from a single shared memory multiprocessor to a heterogeneous set of parallel/distributed cluster machines. We conclude by discussing our current directions in managing distributed laboratory infrastructures.

II. Distributed Laboratory Hardware and Examples

The equipment infrastructure at Georgia Tech provides a testbed for the distributed laboratories project. With its heterogeneous compute servers, data storage facilities, and remote access facilities, Georgia Tech has an environment that on a smaller scale mirrors the larger testbeds such as GUSTO [10] being explored elsewhere. The equipment infrastructure shown in Figure 1 consists of several cluster resources, including a 16 quad-processor 200 MHz Pentium Pro cluster with both 100 MB Ethernet and Myrinet interconnects, a 48 node dual processor 300 MHz Pentium II cluster with 100 MB Ethernet, and a 16 node UltraSparc cluster also using both 100 MB Ethernet and Myrinet interconnects. These clusters are connected with a 622 MB ATM backbone and using Gigabit Ethernet. Computationally intensive applications may also use a 16-processor SGI Origin multiprocessor connected via Gigabit Ethernet to the clusters. Finally, all of these computational resources are connected to a second set of HPC engines via a campus-wide high performance ATM backbone. Through this backbone and its connection to vBNS, one is able to access remote supercomputing facilities, including those at NCSA and PACI.

It is clear from Figure 1 that this hardware infrastructure can support a distributed laboratory scenario where multiple simulations run on heterogeneous machines, and data originates from multiple sources including historical data from files. Furthermore, it is apparent that multiple users interacting with scientific applications will have differing scientific and computational needs and abilities. For example, one scientist may employ a high-end multimedia workstation to visualize data while a second user may interact via a home PC connected to the infrastructure using a broadband or a modem and telephone line. An important requirements of Distributed Laboratories is their responsiveness to this dynamic and heterogeneous group of end users.

The distributed laboratories project is testing and revising concepts using a number of diverse and interesting applications. One application motivating our work in the past is an atmospheric global transport model developed in collaboration with Earth and Atmospheric scientists at Georgia Tech. The global transport model simulates the transport of chemical compounds (called *species*) through the atmosphere. It uses assimilated windfields [11] (derived from satellite observational data) for its transport calculations, and known chemical concentrations also derived from observational data as the basis of its chemistry calculations. Our model contains 37 layers, which represent segments of the earth's atmosphere from the surface to approximately 50 km, with atmospheric species concentrations represented by superimposed waves with different weights. Details of the model's solution approach and parallelization are described in [12].

A visualization of N^2O concentrations in the atmosphere is shown in Figure 2. Species concentration information is extracted from the transport model using the Falcon [13] on-line monitoring and steering system. Depicted in the figure is a steering isosurface. To effect a steering change in the model, the user first defines the isosurface and then sets a new value for the area described. The new values are propagated back to the model using Falcon's steering mechanisms.

We are working with other applications as well. An environmental modeling project underway at Georgia Tech couples global chemical and climate models with regional air quality and climate change models. In this project, the importance of supporting different end users and their diverse needs at multiple locations becomes paramount. Experts in global modeling in Environmental Sciences at Georgia Tech desire to work jointly with local experts at pollution modeling in Civil Engineering and/or remotely at NOAA/EPA, where each is concerned with the models' operation at the different time and spatial scales of interest to them. Furthermore, this application displays dynamic end user and application/ancillary computation behaviors, the latter due to changes in resolution across time and spatial scale, the former due to changes in end users' needs concerning desired data fidelity, and due to the runtime addition and removal of end users, machines, and desired ancillary computations.

The Chesapeake Bay simulation [9] constitutes another multiple-model simulation with diverse user needs, ranging from fully immersive environments to web-enabled visualizations. In future work, we will be adding steering infrastructure between the hydrology application and the CAVE5D and VizAD visualization tools. In other work, interactivity support has been integrated with high-end visualization to provide steering of a parallel molecular dynamics application [7], Habanero [14] and Tango [15] provide collaborative support with Java-based environments, and Cavernsoft provides collaboration support for Cave-based visualizations[16].

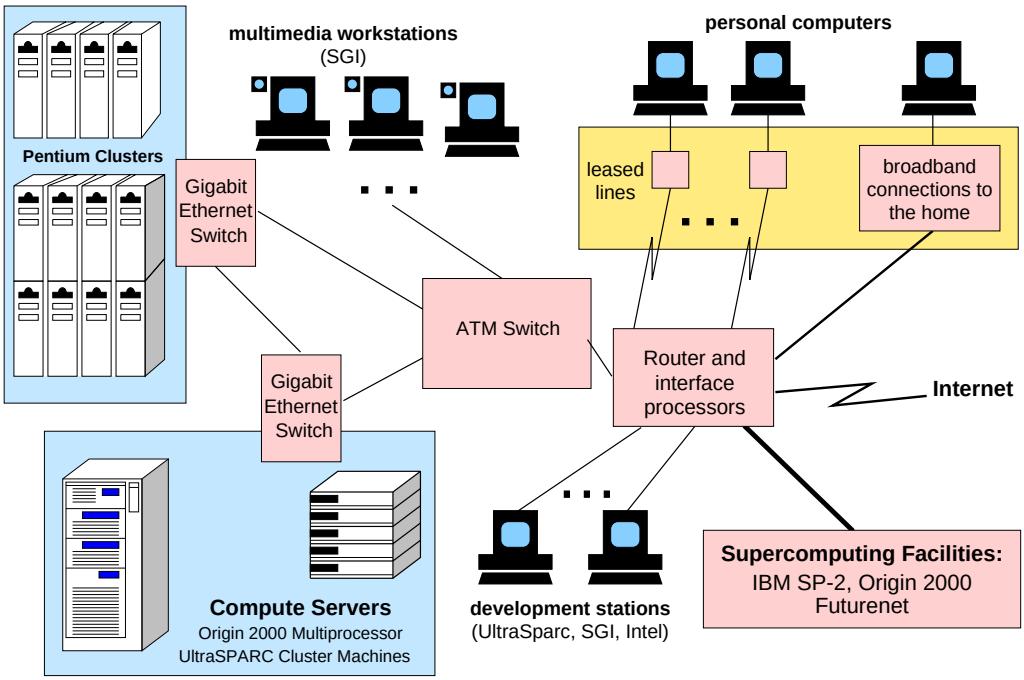

Fig. 1. Hardware infrastructure at Georgia Tech.

