

A Distributed Blackboard Architecture for Interactive Data Visualization

Robert van Liere

Jan Harkes

Wim de Leeuw *

Abstract

In this paper we present a distributed blackboard architecture for scientific visualization. We discuss our motivation, design and application of the architecture. The main advantages of the architecture is twofold. First, it allows visualization tools to be tightly intergrated with simulations. Second, it allows qualitative and quantitative analysis to be combined during the visualization process.

1 Introduction

The need for enhanced data modeling and data integration in visualization environments has been widely recognized and has been a topic of interest in many workshops and conferences [1, 2]. Although visualization environments provide rich support for graphical modeling and rendering, many visualization researchers feel that integration of the visualization environment with the simulation environment remains a problem to be solved.

In this paper we present a distributed blackboard architecture for interactive data visualization. The architecture has been used in various projects related to high performance computing [3] and computational steering [4].

Motivation The motivation for this work was triggered by three observations:

1. State-of-the-art scientific visualization tools are not sufficiently intergrated into simulation environments. For example, to support computational steering tight intergation is required. Computational steering allows the user to investigate intermediate results and permits the modification of the computation before completion. In many problems, feedback during the investigation may be required, for example to steer computation around local minima. Users may interactively steer simulations through adjustment of an application's critical parameters.
2. Large data sets defy inspection of by visualization alone. Analysis of simulation output often consists of a combination of visual inspection and numerical or statistical analysis. Scientific visualization environments lack the integration of general purpose analysis tools. Visualization combined with interactive analysis will allow the scientist to explore the data both visually and numerically. Combining the two allows the user to perform qualitative and quantitative analysis simultaneously. In addition, the output of the analysis tools can be combined with the visualization of the simulation data. Analysis tools should either operate autonomous or be manually driven by the user.
3. Efficient support for very large data sets resulting from high performance simulations is limited. Very flexible data models have been defined that allow import/export data from/to the simulation environment [5, 6]. However, in order to effectively embed these data models in the visualization environment, two system related issues have to be addressed; scalability of data, and access to remote computations.

*Center for Mathematics and Computer Science CWI, Amsterdam, the Netherlands, {robert1 | jaharkes | wimc}@cwi.nl

Many visualization researchers believe that state-of-the-art visualization environments do not adequately address these problems. For example, Foley and Ribarsky point out in [7] that next-generation visualization tools require, amongst others, a means to bind data to graphics and a general analysis model.

Blackboard Models Blackboard models have been widely used in the AI community as a particular kind of problem solving model [8]. The blackboard model allows multiple independent agents (usually called knowledge sources) to share information in a central store (called the blackboard). The model serves as a schema for organizing reasoning steps and domain knowledge to construct a solution to a particular problem. For example, in a forward reasoning model, problem solving begins by reasoning forwards from initial data towards a goal. In this case each knowledge source will contribute towards the goal.

Knowledge is segmented into modules and a separate inference engine is provided for each module. Communication between modules is realized by reading and writing in the blackboard. The blackboard can be partitioned so that it contains regions with differing, but perhaps related, data structures. In this way applications can organize the solution space into one or more application dependent hierarchies.

The blackboard model does not explicitly specify a control component. It merely specifies a general problem solving behavior. The actual locus of control can be in the knowledge modules, in the blackboard itself, in a separate module, or in a combination of these.¹

The difficulty with this description of the blackboard model is that it only outlines the organizational principles. For those who want to build a blackboard system, the model does not specify how it is to be realized as a computational entity. However, given a problem to be solved, the blackboard model provides enough guidelines for sketching a solution. We believe that the blackboard model can be used as a framework for future visualization solutions; i.e. a framework to iteratively and interactively gain insight into data.

