Remote and Distributed Visualization Frameworks Workshop

Findings Document
April 14-15, 2003
Emeryville, CA

Executive Summary

This report presents the findings and recommendations that emerged from a two-day workshop held in Emeryville, CA on April 14-15, 2003. The motivation for the workshop was to identify objectives and goals related to future activities in visualization software design and development. The workshop participants included visualization researchers from a number of DOE and government laboratories, as well as academic institutions.

Visualization, which is the transformation of abstract information into images, plays an integral role in the scientific process by facilitating insight into observed or simulated phenomena. The evolution of visualization software over the years has shown that component-based tools executing within a well-defined framework offer the greatest potential in several key areas. The combination is extensible, as developers can add new components that extend the capabilities of the system. It is flexible, since general purpose components can be combined into applications that address domain specific needs. It provides a common development environment so that components of one institution can interoperate with components from another. The boundaries of such component-based frameworks are encountered when attempting to perform visualization of large datasets, when attempting to use components that span multiple locations, when attempting to perform collaborative visual analysis and when attempting to use components and finished tools from different sources. Unfortunately, there does not exist a common framework or component definition that is suitable for use in a remote and distributed visualization (RDV) context. Such a component architecture, which should be freely accessible and community-developed, is sorely needed to better support science programs, and to promote unity within the community of visualization researchers, developers and users.

The workshop findings indicate that such a framework is technically feasible, and will be of great use for visualization researchers, developers and users. A key objective is the ability to integrate a diverse range of existing technology, including commercial visualization applications and prototypes of new techniques from the research community. Another objective is the ability to use, within a single framework, diverse resources ranging from large parallel machines to small desktop workstations, where such resources span multiple sites. The framework should support display device independence so that it can be used in a modular fashion as the back-end for diverse presentation modalitities, ranging from PDAs through desktop platforms, and immersive display environments. It should also support single or multiple user operation. To be successful, the framework and components should be made available to all through an Open Source license at no cost to promote widespread adoption and use, and to ensure a stable environment for scientific and visualization research and development in the years ahead.

Problem Statement

Scientific visualization - the transformation of abstract information into readily comprehensible images - plays an integral role in the scientific process. Over the years, scientific visualization has evolved to keep pace with advances in computer science, software engineering, data modeling and data management. The earliest scientific visualization "systems" were subroutine APIs, and each new visualization application required writing software to read data and perform subroutine calls to perform visualization and rendering. Later, applications that implemented procedural data processing (and visualization) languages emerged, such as IDL, Matlab and the like. During the 1990s, a class of visualization systems emerged that were highly successful. These systems were composed of visualization software components, an interface for rapidly constructing an application using the provided components, or possibly new components created by a developer. An internal "flow executive" scheduled execution of these components and moved data through the network of software components. These systems were unexpectedly successful because they were extensible, since developers could extend the functionality of the base system through the addition of custom software components. Communities of users and developers coalesced around each of these systems, for a common framework and data model fostered a stable environment for sharing components, data and ideas.

The language-based systems as well as the dataflow systems, while immensely successful, have inherent design limitations that prevent their use, or adaptation for use, in a class of contemporary and near-term environments. Remote and distributed visualization environments pose new challenges not anticipated by the designers of earlier visual data analysis technologies. The challenges include the need to use distributed heterogeneous resources, the resource demands large and distributed scientific data, the ability to include capabilities that are fundamental for science (e.g., statistical analysis), the need for communities of developers and users to share visualization tools and resources, and the need for a technology base of “raw materials” that can be quickly adapted to domain-specific use.

Definitions

The following list of definitions were borrowed from www.cca-forum.org/glossary.html, and adapted for use in the context of RDV components and frameworks.

A **component** is a software object, meant to interact with other components, encapsulating well-defined functionality or a set of functionalities. A component has a clearly defined interface and conforms to a prescribed behavior common to all components within an architecture. Multiple components may be composed to build other components. In the context of a visualization system, a component encapsulates one or more visualization algorithms, presentation modalities, or data representations.

The **component interface** is a set of methods supported by a component, and type definitions for the data used for arguments to those methods. An interface itself is a type and can be an argument for a component method.

A **component architecture** is a system defining the rules of linking components together. The CCA model of a component architecture is composed of the following elements An **Interface Definition Language** understandable to all components. Interface definitions expressed in this language allow components to find out about each other either through introspection or through consulting a repository, and give a component architecture the potential to dynamically add and delete components in multi-component applications (whether this potential is actually realized or not depends on a specific implementation of the architecture).

