Scivis

Byeongseob Ki (bki@top.cis.syr.edu)
Scott Klasky (scott@npac.syr.edu)

Northeast Parallel Architectures Center
111 College Place,
Syracuse University
Syracuse, NY 13244-4100
http://kopernik.npac.syr.edu:8888/scivis/

Abstract

Visualization affects the process of analysis and design in a number of ways. Applying visualization concepts to scientific data gives scientists greater abilities to understand their data. We have designed and implemented Scivis, which is a collaborative scientific visualization system. Scivis is implemented in Java, taking full advantage of the benefits of Java. We have concentrated on three important issues: ease of use, a collaborative framework, and a highly customizable system. Since it's introduction, Scivis has generated significant interest throughout many research labs.

1. Introduction

Scivis is a client-server data visualization and analysis system which takes full advantage of Java. The purpose of this system is to provide researchers with a customizable data analysis system. Scivis is a Java application which now contains over 45K lines of code.

Our experience in designing Scivis comes from the initial work by M. Choptuik, who wrote a 1-dimension client-server visualization program in C using the native SGI language GL[16]. Scivis was initially designed from this implementation. We took Choptuik's initial ideas, and initially designed Scivis for this community of researchers (numerical relativity). Our system is currently being used in the NSF Grand Challenge, "The Binary Black Hole Grand Challenge", which is an alliance of 10 universities[17].

One of the key features of Scivis is that the filters are user definable. A filter is an action, which is performed on data set(s) to create a new data set based on this action. Since Scivis is collaborative, users can share their filters with others. This creates a powerful platform where many different users can exchange their filters with their peers, by showing other users how his/her filter performs. Since it would be impossible to design filters for all applications we have created an abstract Java class named *Filter* for users to design their own filters. The users can override this class to implement their own filters.

We also provide a collaborative framework, where users can exchange their data and their own personalized filters. Users run Scivis, a visualization server, on his/her machine. Multiple servers can be run on multiple machines, which can collaborate in a master-master or master-slave configuration. There are several known applications in Java, which have been developed to give applications porting APIs [1][4][5][6]. Since these applications were still in the development stage when we started to design Scivis, we felt it was necessary to develop our own collaborative system. To support collaboration among users, we use the *java.net* package to implement the client and server side of the connection.

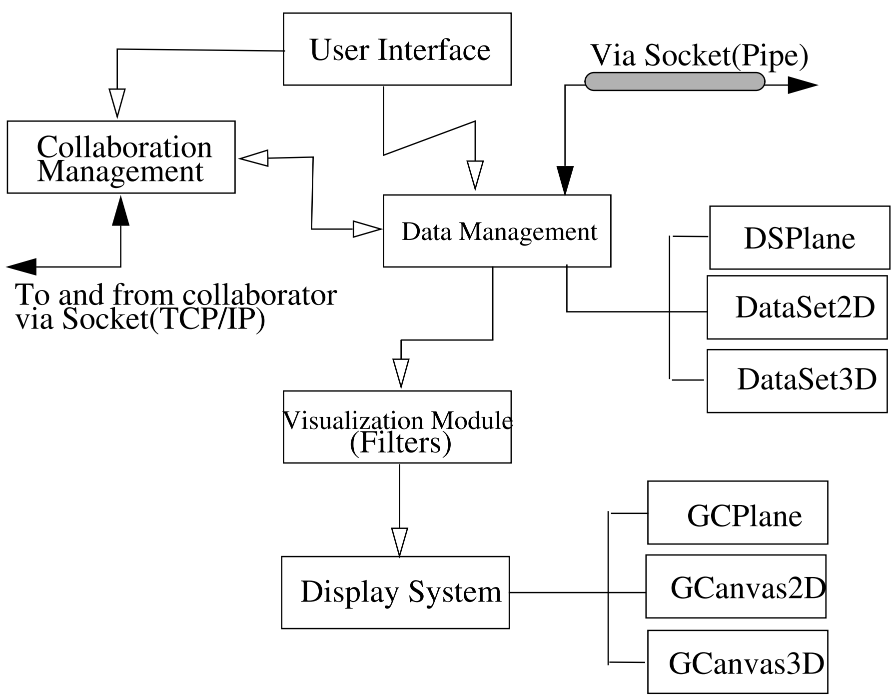
Scivis obtains each data set to be visualized from clients via sockets. Data sets are characterized by a unique name. Data sets with different names are shown on different windows. Each window contains data for a particular function which is characterized by multiple time slices. Windows can be animated for all of the time slices that the window contains; however users also have the ability to buffer only a certain amount of time slices, in order to save memory.