Overview A simplified overview of the architecture of the environment is shown in figure 1. The environment provides an interface between a user and a simulation. The architecture is centered around a *blackboard*

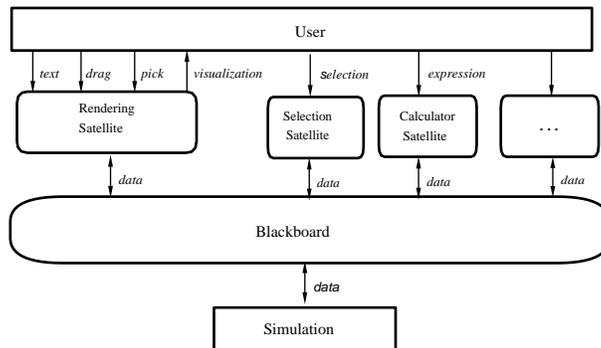


Figure 1: An overview of the centralized blackboard

and *satellite* processes that produce and consume data. These satellites implement the simulation, analysis programs, geometry mapping and rendering algorithms. The purpose of the blackboard is twofold. First, it manages a database of variables. Satellites can create, open, close, read, and write variables. Second, it acts as an event notification manager. Satellites can subscribe to events that represent state changes in the blackboard. Whenever such an event occurs, it will publish the event to all subscribing satellites.

¹Initial proposals considered the blackboard only as a passive memory with external control modules that monitored the changes in the blackboard. Later – efficiency related – refinements integrated the control modules into the blackboard.

A large collection of general purpose satellites have been developed. For example, a data slicing satellite, a calculator, a data logger, a VTK satellite that provides all VTK functionality [9], etc. Also, a 3D interactive editor and rendering satellite that binds data to geometry has been developed [10].

The format of the paper is: In the next sections we present some design details of a distributed blackboard architecture. The distributed blackboard extends the centralized blackboard in many ways; it provides support for efficient data movement among heterogeneous hosts, scales for large data sets, and offers a richer data model. Although the architecture is distributed, it is important to note that the programming model is not affected; i.e. the programmers view of a blackboard is a centralized data store. In section 4, we discuss an application of how general analysis tools can be integrated into the visualization process using the blackboard architecture.

2 Distributed Blackboard Architecture

In this section we present the architectural details of the environment. First we discuss the basic blackboard object : the variable. A variable encapsulates all information required to access a blackboard object. Then we present the system architecture and discuss synchronization and programming abstractions. Finally, we discuss the pros and cons of this blackboard architecture.

2.1 Variables

Variables are defined as a tuple consisting of of four components: a name, a type descriptor, raw data and a list of attributes (see figure 2). Names uniquely identify the variable. The variable descriptor determines the type, size

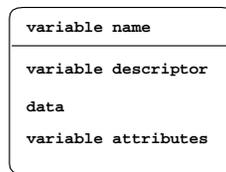


Figure 2: Four components of a variable.

and layout of the data. The data component is the storage container for the raw data. Attributes are name value pairs that may be used to describe meta-data of the variable.

The underlying data model supports data of two types:

- *regular topology*. Data which has been generated from the following grid types; i.e. uniform, rectilinear and curvilinear grids.

The regular topology data type is very similar to the data model supported by NetCDF [11]. In this case, the type descriptor contains all information concerning the shape and dimensionality of the variable.

- *geometry lists* A geometry list is a list of geometric elements. Each element can be a simple geometric object, such as polygon, polyline, mesh, light, camera, etc.

The the functionality offered by the geometry list is very similar to that offered by the low level interface of P3D [12].

Future extensions to the data model will include explicit support for data of irregular topology and scattered data. Note that these extensions do not affect the semantics of the variable, but only the expressiveness of the underlying data model.

The type of the variable may not change during the life of a variable. However, in the case of a multi-dimensional variable, the size may vary. Similarly, the length of a geometry list and size of each geometric element may change.

Operations on variables are very similar to low level file operations: create, open, close, read, write, and get-descriptor. Simple set/get operations are available to manipulate attribute lists.

Two scatter/gather techniques are supported to optimize I/O operations on variables. First, many variables can be read/written simultaneously in one atomic I/O operation. Second, a comprehensive data mapping mechanism is provided to allow data to be sliced, subsampled, etc during the I/O operation. This allows applications to use a different data layout compared to the data structure stored in the blackboard. The identity mapping allows a one-to-one copy between storage in the application and data manger.

2.2 Architecture

In this section the architecture of the system is discussed. First, we briefly define each building block of the system. Then, we show how these building blocks are combined to form the architecture. Finally, we discuss the pros and cons of this architecture.

Building blocks The architecture consists six building blocks: a global name manager (denoted as *GNM*), zero or more local blackboards (*LBB*), zero or more satellites, command, event and data streams.