A **Domain Interface** is a set of programming interfaces that standardize interactions with a particular application domain. The Visualization community should:

1. Utilize domain interfaces that presently exist when possible or practical, and work with those communities to ensure that those interfaces are appropriate for use in the visualization community.
2. Develop new visualization-specific interfaces for visualization-specific tasks.

A **reference component** is a component that provides a commonly used functionality for a specific domain interface. In the context of visualization, examples of these may include colormap editors, resource management facilities, and so forth.

A **Binding** exists between the interface definition syntax and a language or framework of actual component implementation.

A **Composition API** allows the programmer to link components into multi-component applications and save those compositions. Such a mechanism could be provided for example by a GUI or a scripting language, and need not be standardized in a CCA.

A framework is a specific implementation of a component architecture.

A Common Component Architecture (CCA) is a component architecture defining standards necessary for the interoperation of components developed in the context of different frameworks. To date the need for three such standards has been identified: the Interface Definition Language, an interaction model and a set of services based on this model which can be expected by every component, and should be provided by every framework, and a standardized way of retrieving information from the repository.

Visualization Framework and Component Requirements

In this section, we present a number of requirements and objectives of visualization components and frameworks. In some cases, a given requirement applies to both components and frameworks, while in other cases, the requirement applies to just components or just frameworks.

1. **Framework.** The framework is a high-level, adaptive system that is capable of executing visualization components in dynamic environments on heterogeneous

resources, including resources that span multiple sites. Some components may be parallel, while others may be serial. The framework should also be resilient and responsive to error conditions, such as when a component unexpectedly dies.

2. **The Audience.** We view the community of visualization developers and researchers as being the primary audience, or adopters, of a visualization framework and component-based tools. Visualization developers can use the framework and component-based tools to create finished applications specially tailored for a given domain area. Similarly, visualization researchers can leverage the framework and services used to build components to create new components to test out new visualization research ideas.

3. **Interoperability.** Visualization developers should be able to share components, and expect that a component produced by one developer will interoperate with components produced by a different developer. Visualization developers can create domain-specific applications from collections of components that execute within the framework. Discipline scientists want finished applications that meet their particular needs, as opposed to a completely general-purpose tool.

4. **Technology Recyling.** A general objective for the framework and components is the ability to reuse software when creating new components. Such reuse occurs at many levels. An existing visualization application may be “wrapped” with a component interface, allowing to server as a single component in a larger application. Pre-existing applications like VTK can act as “components,” as well as small, lightweight single-purpose tools, like stream multiplexers. During specification, design and implementation of the framework and components, those involved with such activities should incorporate, where feasible, as much existing technology as is practical.

5. **Open Source Distribution.** The framework, components and toolkit elements should be distributed using an Open Source license. The ultimate objective is to foster an environment that promotes widespread use and encourages contributions and involvement of visualization developers and users.

6. **Common Data Structures, Formats and Models.** Components, which are the fundamental building blocks of applications, have, by definition, well-defined interfaces. The scope of “data” that flows through visualization component interfaces needs to be defined. Classes of data include things like scientific data (structured and unstructured grid), graphics and visualization data (images, geometry) and control data (related to the execution of a component). For practical purposes, we as a community must develop a limited set of commonly agreed-upon data structures that implement a range of domain-specific models. Because this is not a fully generalized data model, it will not initially be sufficient to cover all possible scientific application requirements, but will fully cover a large proportion of them. These data structures can be expanded over time to provide more general coverage.

7. **Presentation Independence.** The framework and components should be flexible enough to support display and interaction on a wide range of output devices. These range from handheld devices, desktop displays and segmented/immersive environments. Each of these different display devices exhibits vastly different characteristics in terms of amount of bandwidth required to drive the device, the amount of compute or render resources required to drive the device, and so forth. There is an implicit need for the ability to detect and respond to varying levels of resource requirements and levels to impedance-match required with available resources.

8. **Security.** Communications activities, both between components and between the framework and the components it manages, should be capable of occurring in a secure fashion. This means that connections can be authenticated and encrypted. Development of tools that implement control and data transport between components and between frameworks and components should leverage existing work in the field of secure communications.