Even though Scivis is still being designed, many researchers and students from diverse institutes use Scivis, providing us with vital feedback which enriches our system. In this paper, we will focus on the design issues of Scivis.

2. System Architecture

The architecture of Scivis is shown Figure 1.

Figure 1 : Scivis System architecture.



The **User Interface** component defines the mechanism by which the user communicates with Scivis. This can be in the form of a command line script or graphical interface.

The **Data Management** component handles the data set(s). Data can be read from a data file or can be received from a simulation server (client with respect to Scivis). If Scivis is in collaboration mode, data can be imported from other collaborator(s). This component handles the internal data organization.

The **Visualization Module** is a set of *filters*. This component is managed by the user since all of the filters are user definable. We believe that this feature is the one of most important features of Scivis.

The **Collaboration Management Module** handles all of the collaboration features of Scivis. All of the features of Scivis are collaborative: animations, filters, whiteboard, choices of how each plot looks, etc. This feature can be used to aid researchers who which to work together to do data analysis.

The **Display System** encompasses the mechanisms which display and manipulates the geometric primitives which come from the visualization module. These primitives can be markers, text, points, lines, or polygons. This component is closely related to the User Interfaces since the user can change items such as the color of an object, its rendering style (line, flat shaded, Gouraud shaded, etc.) which effects the Display System. This component determines where and when the objects are displayed. Scivis has an ability to have multiple graphics windows to display different data types in each windows. When the orientation or view of a graphics object is modified, such as by a rotation, the Display System traverses the graphics primitives and determines what is visible and displays it with the appropriate viewing transformation. Another important operation of the Display System is the support of printing and saving with popular file format such as GIF and MPEG. With the Java Development Kit(JDK)1.1, printing can be done by using the standard class library.

Since Scivis provides a layered drawing tool, the Display System also handles geometric primitives which are drawn by the users. Selecting objects for editing can be done by picking them directly from the computer screen. Color manipulation is a very useful technique used to enhance and highlight data. Users can customize the color of the objects by dragging the color graph of Scivis. The user can also store color maps to a file and read it later.

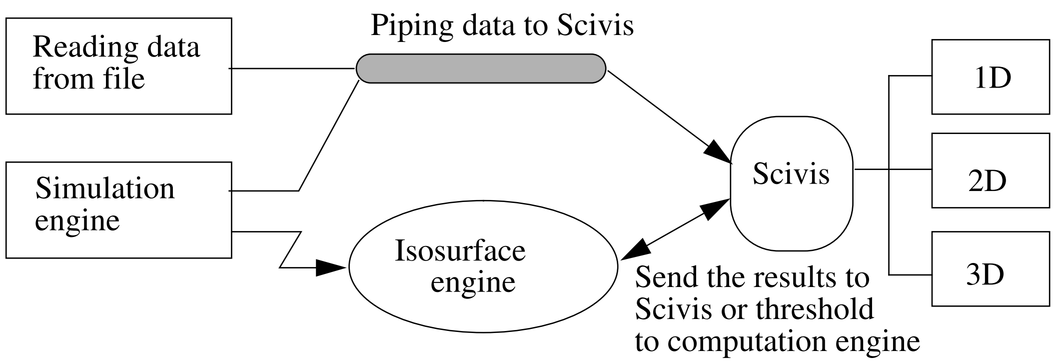
In Scivis, the Display System is implemented in 3 abstract classes called *GCPlane, GCanvas2D,* and *GCanvas3D*. Each of these classes handles 1D, 2D, and 3D displays, respectively. Various displays such as vector graphics and polygon graphics, etc. can be implemented by inheriting among these 3 classes.