- **Global Name Manager:** The GNM maintains the bookkeeping information of all LBBs and variables in system. The GNM maintains a list of all variables in the system and in which blackboard these variables exist. Only the variables names, descriptors and attributes are stored in the GNM. Variable data is *not* stored in the global name manager.
- **Local Blackboard:** A local blackboard resides on each host in the distributed environment. Local blackboards accept connections from satellites executing on the same host and other LBBs.

Variable data is stored in the LBB, and is shared by all connecting satellites. Each LBB maintains a copy of the variable data.

The LBB manages only those variables that are opened by the connected satellites. When a satellite opens a variable, the LBB consults the GNM to check if the same variable exists in other LBBs. If this is the case, the LBB will connect with these LBBs. This connection is used to maintain variable consistency (we explain variable consistency in the next paragraph).

- **Satellites:** A satellite is a process which communicates with its corresponding LBB. Satellites may create, open, close, read/write, and subscribe to variable events.

An abstract satellite is shown in figure 3. Basically, it consists of an operator that transforms input data into output data. Control determines when this operation has to be carried out, or, in other words, when a satellite is triggered. Operators are also controlled by an additional set of parameters, manipulated via user interface widgets.

Data input and output is performed by read/writing variables. Input and output triggering is discussed in section 2.4.

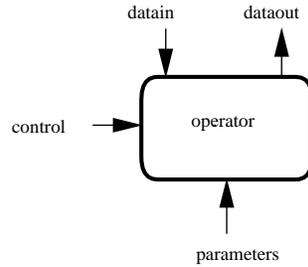


Figure 3: Interfaces to an abstract satellite.

- Command, event and data streams: A connection between a satellite and the LBB consists of a command, event and data stream. Communication to the LBB is done by sending commands on the command stream. Communication from the LBB is done by receiving events from the event stream. Data streams are used to transport data between LBBs and satellites..

Distributed Architecture Figure 4 shows a example configuration of the architecture. This configuration shows two LBBs and four satellites. Both local blackboards are always connected to the GNM. The local blackboards share a variable, hence they are connected.

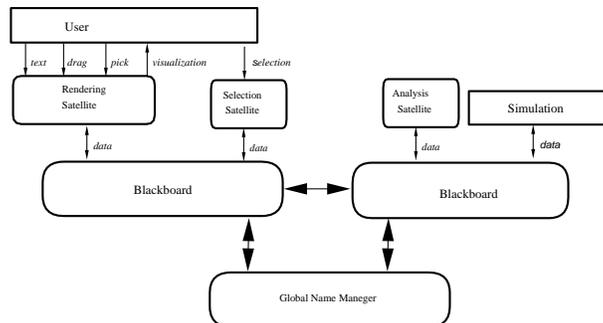


Figure 4: Distributed blackboard architecture.

Satellites execute in parallel but LBBs are single threaded, so that only one satellite can access the LBB simultaneously. However, access to different LBBs is concurrent.

When a satellite writes a variable, the LBB will broadcast a mutate event to all connected satellites and LBBs that share the variable. When a satellite reads a variable, the LBB will first check if the data is up to date and, if so, will service the read request. If the data is not up to date, the LBB will first get the latest copy of the data from another LBB before servicing the read request. The details of this algorithm are very similar to the cache consistency algorithms found on cache based SMP machines.

2.3 Programming Abstractions

Satellite programmers can access the LBB using three layered APIs. Each layer provides a higher level of abstraction. Higher layers are easier to use, but do not provide all functionality provided by the lower layer.

1. The local blackboard API is a low level library interface which provides functionality for LBB communication, variable management and event management. This layer provides all details of the underlying LBB protocols. It requires detailed knowledge and is difficult to use due to the inherent parallelism in the system.
2. The data input output layer built on top of the local blackboard API and provides a high level access to the LBB. Many cumbersome low-level details are shielded from the user. In particular, this library hides the notion of events and has builtin support for structuring variables into sets, and support for handling efficient set I/O.