9. **Resource Management.** Components should participate in dynamic resource allocation by exposing instrumentation and methods to allow an executive to extend execution to newly allocated resources. Such instrumentation can provide an estimate of resources a component requires to perform a given task. The framework should be capable of using such estimates to predict performance. Furthermore, the framework should be capable of selecting a different set of resources if the performance estimate exceeds a performance budget, or if the selected resources fail to deliver required performance; a situation referred to as "contract violation." This task will become more complex as a runtime analysis problem as variables in the visualization application change in response to user events, changing input data, changing performance constraints or changing environmental conditions. Performance estimates, modeling and runtime monitoring play an important role in "impedance-matching."

10. **Interactive Application Construction.** There exists a need for one or more user applications that permit run-time construction and execution of component-based applications that use the framework for component launching and execution control. Over time, domain-specific applications that realize similar objectives can be tailored for use in specific display and interaction environments.

11. **The Brooks 80/20 Rule.** When specifying, designing and implementing component interfaces, those involved with such activities should chart a path of graduated steps in interface development that is conducive to rapid development and implementation of early prototypes. The design, specification and implementation of component interfaces is expected to evolve over time to gradually encompass increasing levels of generality. We wish to avoid a common pitfall that requires a specification and design that is all-inclusive at the expense of demonstrable and rapid development and deployment.

12. **Location Independence.** The framework must provide abstractions that support location independence. This includes transparent mechanisms that serialize data for invocation of remote methods while still providing direct low-overhead connections for components that are co-located. Placement and launching of components in the distributed environment should be simple, not complex (effortless). There exist several frameworks in the CCA community that provide this abstraction.

13. **Transport Independence.** The framework should have modular components that provide abstract transport mechanisms for distributed objects (data or code). The abstract methods for data transport will support pluggable/modular replacement of transport methods depending on the transport medium (Myranet vs. TCP), security layers (encrypted or authenticated data channels), and serializations (XDR, XML or raw binary).

14. **Group Communication.** In addition to point-to-point communications and data transport, the framework should support multicast/multiplexed communications streams to support collaborative interfaces and deployment, possibly through interposing components.

The RDV Component-Based Framework Vision

The vision for an RDV component-based framework is to realize an interoperable set of visual data analysis capabilities. The building blocks of the framework are components, which perform operations on data objects. Data is communicated between components using framework libraries that move, encode, and decode data objects. Synchronous execution of components can be moderated by a central authority, or "executive." Alternately, the components might execute in a peer-to-peer fashion, depending upon a particular application's requirements. In the "centrally managed" configuration, the "executive" schedules execution of components, which might reside on resources at different sites. In the peer-to-peer configuration, the data connections and execution model might be procedural, thereby avoiding the need for a central "executive." The framework should be capable of supporting both types of execution architectures using the same fundamental building blocks. The figure below is a rough "first draft" that shows the framework and component elements, along with relationships in terms of control and payload data movement through the system.