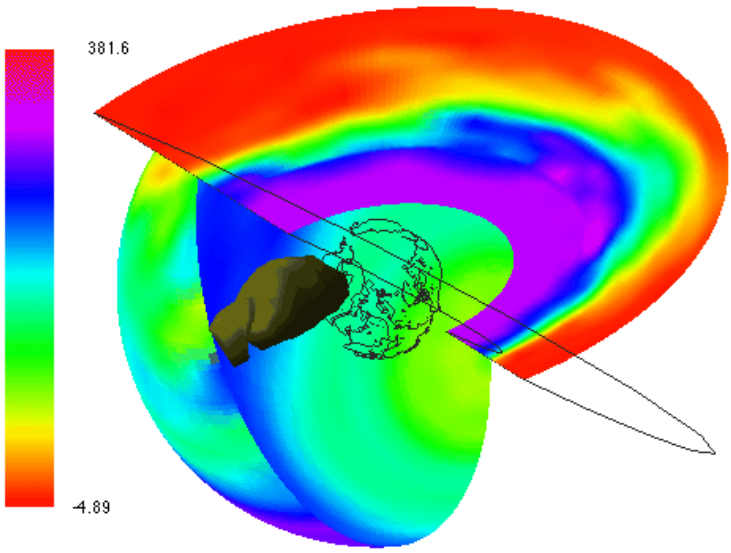

Fig. 2. N^2O species concentration with isosurface positioned over the South Pole.

III. Middleware Services for Distributed Laboratories

A. Service Description and Usage

Consider an atmospheric scientist working with a global chemical transport model, modeling the dispersion of chemical species through the atmosphere. The scientist starts the model, brings up a visualization/steering interface, but then decides to enlist the help of a chemist at a remote site to explain an observed phenomenon. Toward this end, the remote researcher brings up a visualization suited to understanding model output from a chemical perspective. Subsequently, the atmospheric scientist and the remote chemist take turns steering the model, changing parameter values, viewing the effects of such changes, rolling back the model to earlier points in the simulation and trying a different set of parameters until a resolution is found.

The complex physical models used by these scientists have computational and data storage demands that require their parallel and distributed implementations. Assuming this fact, our work focuses on the resource demands due to the extensive application interactivity inherent in scenarios like these. Namely, when the possibility of multiple spatially separate users is considered, the display and interactivity system itself becomes a complex, dynamic, and resource-consuming distributed/parallel system. The goals of our research are to identify the unique demands of this system and to develop a software infrastructure that meets those demands. This section identifies these demands, by providing a brief description of resources and their usage due to interactivity.

Task	*Description*
communication	– variable bandwidth/latency transfers – multiple transmission methods for heterogeneous media – data streams with intermediate tasks – binary I/O
computation	– information filtering and analysis – use of heterogeneous execution engines
storage	– binary data formats, meta-data, restart and I/O files – use of distributed and heterogeneous storage engines
display	– highly diverse and active user interfaces – use of direct manipulation
monitoring and steering	– capturing, transporting, and analyzing application-specific and performance data – non-uniform perturbation, latency, throughput requirements – performance tuning, program adaptation, and program steering imply: highly diverse needs for latency and perturbation
collaboration	– client to client vs. via shared computational instruments – alternative collaboration methods

TABLE I
Basic Services.

Table I lists the basic tasks of an interactivity infrastructure, followed by a brief characterization of those tasks. Concerning communication, the binary data flowing from applications to clients is likely to pass through multiple intermediate information filtering and analysis tasks prior to reaching clients. For computation, this implies that such tasks must be mapped and scheduled on heterogeneous underlying execution engines in conjunction with the mappings performed for the computational instruments themselves. Similarly, the storage of intermediate and final results (e.g., to enable rollback) must deal with heterogeneous platforms and file formats. Furthermore, displays are problematic, particularly when considering the simultaneous use of interfaces that collaborate across both home PCs and laboratory machines. The implications on monitoring, steering, and collaboration derived from these highly variable resource characteristics and needs are apparent: no single 'software platform' or solution is likely to provide appropriate services to end users. This is reflected in our own work, in which we are attempting to provide a CORBA-like framework for building high performance interactivity systems able to cooperate across heterogeneous software and hardware systems. It is also reflected in the broader community's provision of both C/C++ and Java-based visualization, monitoring, steering, and collaboration support.

The entries of Table I concern the basic resource needs and behaviors of interactivity systems; they do not address dynamic system behaviors. For example, when a scientific researcher wishes to enlist the help of a remote collaborator, a new data stream must be created dynamically, 'tap' into the ongoing flow of data from the application to the end user, and establish its own parameters and needs with respect to required data items, intermediate analyses, transfer rates, and displays. For example, the chemist in the atmospheric simulation requires grid-based visualizations of chemical concentrations, wishes to consider cumulative effects across multiple timesteps, and may be constrained by a relatively 'lightweight' home-based connection. We refer to the characterization and support of the dynamic behaviors of interactivity systems as *experiment management*.

Table II identifies the components involved in experiment management and some of the necessary operations on

Task	*Description*
computational components	– filter – transformer – composer – adaptation-awareness
experiment	– mapping and scheduling computations – component placement and composition – remote execution control and component adaptation – runtime optimization across sets of components (e.g., migration) – dynamic connection of clients and data flow management – data stream creation and control – dynamic visualization needs

TABLE II
QoS and Management Services.