3. Client -Server Model

Scivis obtains data from sockets opened up from a *simulation engine* (which we shall call the client for now). The client in this system is responsible for sending data to Scivis, which opens up a window to visualize the data. Scivis can visualize 1-3 dimensional data sets. Clients can be implemented in two ways: a program which reads data from a file and sends it to Scivis, or a program (*simulation engine*) that produce the data. Since clients can send data to Scivis via sockets, clients may be implemented in any language. We have provided API's for programmers to connect to scivis via function calls.

The *isosurface engine*, a separate engine from the visualization engine(Scivis), runs a marching cubes algorithm to take isosurfaces from the users 3D data. The 3d data from the simulation engine is piped into the isosurface engine which generates a triangle list and is then piped into Scivis. Scivis users can change the threshold values for any/all of the time slices, and the triangles are then recalculated on the isosurface engine and the new triangles are sent back to Scivis. We choose to implement the isosurface engine independently of Scivis because we wanted the user to have the flexibility in choosing where this will run, because of its computational expense as well as the expense of sending 3D data (which is far more expensive than sending around the triangles). Figure 2 shows the scheme of piping data to scivis (and to the isosurface engine).

Figure 2 : Data production and visualization by using client-server Model



All of the data from the simulation engine can be buffered in memory, so that we can animate each window over time steps. But how many time steps can be buffered in the visualization engine? It is hard to answer this question. Scivis has an ability to change the memory buffer size that holds the data. So if the buffer size is set to one, whenever the visualization engine receives new time step, it will discard the old time steps and display the new time step's data.

When Scivis receives data from a client, it checks the name of the data set, and open up a new window if the name has not been used; if it has, the data will be merged onto the window with the same name. Since Scivis opens up a separate window for each independent data set, if the name of the window is the same as the name of a data set that is already open, the datasets will be merged. We use the *BufferredInputStream* class library provided by JDK to send data to Scivis. This class implements a buffered input stream. Data is read block by block into a buffer, which allows subsequent reads to directly access the data from the buffer, which speeds up sending the data from the client to the server.

4. Filters

In order to provide a general visualization/data analysis tool we have realized that Scivis must provide a system that end users can customize, and provide new filters to analyze their data. We provide APIs so individual users can implement their own custom algorithms. Features of Scivis make it easy for the users to write their own filters to manipulate their data.

We currently support two types of filters. This is based on the desired input of data from either one data set or multiple data sets. The user can implement his/her filters inherited from the abstract class *Filter.* This class has only one abstract method called *performFilter.* This means the only thing for users to implement in their own filter is overriding the method *performFilter.* Users can add their own filters in the system by using API's which we provide, along with sample filters. Some filters need data from multiple data sets, which are distributed on multiple windows. We define such filter types as *MULTIPLE_DATASET,* which is defined in the class *Filter*. We provide support for the user to point and click their mouse to specify which data sets she/he wants to operate on in the class *Filter*.

If a user defines his/her own filters, they can become a part of Scivis by simply adding one line to a simple menu definition file (*sv_menu*) This HTML-like script is a definition of the user interface menu.

Some visualization systems provide application builders[14]. Application builders provides an interactive tool to build a directed graph. The user can select a set of modules from a palette, connect the modules together, and create complex visualization applications. Custom modules to perform tasks particular to a user's application can be written and added to this palette. The nodes of the graph are visualization modules, the edges are paths by which data can flow from one module to another. The output of one or more modules can be used as inputs to other modules. An application builder can also use a language-based interface to construct applications. The user constructs the visualization application by writing a program within an environment which supports an interpretive language. An interpretive language environment allows programs to be written, tested, and debugged interactively.