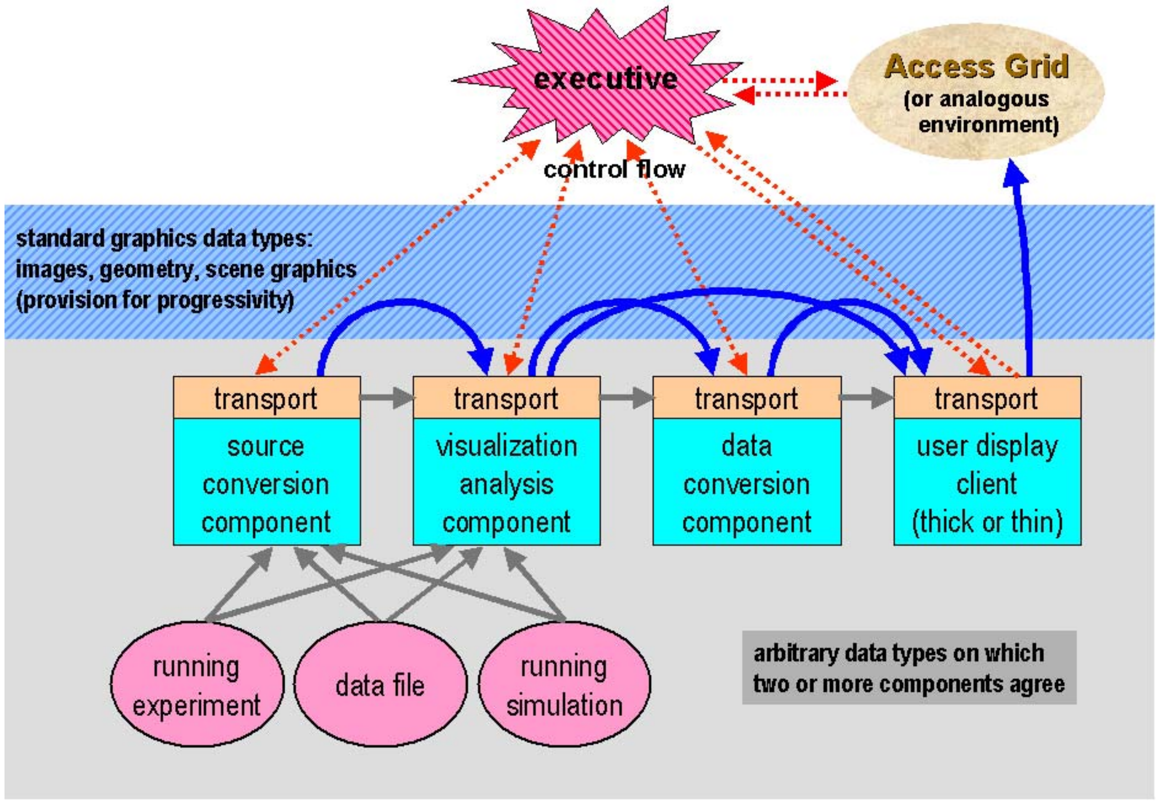
executive
Access Grid
(or analogous environment)
control flow
standard graphics data types:
images, geometry, scene graphics
(provision for progressivity)
transport
source conversion component
transport
visualization analysis component
transport
data conversion component
transport
user display client (thick or thin)
running experiment
data file
running simulation
arbitrary data types on which two or more components agree

Figure 1. Component-Based Framework Architecture Overview

Barriers

Some barriers to adoption, use and contribution to a component-based visualization framework are social in nature. For example, some are reluctant to use technology that was not created in-house, or the “not invented here” (NIH) syndrome. In some cases, NIH stems from concerns about not having in-house expertise in a particular piece of key technology. In other cases, intellectual creativity and productivity are the currency of domain expertise. A community-supported project helps to allay both sets of concerns. With strong community support, a critical mass of developers, users and advocates ensures the project will remain vigorous and promote longevity. The Open Source phenomenon bears out many examples: Linux, X11/Xfree86, Apache, and so forth. A stable framework actually makes possible the likelihood of increased intellectual productivity in specialized research areas, as key, fundamental technologies need not be implemented again and again at each different location. Furthermore, a framework that exhibits the characteristics we have discussed makes possible advances in new directions that might not have been possible without such a framework.

With a project such as we have described, one significant challenge is to define project scope. The scope of the project should balance between being all-encompassing and having sufficient breadth and depth to be useful. All-encompassing scope means that the framework and components have capabilities to all imaginable applications. Such expectations are not realistic, and adherence to such dogma will prevent forward progress on the project. On the other hand, a narrowly defined scope will satisfy only a few application needs, and not be useful to a broader audience. An appropriate project scope, which remains elusive at this time, must strike a balance between these two objectives.

A robust data model and common data structures are prerequisites for interoperable components. The notion of a completely general data model and interface has been the subject of much research activity over the years. To date, there is no completely general data model. The reasons are numerous, and to some extent, parallel the two barriers discussed above. Past efforts to develop fully general purpose data models have led to many implementations that exhibited poor performance characteristics or extremely complex APIs. Therefore, we will work on defining data structures that support a diverse set of applications in order to provide a domain-neutral substrate for components to exchange information. Failure to develop any commonly agreed upon data structures leads us, inevitably, towards a “Tower of Babel” where components developed by different groups will still be unable to interoperate. Development of a higher level, fully general purpose data model would be helpful, but not a prerequisite to guiding fundamental component and framework development. Further work on RDV frameworks and components must balance the objectives of working software, which is needed by researchers, with the desire to create completely general purpose tools and methods.