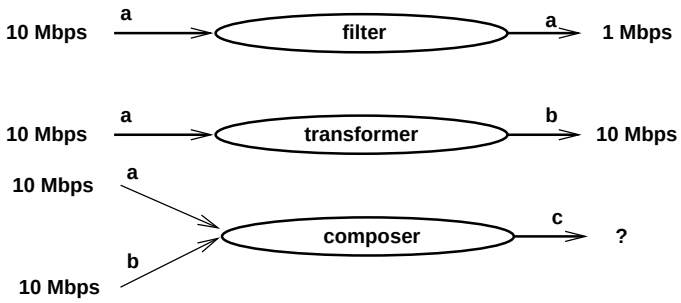

Fig. 3. Computational component primitive types.

those components. Generally, each data stream emanating from or targeting a certain computational instrument filters, transforms, or composes data. These actions may be fixed at the time of component creation or they may be adapted according to current interactivity needs. For example, the stream of chemistry information used by the home PC-based researcher may be subject to runtime adaptation in terms of the degrees of filtering the levels of offered data resolution in order to match variations in home PC connectivity due to changes in intermediate Internet traffic. In this case, experiment management not only involves the initial placement and composition of streams and stream tasks, but also the control of streams, including runtime adaptation like adjustments in the resolution of the data being sent[17]. In addition, experiment management may perform optimizations across a set of streams maintained in the system. Concerning the latter, consider two streams providing identical information to two end users: rather than duplicating conversion and filtering tasks, such tasks may be shared and stream contents may be multicast to both end users.

The remainder of this section seeks to make experiment management more concrete, by description of sample computational components, characterization of their needs, and discussion of sample actions performed on them.

B. Managing Computational Components

We use the term *computational components* to describe the general class of computational tasks performing intermediate processing on the data flows of distributed laboratories. Such tasks may be small, relatively mobile, and easily relocated closer to the data generation source, thus lessening the total network bandwidth consumed by the data streams. Tasks may also themselves be computationally or data-intensive, thus useful to offload processing from visualization and steering clients, which is particularly important for scientists interacting from home PCs.

Computational components are composed from three primitive types: filters, transformers, and composers, as shown in in Figure 3. A *filter* reduces the event rate by discarding unwanted events. A transformer generates a new event type in response to an event received. When clients contend with data from heterogeneous sources, differences in rate of flow arise. Filters can decrease the overall consumed bandwidth and can reduce the processing load of the recipient, the latter being particularly important to resource-constrained recipients. Using mechanisms we have developed, filters can also be adapted dynamically to the needs of a particular client or group of clients. A *transformer* by itself without a filter generates one event for every event received. A *composer* produces a new event type given two or more input types. As shown in the figure, a single event **c** is generated for every event **a** and **b** received.

Figure 4 illustrates sample computational components in the context of the atmospheric model. The parallel global transport model is depicted as a single process, whereas the chemical model is decomposed into processes by atmospheric level; it is shown running on a heterogeneous set of cluster machines. Observational data exists in a repository. Several visualization and steering clients exist: a 3D OpenInventor-based visualization and steering client running on a high-end

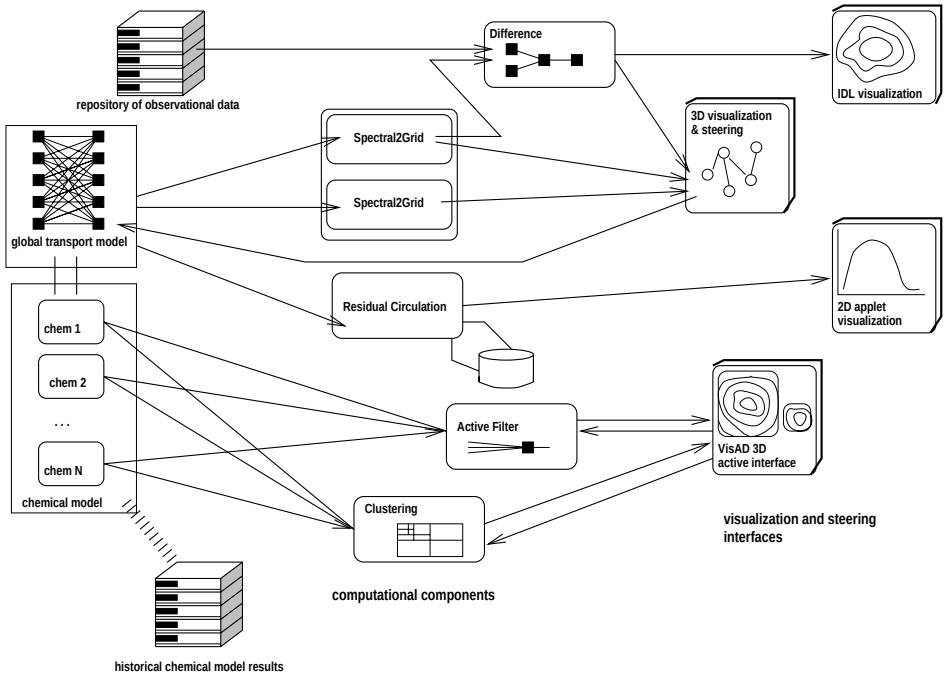

Fig. 4. Computational components in a distributed laboratory.

SGI graphics engine, a VisAD Java-based visualization client, a low-end Java visualization, and an IDL [18] visualization client. IDL is a scientific visualization tool used by Earth and Atmospheric scientists to provide high resolution views of 2D slices of selected atmospheric data.