Filters of Scivis provide similar functionality to application builders but by using different method. All filters of Scivis can be used very easily by a menu system, or point and click the mouse to specify which data sets he/she wants to operate on. In addition to the ease of use, filters of Scivis can be extensible and fully customizable. Filters can be added or removed to and from the system by manipulating menu definition file. Once the filters are part of Scivis, users can perform this filter continuously by choosing from a menu.

5. Data Management

Scivis can visualize several types of data sets such as 1-3 dimensional data, vector data, isosurface data, etc. Each unique data set is composed of distinct, sequential time steps. Scivis facilitates animations and allows users to choose to loop through the time data, or step through the sequential time slices in either direction. We also allow users to go to any time slice by simply choosing its iteration number.

Figure 3 : Scivis data structure

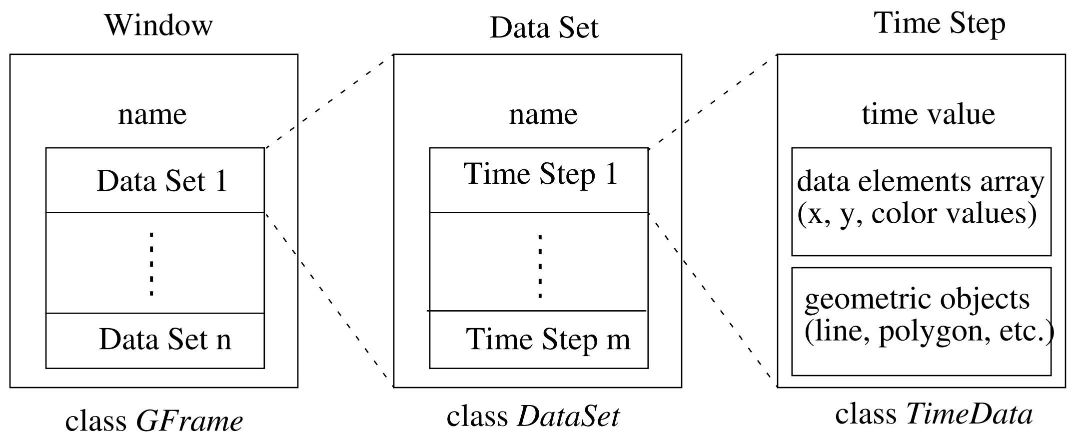


Figure 3 shows the data structure of Scivis. Each visualization window and data set has it's name to identify it from other windows and data sets, respectively. When Scivis receives a data set from a client, it creates a visualization window called *GFrame.* Each visualization window is a subclass of this abstract super class *GFrame*. The visualization window can hold several data sets. When Scivis receives data with a name that is already has, it appends this data. If the client wants to add new data to the data set, Scivis also check the data set's name and the time value of each time steps. If all are same, the data are just appended to existing data set. Otherwise Scivis creates a new data set and new visualization window which holds the data just received.

The data set is implemented by an abstract class *DataSet*. This class can be overridden by a subclass such as *DataSet2D, DSPlane,* etc. according to its rank. A class *DataSet* is implemented by using the *Vector* data structure. A data structure *Vector* is an array that can grow.

Visualization of large data sets on workstations can be very time consuming and sometimes almost impossible, even on machines with a lot of memory and processing power. The size of the data to be visualized maybe much greater than the capacity of the machine (in terms of memory) on which the visualization is being performed. Scivis has an ability to change the vector size of the data set. So the server can only hold parts of incoming data in memory. Old data which was in memory may be replaced by the incoming data or stored to a file.

The ability to write data to a file is essential. Since we are working with a client-server model, the data resides in memory, which may be stored by writing the data out to a file. When a window is closed its contents are cleared from the systems memory. Thus, when a user wants to save a particular data set, he/she will save the results in Scivis, which can be piped into the system at a later date.