A design goal of this layer was to make the required changes to the simulation code are absolutely minimal. As an example of this library, consider the following C program:

```
simulation(float *s, float *t, int *size, float *x)
{
    int continue = TRUE;

    /* Open connection, connect and subscribe variables */

    dioOpen("borneo.cwi.nl");
    dioConnectFloat("s", s, READ);
    dioConnectInt("continue", &continue, READ);
    dioConnectFloatArray("x", x, 1, size, UPDATE);
    dioConnectFloat("t", t, WRITE);

    /* simulation loop and update data */

    while (continue)
    {
        t = t + 1.0;
        calculate_values(t, s, size, x);
        dioUpdate();
    }
    dioClose();
}
```

The structure of this example, which is typical for a continuous simulations, consists of two parts. First, variables are initialized. The required changes to an existing source code are limited to opening and closing a connection with the Data Manager and connection of the variables via the `dioConnect` routines. Second, a main loop is entered where time is incremented and new values are calculated. The required changes to the source code is a single call to exchange data. The locations where to insert these calls are easy to find: typically at the outer level of the simulation program.

The first parameters of the `dioConnect` routines are the name of the variable and its address. For the connection of arrays the number of dimensions and their sizes must also be specified. The last parameter describes the direction of the data flow. This information is used by the `dioUpdate()` routine to decide what must be done. In `dioUpdate()` first the event stream from the LBB is checked if variables to be read or updated have changed. If so, these variables are read from the LBB. Next the values of all unread variables are written to the LBB. The net result of `dioUpdate()` is that all connected variables have the

same value in the simulation and LBB. With these few calls the user can interact with parameters (s) of the simulation, to stop the simulation (`continue`), monitor its progress (t , x) or change state variables (x). To deal with more hierarchical situations, variables may be grouped into sets. In the main loop the application can read and write specific sets, and wait until a particular set changes.

3. A extensible and embeddable scripting language built on top of the data input output. Scripting can be used for simple operations on variables, such as slicing and logging. The advantage of scripting is its ease of use.

2.4 Satellite Control and Synchronization

Satellites cooperate via the basic input/output mechanisms that are provided for variables. Writing to a variable will cause an event to be sent to all satellites subscribed to that variable. This mechanism is used in two ways. First, the user can specify that only if one particular variable, the *input trigger variable* is changed, the operator has to be re-evaluated. The action of operator re-evaluation is called *triggering*. Second, if no such trigger variable is specified, then upon each mutation of any input variable the operator is re-evaluated. In this case, the satellite will subscribe to all its input variables.

Further, the user can also specify an *output trigger variable*. This variable is written to each time the operator has been re-evaluated, and can be used to link the control flow for satellites.

A trigger scripting language has been developed to simplify the definition of trigger variables. The scripting language allows users to define trigger variables and specify how these variables link two different trigger variables together. When one satellite writes the output trigger variable, it will trigger the satellite owning the second variable. A trigger satellite has been developed that interprets the trigger script.

As an example, consider the following configuration. A slicing and dicing satellite operate on simulation output, which in turn is the input for the viewing satellite. The simulation may only compute the next step after the viewing satellite has drawn the previous frame. The viewing satellite depends on variables computed by the simulation and dicer. The script to realize this configuration is:

```
simulation > slicer
slicer > dicer
simulation & dicer > viewer
viewer > simulation
```

This script defines the configuration shown in figure 5.

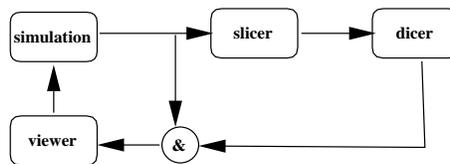


Figure 5: Control loop of four satellites defined by a synchronization script

2.5 Discussion

The architecture has been implemented on a number of heterogeneous UNIX and NT machines. TCP/IP is used for across host LBB communication and, when possible, shared memory is used for satellite to LBB communication.

In addition, viewers are available for many display, ranging from low-end laptop displays to sophisticated VR displays like the CAVE.

In retrospect, we believe that the architecture provides support to fulfill our major requirements:

- *integration*. The variable naming mechanism is used to bind data structures in the blackboard to data structures in the satellite. The event mechanism is used to maintain the consistency of these data structures. The net effect is that, for example, a viewing satellite can directly manipulate data structures of other satellites. Hence, computational steering is supported. The integration is not restricted to the simulation and visualization alone, but any number of analysis tools can connect as satellites.
- *efficiency of data transport*. Distributed blackboards maintain local copies of a data structure. A single event will be broadcasted when a satellite mutates the data structure. The data structure will be transported to another blackboard only when it is needed. This mechanism, which we call 'transport by demand', saves bandwidth when data structures are often read.
- *ease of use*. Using the low level libraries require knowledge about event driven and parallel programming abstractions. However, higher level libraries shield all these details, and allow a programmer to easily bind Fortran data structures to variable store in the blackboard. In this way existing simulation code can very rapidly be intergrated into the environment.