The computational components have varying tasks. Since the transport model performs most of its computations in the spectral domain, yet the most meaningful representation for display of this data requires a grid-based representation, the spectral representation must be evaluated at each grid point. Since this is a computationally expensive conversion, we employ parallel `Spectral2Grid` transformers. `Difference`, `Residual Circulation`, and `Clustering` are all examples of composers. `Difference` accepts ozone observational data from a repository and computes ozone data from the transport model. The computed data is converted from spectral to grid representation before `Difference` colors each grid point based on the calculated variance between computed and observed values at that point. Since model data is generated with higher frequency than observed data (once every two hours versus once a day), the component also filters unneeded transport events. `Residual circulation` computes the average species concentration over one month, factoring out the effects of planetary waves. `Clustering` creates multiple levels of abstraction on the data by grouping values based on common characteristics, with the ability to zoom into and out of lower levels. The final component, termed `active filter`, is discussed below, following the definition of *active interface*; active interfaces and active filters work together to create a dynamic environment that can be highly responsive to user needs.

B.1 Active Interfaces.

An *active interface* is a user interface which continuously exports its state by pushing a stream of user events to the interactivity infrastructure and to specific computational components. State export is performed for two reasons: (1) to allow the interface to offload selected data analysis and processing tasks to remote machines and/or for concurrent processing of those tasks, and (2) to influence and steer those tasks and data generation to ensure they meet current user needs. The events produced by the interface may result not only from explicit button clicks or menu choices but also implicitly from a user's direct interactions with the data. While researchers have developed similar feedback mechanisms for accessing information databases [19], and tools for the direct manipulation of database data [20], active interfaces differ in that they integrate feedback, direct manipulation, and distributed processing.

Our current implementation of active user interface provides user events that export state in response to a user's direct manipulations of the data being visualized. Each such event consists of a bounding box and a preferred resolution for the box. The bounding boxes defined correspond to (1) a focus area of special importance, (2) the area onscreen or currently visible, and (3) a larger area that might be onscreen in the future. Each of these three regions has a natural meaning applicable to almost any visualization or graphical environment. In a VR system, for example, they correspond to objects in the center of vision; in the peripheral vision; and behind the user's head or just out of sight. Likewise, each has clear implications for quality of service, from higher level of detail at the center, to low-resolution prefetching in the

area offscreen.

B.2 Active Filters.

An *active filter* is an adaptation-aware computational component responsive to an active interface. Active filters either directly respond to events produced by active interfaces or more typically, they are controlled by experiment managers. A manager receives events from active user interfaces, is aware of current state of computational components, and may adjust components to better serve a certain interface, the data streams in which it participates, and/or the experiment being performed. Our work on active filters is based on dynamic queries [21], wherein complex filter conditions are easily stated using a temporal SQL-based query language. Condition analysis is implemented as the evaluation of these temporal SQL queries on the event flows and built into the work is an approach to modify and reoptimize the queries dynamically.

In data-intensive scientific applications, it is not immediately apparent how amenable the data is to access by a query evaluator. For example, for the atmospheric model, should the user have the ability to specify a condition involving any single gridpoint in the earth's atmosphere? Clearly, the overhead to implement such visibility would be high. To avoid such overheads, we assume that end users or experiment implementors state attributes for data records that expose relevant information about the data. Queries, then, are specified over these attributes. For instance, for the data event containing grid information for the atmospheric model, the grid data may be stored as a contiguous array of floating point numbers, while the attributes might include pointers into the array corresponding the beginning value for a particular latitude.

C. Evaluation

When assembling an experiment from disparate computational components, there will be differences, at times significant, in the event consumption and generation rates of components. To illustrate, as depicted in Figure 4, consider the case where the transport model generates spectral events that first flow through the `Spectral2Grid` transformer, which then produces grid events that are fed to `difference`, prior to the forwarding of difference events to the IDL visualization. When the transport model is run on a 12-processor SGI Powerchallenge connected via ATM to an Ethernet switch, and when both computational components are assigned to a uniprocessor SGI Indy workstation connected via 10Mb switched Ethernet, Table III shows a transport model generating events at an oscillating or cyclical rate, perhaps due to load variations of such a nature on the transport machine. Specifically, the two values for `Spectral2Grid` and `difference` represent their respective input and output event rates, and the 'output' value for IDL Vis represents the rate at which the client is able to visualize events. In these examples, an output event rate that is consistently smaller than the input event rate indicates that the computational component currently does not have sufficient CPU resources to process all of the events it receives. An input event rate that initially exceeds the output event rate but eventually decreases to be less than the output rate indicates that the instrument is 'catching up' on previously buffered events. The former is illustrated in Table III by the 10min entry for Spectral2Grid. The latter is illustrated by a combination of the 10min and 15min entries for `Spectral2Grid`, where `Spectral2Grid` has used the lapse in transport event generation existing at the 15min time to process its backlog.

Cyclic/oscillating Event Rate (in/out)(2x)				
Time	Global Transport	Spectral 2Grid	Difference	IDL Vis
5min	74(1x)	74/74	74/74	74/74
10min	148(2x)	148/120	120/120	120/120
15min	74(1x)	74/74	74/74	74/74
20min	148(2x)	148/120	120/120	120/120

TABLE III

Event rate for components where transport model generates events at an oscillating rate

These experiments illustrate the need for solutions to heterogeneous components in interactivity infrastructures. We are addressing the problem with three different approaches. First, our work on the Interactivity Layer Infrastructure described in [22] explores two specific component adaptations, which are component cloning and migration. By monitoring attributes like event rates, one can detect an imbalance and either clone a new component (e.g., Spectral2Grid when the output rate consistently falls below the input rate), or migrate a computation to another resource with, say, a lower CPU load. The clone adaptation performs well at removing computational problems while the migrate adaptation better serves bandwidth problems. We will explore other adaptations as well. For instance, a merge adaptation will be

applied when an instrument consistently does not meet its minimum event threshold and there exists another identical instrument that is at no more than 75% of its maximum event rate threshold.