Next Actions/Recommendations

1. More effort is needed to further refine the scope of a component-based framework for RDV, and to identify the core technology components.
2. A first step towards realizing working components is the need to define data interfaces for visualization components. The scope of this task requires careful balance between generality and specificity, with an eye towards reusing as much existing technology as possible. Examples of data objects and access methods that need to be defined include, but are not limited to: geometry primitives (triangles, vectors, spheres, etc.), images, colormaps, fundamental grid/mesh objects, multiresolution representation and transmission, progressive (meta)-data interfaces/models, and so forth. The scope of the initial activities should be targeted towards a "minimum set" of capabilities so that work can proceed.
3. Closer collaboration is needed between those interested in component-based visualization tools and frameworks, and other groups working on data interfaces/models. These groups include but are not limited to: Field Model (NASA-Ames), TTST, GGF-ACE, Web3D Consortium, Earth Systems Grid, etc. Such collaboration should produce benefit to all groups.
4. While initial activities will be performed without formal funding, the long-term success of an RDV framework requires formal funding.
5. A small set of working components applied to an important science project would help to prove the efficacy of the design to stakeholders.
6. Given the high level of community interest and the sizeable project scope, additional workshops and meetings are required. We agreed to hold a meeting in conjunction with the CCA workshop to be held on July 10-11, 2003 in Eugene Oregon. We also plan to hold a smaller meeting sometime in June in the Washington DC area to further elaborate upon the scenarios described in these findings.
7. The issue of how we organize ourselves needs to be further refined. In the near term, subsequent meetings will be open by invitation only to others with similar goals. The goals are specification and implementation of interoperable visualization components within a common framework.
8. Create project website and email list at LBL.

Case Study 1 – Parallel Visualization of Large and Complex Simulation Results

To help define the scope of a visualization framework, one can walk through a specific application and enumerate the issues and expectations a developer may encounter in the realization of that case. An example visualization problem is a post-processing application that computes a custom derived quantity and displays the results in a custom GUI interface. Specifically, the result of the simulation is a set of 1024 HDF5 files containing a distributed set of particles with position and energy and a "boxlib" AMR mesh with a boundary field. The particle data is assumed to exist in the physical domain of the AMR mesh.

We wish to run a parallel visualization application on a distributed Linux cluster that uses a maximum likelihood classifier to cluster the particles into energy bins. The output should be a plot of the classification and an interactive display on the local user's desktop of an isosurface of the boundaries between the clustered particles. It is assumed that all the data files that make up the dataset are accessible by at least one of the nodes of the cluster. The example application goes through a number of concrete steps in accomplishing this task. We will walk though these various steps and call out the use and expectations of the application on the visualization framework throughout the execution of the application.

Initialization/Startup

The application needs to bring itself up in parallel on the cluster. To make this happen, the application relies on the framework's "Environmental Services". These services include resource management, job scheduling and initiation. Once the application is running, the framework will be relied on to manage secure communications and data transport between the nodes.

Data Selection

Once the application is running, it needs to identify the datasets to load. This is done through a set of "Data Manager/Registry Services". These services identify the available datasets, provide for data staging (e.g. from tertiary storage) and metadata queries. The framework must, at this level, be aware of data entities and the general nature of underlying data (e.g. variables, sequences, interpolators, etc). The application then uses the framework "Data Access Services" to attach to the selected data and instantiate all the necessary data access adaptor components, making it possible for the application to read selected portions of the data.

Data Access

At this point, that application needs to read the actual particle data for the cluster analysis. The specific access patterns could very substantially based on the target algorithm. Applications intend to leverage various iteration mechanisms to access the data (e.g. block access, streaming, random, etc). The framework is expected to provide a number of interfaces to various distributed data access patterns. The application is free to use these components or provide their own iteration interfaces. In this application, the application will use a statistical mechanism to compute the fit, so a random access pattern will be user.

Computation

The example application provides the core fitting algorithm. It walks through a sample of the particles and computes a fit and classifies each individual particle. While no new framework features are used for this, it is expected that the framework will provide a number of componentized transform mechanisms and operators packaged as utilities and extensions. The result of the classification operation takes the form of a new variable. The framework is expected to provide a mechanism for creating a new distributed variable (or tagging an existing variable) within the context of this application.

The "Plot"

The application is now ready to provide its first output, a 2D plot. It is expected that the framework would provide a "Plotting Interface" as a set of utility components. The plot is expected to be displayed into a GUI perhaps developed outside of the framework itself, using common graphics rendering APIs. Additionally, the application will expect that the plotting interface would be capable of providing high quality (e.g. PostScript) output as well.