Also, programmers need not know that the blackboard is distributed. The abstractions provided by a variable are very similar to the familiar file abstractions.

3 Related work

Many research and development teams have designed and implemented interactive visualization environments. Giving an in depth analysis of other visualization environments is outside the scope of this paper. Instead, we discuss only some issues that resemble our blackboard architecture.

Williams, Rasure and Hanson [13] provide a framework to understand design tradeoffs when developing data flow based visualization systems. Data flow systems are attractive because of the similarities with the visualization pipeline: users can easily organize their simulation, filter, mapping and render modules in an intuitive way. However, data flow environments do not provide support to deal directly with the underlying data, except for importing/export data from a file or simulation. Hence, the integration with the underlying data is limited.

CAVEvis [14] is a distributed real-time visualization system for streaming large scalar and vector fields into the CAVE. The governing idea is to render the geometry as fast as possible in order to maintain the highest level of interactivity. It uses separately running modules to asynchronously generate the geometry data, such as modules to generate isosurfaces or particle paths. Sequence names and time-stamps are used to gather data related to a frame, regardless of the order in which the data is received. Our blackboard architecture does not explicitly support any form of sequencing and control. Rather, the synchronization is used to provide similar functionality.

SuperGlue [15] is programming environment for scientific visualization whose main goal is extensibility and ease of use. The approach used is to offer a very generic set of primitive data structures and a inter-language interface, which programmers use to integrate data into SuperGlue system.

4 Radon Forecasting

We applied the distributed blackboard architecture to an atmospheric transport application. In this case the goal is to accurately forecast the atmospheric transport of the radioactive noble gas radon-222 (^{222}Rn) based on measured

wind and emission fields. In this particular study the concentration of radon over the Indian ocean was tracked. In the hope that systematic simulation errors can be found, researchers are interested in comparing simulated radon concentrations with actual measured radon data. Simulation errors can arise from modeling errors, numerical errors, visualization errors, and input errors. In this case, however, the main interest are errors due to incorrect input values; e.g. incorrect emissions, incorrect wind fields, etc. In addition, we consider only systematic errors that occur due to a combination of:

- spatial errors: geographical locations of the simulated data differ from the measured data.
- temporal errors: the simulated data differ from the measured data in time.
- scaling errors: the simulated data is systematically higher or lower than the measured data; eg. due to an inaccurate emission sources.

Various automated data analysis techniques have been developed that search for regions in the simulated data that fit the measured data. Details of these techniques, which are based on statistical comparison and fitting methods, are outside the scope of this paper.

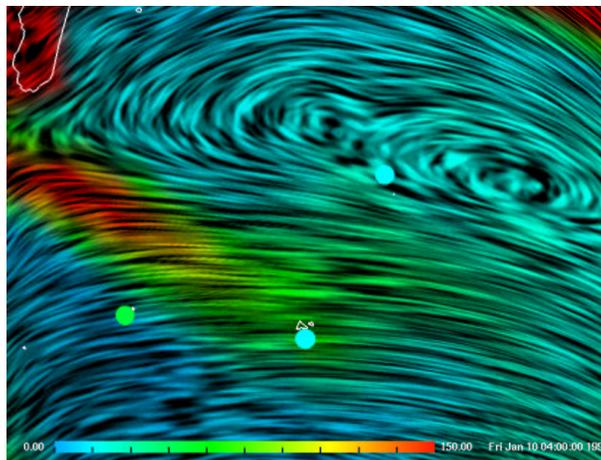


Figure 6: Radon concentrations over the Indian ocean. Small colored circles show measured sites data.