The second approach addresses the fact that many monitoring systems are built using an event-stream abstraction. Namely, they treat the application and monitoring clients as processes linked by a stream of data records. Some event stream-based systems can direct or filter monitoring data records by type, but they generally do not directly support content-based event routing[23], [24]. This limitation simplifies the event handling infrastructure, but such systems cannot handle the selective data distribution necessary for the data-parallel analysis of monitoring data. In addition, some systems draw a sharp distinction between the application and monitoring clients, thereby making it difficult to build multi-level data reduction schemes[25], [26].

In response to the difficulties described above, our work in MOSS [27] established goals for a new model for program monitoring and steering systems:

- to scale gracefully with increasing complexity in both the application and the monitoring system;
- to provide for automatic control of monitoring data collection in the presence of many simultaneous clients;
- to permit self-application for multi-level processing and control; and
- to provide a natural model of steering.

Specifically, MOSS [27] is an object-oriented monitoring and steering infrastructure that meets these goals by employing CORBA style event channels as a replacement for event streams. The benefits are clear: on a Sun Sparc Model 170, the conversion of one level of atmospheric species data from spectral to grid form requires 2.5msec, consuming 253 complex numbers (about 2 Kbytes). If the conversion of all 37 levels were decomposed across 37 processors and run at full rate, MOSS would require a network bandwidth of roughly 30Mbps between the application and the conversion processors. In comparison, an event stream-based system would require 37 times as much bandwidth, or over 1 $Gbps$. Conversely, if we were to assume that bandwidth were limited to 100 $Mbps$, MOSS could easily still support a 37 processor system which would process the model output at 400 Hz. However, an event stream-based solution would saturate the network at 11 processors, a level of parallelism which would result in a 100 Hz processing rate.

Finally, a simple example illustrates the third approach to the event rate mismatch problem: active interfaces/active filters. If the transport model generates events at a rate of 30Mbps on an R10000-based SGI Origin machine and if Spectral2Grid requires .979 ms to convert a single record (one timestep and one level) from spectral to grid form, then for Spectral2Grid to keep up with model output, seven parallel conversion components are needed. However, if the active interface were to return specific information as to latitudes of interest, Spectral2Grid would be able to accept a range of latitudes and then convert only the requested latitudes. Thus, if the user-specified filter effectively filters 85% of the received records, then only a single Spectral2Grid component would be needed to keep up with the flow of events from the transport model. We will pursue effective information filtering using query languages and associated optimizations derived from prior work on temporal databases.

IV. Integrating a Multi-model Scientific Application

A. Background.

Substantial recent work by our group involved the parallelization of a complex chemical atmospheric model obtained from M. Pirre of the Laboratoire de Physique et Chimie de l'Environment, CNRS, Orleans, France [28], and the integration of this code into an existing parallel transport model. The former incorporates 31 chemical compounds involving 60 gas phase reactions and 15 photochemical reactions, all of which are evaluated at each time step on a global basis, either on the model's three-dimension grid mesh or in the spectral domain. The model's purpose is to enable atmospheric scientists to model the production, consumption, and transport of more reactive species like ozone.

We report this work here, as it constitutes an example of the complex parallel/distributed applications considered in our research. More specifically, we comment on the effects of heterogeneity in computational/communication overheads on the end users interacting with this complex computational instrument. In this research, our integration efforts were facilitated by the fact that the spatial scales of both models are identical; that is, the transport model and chemical model use the same grid resolution, which is approximately 2.8 degrees by 2.8 degrees. This is equivalent to 2048 grid points at each layer. Moreover, while in the levels above 37, chemical interactions still take place and influence the interactions of species at lower levels, no actual species transport is performed in the model. This is not an unreasonable assumption as the winds above 1 millibar are not as strong as those at the lower altitudes.

B. Parallelization.

Parallelization involves the creation of one chemistry process per atmospheric level, at the potential cost of increased communication costs due to the increases in data volumes passed between transport and chemistry. Such communication as well as communication between different chemical levels are performed with a customized all-to-all communication scheme from transport to chemistry, followed by one-to-one, level-to-level return data communications. Specifically, temperature data undergoes a change every timestep as a result of transport, and every level of the chemical model requires temperature data for every level.

C. Performance Issues with this Computational Instrument.

Although online interaction with the transport model is straightforward, timings show that the chemistry computations are more than three orders of magnitude slower than transport when measured for a single time step. This presents a problem for the integration of the two codes, resulting in different levels of parallelization for each. We address this problem by additional parallelism 'inside' each chemistry process, and by the judicious mapping of chemistry vs. transport tasks to parallel machines. It is unclear to us, however, how to address the problems posed by the interactive steering of this application by multiple end users. Specifically, when such steering is performed, we may (1) have to force the end user concerned with transport to operate at the 'slower rate' of the chemistry code, and (2) it is unclear how to deal with the fact that the evaluation of issues with transport typically requires a larger number of model time steps than the evaluation of the chemical reactions that are taking place. In other words, when interacting with these codes, different end users will wish to operate at different time scales, even if the codes themselves have been integrated successfully. We posit that additional functionality must be introduced into the interactivity infrastructure to deal with such issues.

V. Current Directions in Experiment Management

As demonstrated in Section III-C, experiment management is important when dealing with mismatches between computational components and with changes in the needs of individual end users. Experiment management is also important when dealing with large-scale experiments comprising many computational instruments, components, and end users. For such experiments, experiment management provides the single point of control from which experiments may be initiated and controlled, users can view current resource characteristics, allocate resources to tasks, start component pieces of the experiment, and view experiment progress. Toward these ends, an experiment manager must perform the following tasks:

- experiment setup, control, and teardown,
- data management,
- configuration management, and
- configuration analysis and adaptation.