Each time step is implemented by class *TimeData*. The class *TimeData* is an abstract class.

Therefore we need a subclass such as *TimeData1D, TimeData2D*, etc. according to its rank to initialize the object, time step. Each time step holds the data elements such as x and f(x) arrays(1D), x, y, and f(x,y) arrays (2D), or meshes(3D). These data elements can be converted in a form of geometric objects such as point, line, triangles, and polygons, etc. Scivis can convert the data elements to the geometric objects to display on the visualization window whenever the incoming data is received.

6. User Interfaces

The user interface is very important because it is the view of the visualization system presented to the user and it governs the user's perception of the system. Scivis provides three types of user interfaces: an interface script, graphical, and interface APIs.

A interface script provides a textual interface to Scivis and is useful when used in conjunction with a graphical interface, where there is one-to-one relationship between the script and the graphical interface. This interface script is stored in as a configuration file, creating a user defined environment at the start-up of the visualization system and records a visualization session for playback later.

A graphical interface provides a visual interface to Scivis with a mouse and keyboard. These interface components are implemented by using Java Abstract Window Toolkit (AWT). The AWT provides many standard GUI components such as buttons, lists, menus, and text areas. It also includes containers (such as windows and menu bars) and higher-level components (such as a dialog for opening or saving files)[10]. With JDK 1.1, the web-centric AWT GUI toolkit becomes part of a comprehensive set of GUI classes and services called The Java Foundation Classes(JFC). The JFC components (nicknamed "Swing components") facilitates a customizable look and feel without relying on the native windowing system, and simplifies the deployment of applications. At the point of writing this report, the JFC is still in a beta stage (version 0.51); but it helps us to reduce an efforts to write fancy GUI components.

An interface API provides the user to customize the look remotely. It is useful when the user generates the output within the program or script. When the user wants visualized outputs automatically, he/she can sends variables to the server to change the interface. We support APIs to change labels, titles, and tick marks, etc.

7. Collaboration

There are several possible methods to implement collaboration features: using socket programing, using Remote Method Invocation(RMI), using Common Object Request Broker Architecture(CORBA) and Interface Definition Language(IDL), and porting to existing collaborative systems.

7.1. Collaboration Architecture of Scivis

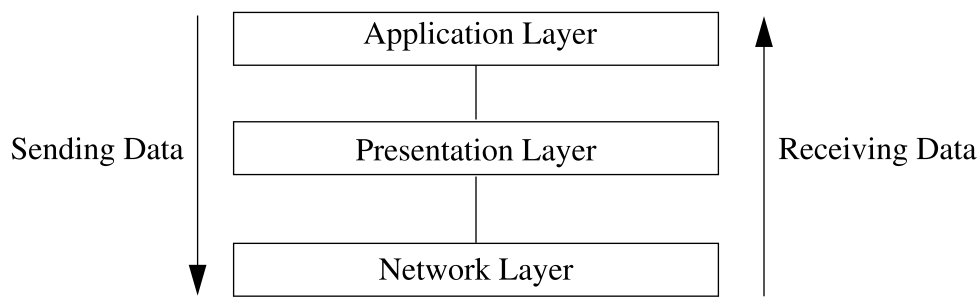
To support collaboration among users, we have made use of the Java Socket and ServerSocket classes. In the beginning stage of our software, we intended to develop collaboration work using RMI (Remote Method Invocation) since Java 1.1 supports RMI. This feature provides simplicity and inherent portability, and, coupled with the Delegation Event Model, it lets us create an extensible collaboration framework that can be used even with proprietary components. RMI also provides a simple solution to distribute the processing of intensive tasks from the client to the server among different servers[12]. But we found that RMI is much slower than using socket as shown in Table 1. The test results of Table 1 are based on Ping performance[15]. Since execution speed is a major factor in designing our system, we choose to develop our own collaborative system by using socket programming.