Boundary Contour

The final operation our application needs to perform is the generation of a boundary surface between the particle clusters. This is done in two steps. The first is to create a new field on the same domain as the AMR mesh whose values are a composite of the classification of the particles. Then, the new mesh is contoured to generate a polygon mesh. The application relies on two features of the framework to make this happen. The first is the "Data Operator". This is used to "deposit" the particles on a new mesh. The framework is used to extract the mesh domain from the AMR grid and realize a new, distributed mesh. The application can use the mesh localization data operators to compute the location of each particle in the new mesh and then update the mesh data values according to the particle classification. Ideally, common operations (e.g. the "deposit" operation) would be available directly in the framework, however, the lower level data access operators would enable the application to write such a function.

The second framework feature is the collection of "Data Filters/Transforms". These components are used to generate new data objects from existing ones. In the example, a contouring object is used to compute a polygon mesh from the particle classification mesh, perhaps sampling the fields from the original AMR mesh field in the process as vertex data. It is assumed that the framework implementation of distributed mesh data includes intrinsic support for "ghost zone" padding in the context of known interpolant footprints.

Interactive Display

The application now relies on the framework to provide a number of basic interfaces to render and manipulate the generated polygon mesh. The "Rendering Interface" is part of the framework core and provides the basic mapping to rendering APIs for target objects (e.g. a polygon mesh). It allows for display of the surface interactively via OpenGL or to a PostScript file. The "Interaction Interface" is part of the framework core and provides a means for interacting with and manipulating the attributes of a rendering via a (possibly internal) abstract event stream. As in the plot example, the "Presentation Interface", which is a collection of utility components is utilized to present the user with an interactive GUI through which they can manipulate the resulting display on their desktop.

In this specific example, the application makes use of a declarative execution model. Ideally, the framework would provide an alternative "pipeline" execution model, but it is important for many applications that this not be the only method for execution control available to an application.

Case Study 2: Response to a Hurricane Threat

[NB: While the value of real-time collaboration should be obvious in the following scenario, we ask for an a priori acceptance of a rationale for the use of 3D visualization techniques for storm volume and terrain, rather than a simplified 2D GIS view, and further, that collaboration services are sufficiently reliable for civil disaster response management]

A tropical storm off the Yucatan Peninsula has intensified under the influence from another tropical depression in the western Gulf, changed track, and is moving rapidly toward the Gulf coast of Louisiana. Quickly reaching hurricane status, it poses a severe threat to the Mississippi Delta, Gulfport and New Orleans, which are already threatened by flooding from the heavy rain of an Atlantic storm that crossed northern Louisiana during the previous week.

The NOAA National Hurricane Center, in conjunction with the National Severe Storms Laboratory have contacted the LA State Office of Emergency Management, and initiated the Severe Storm Threat Analysis and Response (SSTAR) protocol.

For this situation, a team of simulation scientists, meteorologists and emergency management personnel are quickly assembled in an Access Grid Virtual Venue. The team will use current observational data from satellites, ocean sensors and overflights as input for atmospheric storm simulation and deep and shallow ocean circulation simulations running at NCAR and NERSC. These simulations will provide prediction of storm severity and track, wave action and energy and tidal flow. Continued monitoring of observational data will provide simulation correction steering and allow error analysis.

In addition, emergency management personnel identify areas of high vulnerability due to current population and infrastructure status – a bridge has been damaged by a barge collision with one of its piers, and, in another area, there is construction work on a major highway, and the only detour passes through low coastal elevation. Both the highway and bridge are on principal evacuation routes, and early storm damage prediction will permit logistical decisions in evacuation and response management. By identifying these areas of high vulnerability (simulation interest), the team is able to make decisions to narrow a parameter search space for quantifying early predictors of local storm conditions (tidal surge, wave energy and wind conditions). Using this information, the emergency management officials will be able to make decisions to focus response efforts in areas of highest risk.

Assembled in an Access Grid Virtual Venue, the team comprises scientists located at NCAR, NERSC and NOAA, and state emergency management officials in Louisiana. NCAR, NERSC and NOAA have large-format Access Grid spaces, with high-resolution, tiled panels at NERSC and NOAA. The LA state emergency management office has a workstation-based Access Grid node.