The first item concerns the explicit definition and control of experiments by end users, including the definition of suitable input and output files for computational instruments, the specification of experiment repetition and completion, etc. This paper does not address these topics. Instead, we focus on the remaining three tasks, where the relationships between these tasks are shown in Figure 5. The *data manager* maintains information about resources and computational components, and it provides efficient access and update methods to this information. The *configuration manager* determines appropriate locations for computational components, using knowledge of the information maintained by the data manager. It also manages remote execution control, that is, it provides low-level services for tasks like starting components on remote machines. Finally, configuration analysis implements the necessary feedback loop. By gathering information from the experiment, it couples this information with information retrieved from the data manager to make informed decisions that affect future placement and adaptation in the experiment environment.

A. Data Management.

Data management incorporates the management and storage of all information related to a particular experiment. Toward this end, an experiment is represented as a set of resources and a set of computational components. A *resource* might be a CPU, hard disk, RAID controller, or data file, but could also be extended to include networks, labs, and people. *Compute objects* on the other hand, include computational instruments, historical data file readers, visualization and steering clients, and computational components. Informally, resources can be thought of as being the physical entities required by the compute objects.

The data manager maintains a repository, where each entry has an associated set of attributes. There are several alternatives for repository representation, including the encapsulation of information in data management objects [22], or the use of the Metacomputing Directory Service (MDS) [29] part of Globus [10] for information management. With any such realization, *resource attributes* include the resource ID and its type, plus detailed information such as operating system name and version, swap space, and processor load for each processor. *Compute object attributes* include an ID and type, list of input and output event types, set of QoS characteristics, and a set of constraint conditions. The QoS attributes define the processing capacity and needs of the compute object and include the current processing rate (e.g., expressed in events per some unit of time) for an event type e, maximum and minimum rates, rate history, and a list of the adaptations available for the component. The constraint conditions used during placement and analysis are described in more detail below.

B. Configuration Analysis and Adaptation.

A static allocation of computational components to resources is not suitable when clients come and go, and when their resource demands change dynamically. The configuration analysis component is responsible for extracting information from the ongoing experiment and making decisions to reconfigure based on the feedback it receives.

Reconfiguration can involve the dynamic creation and placement of a new component, or adapting or removing an existing component. Reconfiguration is performed based on the facts that certain quality of service constraints are satisfied or are being violated. Our past work for real-time applications has developed high fidelity models of service quality and of expected results of component adaptations[30]. For distributed laboratory applications, we are pursuing two complementary approaches. The first approach is based on high level representations of computational components and their interactions using the workflow model. In this model, tasks are represented as nodes in a graph. A corresponding constraint violation graph records the magnitude of constraint violations with respect to the QoS attributes listed above. There are four reasons for such violations: the node is sending (1) too many or (2) too few events downstream, or it is receiving (3) too many or (4) too few events from upstream. The node withe the largest magnitude in the constraint graph exhibits the largest violation. Once it has been determined that reconfiguration is necessary, a new configuration is computed.

The second approach is based on a database model. The constraints associated with a computational component are specified in the form of queries. Specified also for a computational component are a set of application-level requirements or attributes. Based on the work done in [31] in which queries are transformed into executable entities that can then be applied to an event stream, the constraint can be used to aid in query placement (or replacement). For example, there are cases when multiple data sources supply the same event types. In this case, more detailed information in the form of attributes/conditions is needed. The role of attributes and conditions in the placement algorithm is described in the context of Figure 5. Placing component C:1 is straightforward because only process A exports the event type b, so C:1 is placed on machine I. Placing C:2 requires additional information since it requires event type a which is exported by both A and B. By applying C:2's placement condition against the attribute list of A and then of B, one has further information with which to make a placement decision. As an example, suppose A and B are part of a global simulation of the earth's atmosphere where the simulation is structured such that the atmosphere is partitioned into 37 horizontal layers with A computing species transport for layers 0-17 and B simulating the remainder. The attributes stored at the component registrar for A and B would include starting and ending layer numbers. The condition associated with C:2 is then applied against the attributes. In Figure 5, the condition would be true for process A but not for B.

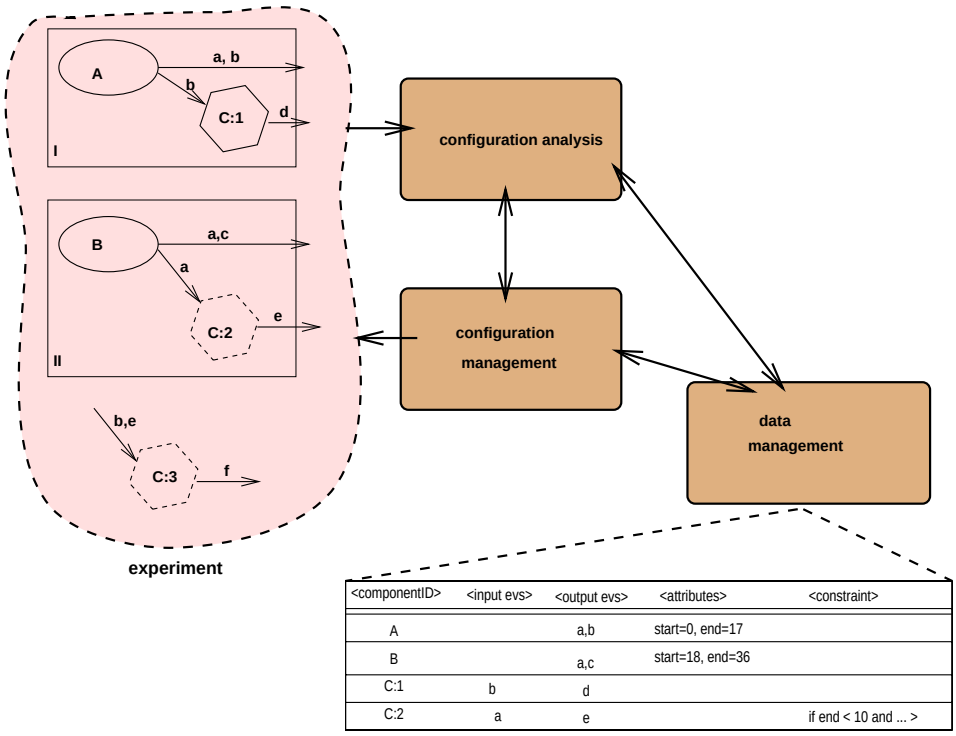

Fig. 5. Experiment manager component placement.