Visualization Figure 6 gives an overview of the ongoing Radon transport simulation over the Indian ocean. Spot noise was used to display the wind fields. A rainbow color mapping was used to display the Radon concentrations and small colored circles showing the measured concentrations are drawn on the three sites where measured data is available. The three sites are located at: Amsterdam Island (77 deg 34' E, 37 deg 50' S), Crozet Island (51 deg 52' E, 46 deg 26' S), and Kerguelen (70 deg 15' E, 49 deg 21' S), all on French territory. To the right of figure 6 shows a 1D-plot of one month of simulated data and measured data at Amsterdam Island.

Figure 7 shows a sequence of snapshots of the automated point fitting process. Semi-transparent circles are drawn at points calculated by the data analysis techniques. The opacity of the circles is mapped to the fit of the data. Transparent circles indicate points of poor fit; opaque circles indicate points of better fit. The left image of figure 7 depicts the points in an initial configuration around Amsterdam Island. A new configuration is derived by deleting a poor fitting point and taking for a new point at a random position close to the best fit point. In this way

the process converges to a minimum, the area which best fits the data. The middle image shows the configuration after a number of steps. Finally the points may converge to a stable configuration, as indicated on the right image.

The plot to the right of figure 7 shows the simulated data of one of these points. The data of the plots is a combination of computed data and measured data.

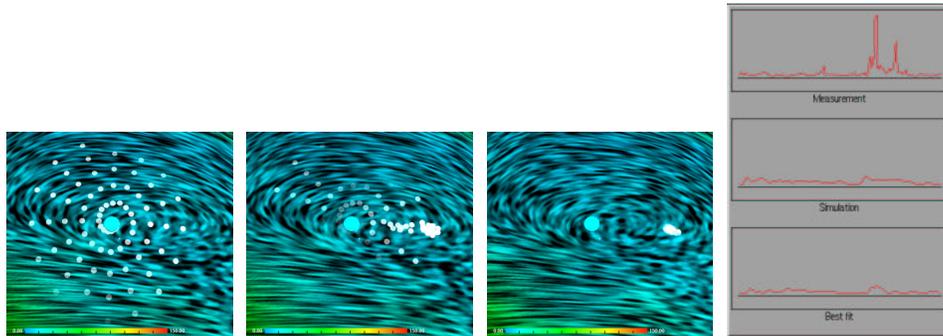


Figure 7: Comparative time sequence of converging point set.

The user may at any time also edit the set of points by dragging any point to a different location. This is useful if the user suspects other local minima in the data which may be missed by the analysis software.

Blackboard Figure 8 is a diagram of the blackboard and the satellites around it.

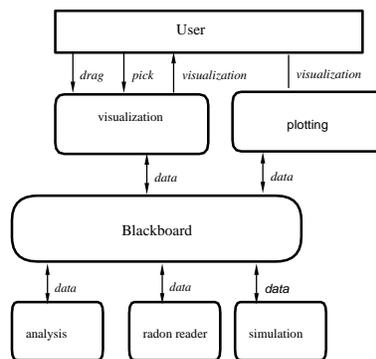


Figure 8: Blackboard configuration of Radon application.

The radon simulation satellite creates a set of three variables containing the wind fields and the simulated scalar radon field. After each time step this set will be dumped into the LBB.

A reader satellite reads the measured site data from a data base and writes this data onto the blackboard.

The analysis satellite creates the variable containing the candidate points. It continuously dumps a new candidate points into the LBB until a stop criterion is reached. In addition, the analysis satellite opens the variables created by the simulation. Whenever a new data set is produced it will read it, and if any of the candidate points have been mutated, it will read the new candidate points.

The visualization satellite will read and display the data sets and candidate points. A user may drag a point to

a new position, resulting in a new set of candidate points which are written to the LBB. The plotting satellite will read and display the time series, which are put on the blackboard by the reader and analysis satellites.

4.1 Discussion

The governing idea of this example is to show how the blackboard model is used in a non-trivial setting. The application combines qualitative user actions (direct manipulation of the visualization) with quantitative analysis tools (computations of numerical algorithms). The information is structured as; on the lowest level is the data combined from data computed by the simulation and measured data, on higher levels is the information computed by the analysis. The user has access to all levels of this information.

The distributed blackboard architecture is a natural framework for solving such problems. Whenever information at a certain level is mutated, the appropriate satellite recalculates its output using the new information and mutates the next level of information.