The team begins the simulation process by defining the model components, input and scope of the simulation. This process is supported by 2D weather map and observational visualization. The visualization application executes at NCAR, with user interface and rendered output multicast to participant locations via framework transport services. U/I control is multiplexed (without explicit locking or arbitration) from input events received from NCAR, NERSC and NOAA. All event transmission is handled by framework services. Data format, processing, visualization, rendering and viewer are the responsibility of the application; however, application components have been developed to the component interface specification, and are reusable within the constraints imposed by the domain data models. The initial visualization application is invoked through an executive dataflow network configuration tool and UI provided by the framework (provided as part of the basis framework execution toolkit), using resource descriptions provided by the domain grid (virtual organization). The resource management and access functions of the executive are built on available facilities and extensions of existing grid frameworks for resource discovery, scheduling and access. Provisions for authentication and security are assumed to be provided by the underlying grid framework.

When the simulation scope is identified, parameter characterization is provided to application component-specific resource estimation plugin objects (developed under the framework interface specification) executing in the executive. The executive then determines best network paths and identifies candidate computational and rendering resources. Given that this is a civil emergency, the simulation scientists can use the framework to select resources for pre-emptive scheduling, rather than the more typical advance-reservation scheduling or best-effort immediate scheduling also support supported by the framework.

Software components include:

- Atmospheric (storm) simulation
- Ocean simulation, coupled with Atmospheric simulation (using surface wind velocity as input) – uses shore data as boundary for shallow water model
- Software direct volume rendering of atmospheric data (a la Visapult, providing six-axis base plane images of blocked volume for each time step). This renderer supports input of the data format used by the storm simulation, which is assumed to be a domain standard format specified by the Earth Sciences Grid.
- Viewer/visualization modules instantiated at each display site, providing view-transformed display of composited volume, and "embedded" geometry (not truly embedded, as only block base planes are available, but as the geometry forms an approximate boundary for the storm volume it will arguably provide an acceptable effect) describing land topography and strategic shoreline and infrastructure features (bridges, buildings, jetties, dikes, levees, etc.), with visualization of water level superimposed on geometry, and wave energy and flow velocity displayed at shoreline and infrastructure features. These modules comprise components constructed with substantially borrowed code (e.g., OpenRM) with interfaces conforming to framework specification, integrating framework performance instrumentation and connected through transport abstractions to underlying transport components. The workstation node

instantiates a more lightweight viewer module that may eliminate the storm volume data from the scene graph, and uses more conservative progressive resolution facilities to meet performance constraints in rendering the geometry data.

- Interposing stream multicast and event multiplexing modules. These modules provide minimal framework support for distributing 2D user interface display, streaming of image data produced by 3D render tasks (in this case, the image data produced by the software volume renderer, but this component is more generalized, containing an H.261 encoder for straight readback/encode/stream of successive images) and collection and multiplexing of events for the rendering task (e.g., mouse, keyboard)

In this system realization, dedicating a final render engine to each display permits local control of the visualization, allowing independent cameras, data selection (e.g., removal of storm volume from draw list, playback of selected observation data, quantitative comparison of selected observation and simulation data, etc.). The multiplexed input event stream (multiplexed by the receiver with events from the local hardware in order received, as events are multicast by all participants) permits viewers to switch to a shared control mode. This switch is a U/I control exposed to the viewer application by the multiplexing component. This change in control state also requires a synchronization of the scene graph view and draw state in order to correctly set shared view. The master's state is multicast in messages through the event transport interface.

This scenario presents a plausible use for collaboration facilities provided by the framework and associated facilities, and The Access Grid or its equivalent is a necessary adjunct of the visualization system in supplying the audio and video teleconferencing facilities that provide much of the intellectual bandwidth for the problem.

With minor changes, this scenario could describe an effort to analyze and plan a response to threat of radiological, biological or chemical terrorism. Weather and local flow simulations (the latter in the case of complicated boundary geometry, as could be used to model dispersion patterns, and could be coupled with epidemiological models for predicting casualty patterns.

Workshop Participants

John van Rosendale, DOE-HQ
Wes Bethel, LBL
John Shalf, LBL
Randy Frank, LLNL
Dean Williams, LLNL
John Clyne, NCAR
Jim Kohl, ORNL
Steve Parker, Utah
Joel Welling, PSC
Pat Moran, NASA-Ames
Ron Kriz, Virginia Tech
Ken Joy, UC Davis
Sam Fulcomer, Brown

Further Reading

Zoltan: (Adaptive runtime load-balancing toolkit for distributed applications) K. Devine, B. Hendrickson, E. Boman, M. St.John, and C. Vaughan. “Design of Dynamic Load-Balancing Tools for Parallel Applications.” Proceedings of the International Conference on Supercomputing, Santa Fe, May, 2000. http://www.cs.sandia.gov/~kddevin/Zoltan_html/

CCA: R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, B. Smolinski. “Toward a Common Component Architecture for High-Performance Scientific Computing,” Proceedings of the Eigth IEEE International Symposium on High Performance Distributed Computing Conference, Redondo Beach, California, August 3-6 1999.