C. Configuration Management and Optimization.

The configuration manager determines component placement. Component placement in [22] is determined through a mapping of tasks to resources. The initial configuration is represented as a graph. The user specifies the initial task set and input/output event types. The event data is used to construct connections in the task lineage graph. Since the initial allocation of components is not likely to remain suitable throughout each experiment's execution, the adhoc approach used by ILI[22] periodically checks to determine whether or not all tasks are currently meeting their QoS constraints and if not, determines a more suitable configuration. That is, a configuration that will either eliminate violations or change the system to a state with fewer violations.

A more rigorous second approach we are exploring draws its motivation from database theory and adapts query optimization and cost functions. As we have shown earlier in our discussion of transformers, composers, etc., computational components may be described in terms of primitive types. Adapting work on query optimization and cost functions, it is possible to express computational components and their relationships to one another and to models, data sources, visualization clients, etc. as an abstract syntax tree. We are working on suitable cost functions that can be applied to the existing AST to generate a more optimal one [32].

VI. Related Research

The Globus project [10] is developing technology to build *computational grids*, execution environments that enable an application to integrate geographically-distributed instruments, displays, computational and information resources. These environments are similar to the environments envisioned in Distributed Laboratories. Work in Globus concentrates on resource location, allocation and security concerns. Additionally, Globus works to leverage existing tools, such as MPI [33] in the Globus environment. Our work is largely complementary to that of Globus and could be integrated into the Globus framework. In particular, our monitoring and steering infrastructure, collaboration and QoS infrastructure address issues not currently targeted by Globus.

Another large meta-system project similar to Distributed Laboratories is Legion [34]. Legion is an infrastructure which transparently provides scheduling, data transfer and coercion, communication and synchronization to parallel applications written using the Legion object system. Unlike Legion, Distributed laboratories does not impose an object-based programming style upon applications. While some tools, such as MOSS (see Section III-C), are object-based, they do not assume that all applications in the Laboratory are so written. Additionally, Legion focuses on parallel execution of monolithic applications, as opposed to the more loosely coupled collaboratories addressed by Distributed Laboratories.

VII. Conclusions and Future Research

This paper has defined Distributed Computational Laboratories, in which multiple, distributed end users interact with each other via shared computational instruments. The two topics explored in detail are the composition of such laboratories and their runtime behavior and control. Specifically, software infrastructures for distributed laboratories must deal with the dynamic arrival and departure of individual clients, must serve clients that differ in their service needs and in the capabilities of the platforms they are using, and they must consider dynamic behavior in the computational instruments themselves and in the ancillary computations and data streams required for their execution and for client interactivity and collaboration. The implication of these characteristics is that software infrastructures for distributed laboratories must provide not only the basic services enabling laboratory setup and configuration, but must also provide support for the specification and enforcement of quality requirements for those services.

Concerning basic services, this paper describes sample laboratories constructed for scientific applications, using basic services providing for event-based data and control information[27] exchanges between interacting and collaborating instruments. Concerning quality requirements on services, we describe several approaches for specifying and enforcing such quality requirements, in the presence of dynamic changes in resource availability and in instrument and end user needs. Results of our work include the identification of naturally occurring differences in instrument quality (e.g., different event input vs. output rates) and ways of dealing with such differences, including the runtime use of service-specific adaptations. Additional results include the exposition of multiple approaches to developing software infrastructures for quality and experiment management.

The Distributed Laboratories infrastructure is applied to an atmospheric modeling application, which is also described in this paper. Interesting results of its application include the identification of issues and the presentation of performance results concerning the interactive use of this application.

There is much evidence that future scientific research environments will heavily rely on Distributed Computational Laboratories. The topic on which we will continue to focus is the runtime configuration and control of such laboratories, based on explicitly or implicitly user-centric notions of the quality of service required for their effective use. Our future work will continue to explore this topic by construction of sample sample laboratories, jointly with end users at Georgia Tech and with additional end users who are also members of the NCSA Alliance. In addition, based on a

large-scale grant received by Georgia Tech from Intel Corporation and using both Unix and NT systems, we are already constructing a Distributed Labs hardware and software infrastructure that spans the campus units participating in this effort (Industrial, Mechanical, Aerospace, and Electrical Engineering, and the School of Physics). Our next steps in this effort concern the extension of the campus-wide software infrastructure to one that spans multiple, geographically separated sites connected via the Internet, including home locations.

References

[1] D.A. Reed, R.D. Olson, R.A. Aydt, T.M. Madhyastha, T. Birkett, D.W. Jensen, B.A.A. Nazief, and B.K. Totty, "Scalable performance environments for parallel systems," in *the Sixth Distributed Memory Computing Conf.*, 1991, pp. 562–569.

[2] David Ogle, *The Real-Time Monitoring of Distributed and Parallel Systems*, Ph.D. thesis, Department of Computer and Information Sciences, The Ohio State University, Aug 1988.

[3] B.P. Miller, M. Clark, J. Hollingsworth, S. Kierstead, S.-S. Lim, and T. Torzewski, "IPS-2: The second generation of a parallel program measurement system," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 206–217, April 1990.