The radon application is very simple and should be viewed as a elementary case study. We plan to apply the distributed architecture to a 3D ozone simulation over the Netherlands. Here the fitting criteria will be a volume, the chemical reactions involved in computing ozone are much more complicated than computing radon transport, and the measured ozone data is less reliable, and there are many emission sources and sinks. Nevertheless, there are signs that this type of visualization based analysis will provide added value to the atmospheric researcher [16].

5 Conclusion

In this paper we presented a distributed blackboard architecture for scientific visualization. The blackboard architecture allowed us to address two important issues concerning interactive visualization environments. These issues are: First, tight integration between simulation and visualization. This is realized through the name concept of a variable which tightly binds data stored in in the blackboard with data in the satellite. Second, to combine qualitative and quantitative data analysis. This is realized by allowing general analysis satellites to operate in close cooperation with the visualization satellites.

The distributed blackboard architecture can be implemented very efficiently.

References

- [1] J.P. Lee and G.G. Grinstein, editors. *Database Issues for Data Visualization*. Springer Verlag, 1993.
- [2] L.A. Treinish. Workshop report: data structure and access software for scientific visualization. In *SIG-GRAPH'90 workshop report*, volume 25(2), pages 104–118. ACM, 1991.
- [3] W.C. de Leeuw and R. van Liere. Divide and conquer spot noise. In *Proceedings Supercomputing '97* (<http://scxy.tc.cornell.edu/sc97/program/TECH/DELEEUEW/INDEX.HTM>). ACM, 1997.
- [4] R. van Liere and J.J. van Wijk. Steering smog prediction. In B. Hertzberger and P. Sloot, editors, *Proceedings HPCN-Europe '97*, pages 241–252. Springer Verlag, April 1997.
- [5] R. Haber, B. Lucas, and N. Collins. A data model for scientific visualization with provisions for regular and irregular grids. In G.M. Nielson and L. Rosenblum, editors, *Proceedings Visualization '91*, pages 298–305, 1991.
- [6] W.L. Hibbard, C.R. Dyer, and B.E. Paul. A lattice model for data display. In R.D. Bergeron and A.E. Kaufman, editors, *Proceedings Visualization '94*, pages 310–317, 1994.

- [7] J. Foley and W. Ribarsky. Next-generation data visualization tools. In L. Rosenblum, R.A. Earnshaw, J. Encarnacao, H. Hagen, A. Kaufman, S. Klimenko, G. Nielson, F. Post, and D. Thalmann, editors, *Scientific Visualization : Advances and Challenges*, pages 103–126. Academic Press, 1994.
- [8] R. Englemore and T. Morgan, editors. *Blackboard Systems*. Adison-Wesley, 1988.
- [9] W.J. Schroeder, K.M. Martin, and W.E. Lorensen. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. In R. Yagel and G.M. Nielson, editors, *Proceedings Visualization '96*, pages 93–100, 1996.
- [10] J.D. Mulder and J.J. van Wijk. 3D computational steering with parametrized geometric objects. In G.M. Nielson and D. Silver, editors, *Proceedings Visualization '95*, pages 304–311, 1995.
- [11] R.K. Rew and G.P. Davis. The UNIDATA netCDF: Software for scientific data access. In *6th Interational Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydology*, pages 33–40, Anaheim, CA, 1990.
- [12] J. Welling, C. Nuuja, and P. Andrews. P3D: A lisp-based format for representing general 3d models. In *Proceedings Supercompting '90*, pages 766–774. ACM, 1990.
- [13] C. Williams, J. Rasure, and C. Hansen. The State of the Art of Visual Languages for Visualization. In A.E. Kaufman and G.M. Nielson, editors, *Proceedings Visualization '92*, pages 202–209, 1992.
- [14] V. Jaswal. CAVEvis: Distributed real-time visualization of time-varying scalar and vector fields using the CAVE virtual reality theater. In R. Yagel and H. Hagen, editors, *Proceedings Visualization '97*, pages 301–308, 1997.
- [15] J. Hultquist and E. Raible. SuperGlue: A programming environment for scientific visualization. In A.E. Kaufman and G.M. Nielson, editors, *Proceedings Visualization '92*, pages 243–250, 1992.
- [16] F. Dentener. *Personal Communication*, Institute for Marine and Atmospheric Research (IMAU), Febuary, 1998.