CCA Forum homepage: http://www.cca-forum.org

BABEL: (A tool for generating multi-language wrappers from CCA specifications. This goes with location independence but also covers the issue of “language independence” of the components, something we alluded to in the meeting but didn’t address directly in this findings document). http://www.llnl.gov/CASC/components/babel.html

Alexandria component Repository: (A central repository for community developed CCA components. A good reference for the section about “community sharing, etc.” http://www.llnl.gov/CASC/components/alexandria.html

CAFFEINE: (A CCA framework for SPMD/data-parallel applications) B. Allan, R. Armstrong, A. Wolfe, J. Ray, D. Bernholdt, J. Kohl, “The CCA core specifications in a distributed memory SPMD Framework,” Concurrency and Computation, Practice and Experience, vol. 14, 2002, pp 1-23. http://www.cca-forum.org/ccafe/

XCAT: A CCA framework for distributed applications http://www.extreme.indiana.edu

XCAT Science Portal: (An example of a web interface to distributed vis software - display modalities.) S. Krishnan, R. Bramley, M. Govindaraju, R. Indurkar, A. Slominski, D. Gannon, J. Alameda, D. Alkaire, “The XCAT Science Portal,” Proceedings of SC 2001, Denver Colorado, November 10-16, 2001.

GrADS (Example of a software framework that implements runtime monitoring/performance-predition/deployment of distributed applications.) F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolski, “The GrADS Project: Software Support for High-Level Grid Application Development,” International Journal of High Performance Computing Applications, Winter 2001(Volume 15, Number 4), pp 327-344. http://www.hipersoft.rice.edu/grads/publications/ .

AutoPilot: (More applications adapting to runtime/environmental conditions.) R. Ribler, D Reed, “The Autopilot Performance-Directed Adaptive Control System,” Proceedings of 11th ACM International Conference on Supercomputing, Vienna, Austria, July 1997.

AppLeS: (Distributed visualization tool launching and resource management.) A. Su, F. Berman, R Wolski, “Using AppLeS to Schedule a Distributed Visualization Tool on the Computational Grid,” Proceedings of the 1998 Clusters and Computational Grids Workshop, 1998.

More Resource Management (the MDS) S. Fitzgerald, I. Foster, C. Kesselman, G. VonLaszewski, W. Smith, S. Teuke, "A Directory Service for Configuring High-Performance Distributed Computations," Proceedings of HPDC 6, August 1997, pp 365-375.

CUMULVS: (A fault-tolerant/distributed visualization framework.) G. Geist, J. Kohl, P Papadopoulos, "CUMULVS: Proving Fault Tolerance, Visualization and Steering of Parallel Applications," The International Journal of Supercomputer Applications and High Performance Computing, vol. 11 #3, Fall 1997, pp 224-235.

Collaborative Software/CaVERNSoft: An abstraction layer for "group communication." Park, K., Cho, Y., Krishnaprasad, N., Scharver, C., Lewis, M., Leigh, J., Johnson, A., "CAVERNsoft G2: A Toolkit for High Performance Tele-Immersive Collaboration," Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2000, Oct 22-25, 2000, Seoul, Korea, pp. 8-15.

TSTT:(data models) http://www.tstt-scidac.org/

FEL: Field Encapsulation Library (data structures) http://www.nas.nasa.gov/Software/FEL/

Field Model: (More data structures/data formats.) http://sourceforge.net/projects/field-model/

IBM DX Data Model: http://www.research.ibm.com/people/l/lloydt/dm/dx/dx_dm.htm

IBM Function Based Data Models http://www.research.ibm.com/people/l/lloydt/dm/function/dm_fn.htm

Archive of Fiber/vector-bundle data models http://pueblo.lbl.gov/~olken/fiberbundle.html.

Advanced Collaborative Environments – Global Grid Forum (ACE-GGF): http://calder.ncsa.uiuc.edu/ACE-grid/.

ACE-GGF (Security document): Jason Leigh, Brian Corrie "Application Security Requirements of Tele-immersive Environments," GGF Draft Document, May 31, 2002.