[4] T. Bihari and K. Schwan, "Dynamic adaptation of real-time software," *ACM Transactions on Computer Systems*, vol. 9, no. 2, pp. 143–174, May 1991.

[5] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, and J. Vetter, "Falcon: on-line monitoring and steering of large-scale parallel programs," in *Proc. Frontiers '95*, 1994, pp. 422–9, Also available as TR GIT-CC-94-21.

[6] S.G. Parker and C.R. Johnson, "SCIRun: a scientific programming environment for computational steering," in *Proc. Supercomputing 95*, 1995, pp. 1–1.

[7] G. Eisenhauer, W. Gu, K. Schwan, and N. Mallavarupu, "Falcon - toward interactive parallel programs: The on-line steering of a molecular dynamics application," in *Proc. Third Int'l Symp. on High-Performance Distributed Computing (HPDC-3)*. 1994, pp. 26–34, IEEE Computer Society.

[8] Rich Wolski, "Dynamically forecasting network performance to support dynamic scheduling using the Network Weather Service," in *Proceedings of 6th High-Performance Distributed Computing (HPDC6)*. IEEE, 1997.

[9] Glen H. Wheless, Cathy M. Lascara, Donald P. Brutzman, William Sherman, William L. Hibbard, and Brian E. Paul, "Chesapeake bay: Interacting with a physical/biological model," *IEEE Computer Graphics and Applications*, vol. 16, no. 4, July/August 1996.

[10] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[11] R. Swinbank and A. O'Neill, "A stratosphere - troposphere data assimilation system," Climate Research Technical Note CRTN 35, Hadley Centre Meteorological Office, London Road Bracknell Berkshire RG12 2SY, March 1993.

[12] Thomas Kindler, Karsten Schwan, Dilma Silva, Mary Trauner, and Fred Alyea, "A parallel spectral model for atmospheric transport processes," *Concurrency: Practice and Experience*, vol. 8, no. 9, pp. 639–666, November 1996.

[13] 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.

[14] Habanero, "National Center for Supercomputing Applications and University of Illinois at Urbana-Champaign," http://notme.ncsa.uiuc.edu/SDG/Software/Habanero.

[15] G.C. Fox and W. Furmanski, "Java and web technologies for simulation and modeling on computational science and engineering," in *Proc. of the 8th SIAM Conf. On Parallel Processing*, 1997.

[16] J. Leigh, A. E. Johnson, and T.A. DeFanti, "Issues in the design of a flexible distributed architecture for supporting persistence and interoperability in collaborative virtual environments," in *Proceedings of Supercomputing '97*, San Jose, California, November 1997.

[17] Davis King, Greg Eisenhauer, Beth Plale, Bill Ribarsky, and Karsten Schwan, "Content-based steering of data streams in distributed laboratories," Submitted to SuperComputing'98.

[18] Interactive Data Language, "Research Systems," http://www.rsinc.com.

[19] E. Tanin, R. Beigel, and R. Schneiderman, "Research report design and evaluation of incremental data structures and algorithms for dynamic query interfaces," in *Proceedings of Visualization '97*, 1997, pp. 81–86.

[20] M. Derthick, S. F. Roth, and J. Kolojejchick, "Coordinating declarative queries with a direct manipulation data exploration environment,," in *Proceedings of Visualization '97*, 1997, pp. 65–72.

[21] Beth Plale and Karsten Schwan, "Dynamic optimization in on-line detection of safety constraint violations," in *Submitted for publication*, 1998.

[22] Vernard Martin and Karsten Schwan, "ILI: An adaptive infrastructure for dynamic interactive distributed systems," in *4th International Conference on Configurable Distributed Systems*. IEEE, 1998.

[23] Weiming Gu, Greg Eisenhauer, and Karsten Schwan, "Falcon: On-line monitoring and steering of parallel programs," to appear in *Concurrency: Practice and Experience*.

[24] Jeffrey Vetter, "Computational steering annotated bibliography," *ACM SIGPLAN Notices*, vol. 36, no. 6, 1997.

[25] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam, and Tia Newhall, "The paradyn parallel performance measurement tools," *IEEE Computer*, 1995.

[26] B. Mohr, A. Malony, and J. Cuny, "Tau," in *Parallel Programming using C++*, G. Wilson, Ed. MIT Press, 1996.

[27] Greg Eisenhauer and Karsten Schwan, "An object-based infrastructure for program monitoring and steering," in *To appear in Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT'98)*, 1998.

[28] R. Ramaroson, M. Pirre, and D. Cariolle, "A box model for on-line computations of diurnal variations in a 1-d model: potential for application in multidimensional cases," *Ann. Geophysicae*, vol. 10, pp. 416–428, 1992.

[29] 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*. 1997, pp. 365–375, IEEE Computer Society Press.

[30] Daniela Ivan Rosu, Karsten Schwan, Sudhakar Yalamanchili, and Rakesh Jha, "On adaptive resource allocation for complex real-time applications," in *18th IEEE Real-Time Systems Symposium, San Francisco, CA*. IEEE, Dec. 1997, pp. 320–329.

[31] Beth (Plale) Schroeder, Sudhir Aggarwal, and Karsten Schwan, "Software approach to hazard detection using on-line analysis of safety constraints," in *Proceedings 16th Symposium on Reliable and Distributed Systems*. IEEE Computer Society, October 1997, pp. 80–87.

[32] Richard Snodgrass, ," 1998, Personal correspondence.

[33] MPI Forum, "MPI: A Message-Passing Interface," Tech. Rep. CS/E 94-013, Department of Computer Science, Oregon Graduate Institute, March 94.

[34] Andrew S. Grimsaw and Wm. A. Wulf, "Legion – a view from 50,000 feet," in *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, Los Alamitos, California, August 1996, IEEE Computer Society Press.