Table 1 Performance Comparison on Ping performance(msecs)

	CORBA/IIOP	RMI	HTTP/CGI	Socket
Performance	3.3	5.5	603.8	2.0

Communicating messages and data among collaborators is done by 3 steps. Each steps can be implemented independently of the other ones. We can define these steps as layers similar to OSI model as shown in Figure 4.

Figure 4 Collaboration Architecture, which is consisted of three layers.



The server socket of Scivis listens to incoming messages and data sets from the client and the collaborator. When a message or the data set arrives, the network layer of Scivis makes a thread to receive incoming message or data set. Since the size of the incoming data may be large, it is desirable to use the multithreading feature of Java. To give the Scivis users better interactive responsiveness and real-time behavior, Scivis sends and receives the message and data set by background thread.

We designed Scivis to make it easy to port to other collaboration system. To achieve this goal, we have divided up collaboration tasks into three layers. Each layers deals with one specific aspect of the communication. Each of which can be implemented independently of the others.

- **Application Layer**
  The application layer consists of two methods:
  public void send(Object obj, int obj_type)
  public void doWork(Object obj, int obj_type)
  Visualization window, Data set, or Abstract Window Toolkit(AWT) events can be sent or received by using

these two methods. JDK1.1 uses delegation event model to handle AWT component event handling. JDK1.1 defines 9 interfaces for different types of listeners. Each listener interface defines methods that will be invoked when a specific event occurs. Collaboration is also handled by event listener when the event occurs. When an event occurs, then the event listener broadcasts events by using the *send* method, and other collaborators receive the events and perform an appropriate event by using the *doWork* method. And when the data set is received from the client, this data set is also broadcasted to the collaborators using the *send* methods as like events.

- **Presentation Layer**
  The objects received from the application layer is converted to a byte stream to send to collaborators using sockets. The byte stream received from the network layer should also be converted to an appropriate object to send to the application layer. These conversion can be done in the presentation layer. The presentation layer also adds a data header to the objects received from application layer and sent it to the network layer or makes an object from the byte stream received from the network layer and send it to the application layer.
- **Network Layer**
  The Network layer delivers byte stream converted from the presentation layer using sockets. Sending and receiving the message or data to/from the other collaborator is implemented by using the multithreading feature of Java. Multithreading gives the user better interactive responsiveness and real-time behavior.
  When collaboration is established, Scivis makes a collaborator table to identify the collaborators.
  The network layer manages collaborator table whenever other collaborator joins or leaves; it also looks up collaborator table to find the destination to deliver the message.

Filters are also collaborative among collaborators. Users can share their filters with collaborators who do not have them. When the user performs a filter, the classes which are the implementation of that filter are sent to the collaborators with the form of a byte stream. Then Scivis of the collaborator reconstructs the filter from the byte stream just received. Since most filters can be converted into less than 3 kilobytes stream of its size, it is reasonable to communicate the filter rather than the data set.

The connection between two collaborators can be either master/master or master/slave mode. The actions of the master are reflected on all other collaborator's windows. The slave users are limited to simply observing the actions of the master. It may be the case that a user wants hide some data from the other collaborators. Scivis supports a *stealth mode* collaboration to hide some data from the collaboration.

8. Conclusion

We have implemented a collaborative data analysis and visualization application, named Scivis. Scivis is flexible and very usable and thus we feel that researchers can benefit from using our software. Many researchers and students in diverse institutes are already taking advantage of Scivis. Scivis has the functionality to fulfill the researchers' needs and can execute efficiently on large data set(s). User can also extend the system by writing their own filters. In addition, Scivis is designed to be easy to use, to be integrated to other collaboration systems, and to be used as a visualization system of a parallel or distributed systems. We have provided APIs to make it easy for users to incorporate Scivis into their research.

Scivis has a sluggish performance when the software renders 3D colormapped surface plots. Scivis can be drastically improved if we had hardware acceleration for graphics. We are going to use the Java3D API when it is available in the future releases. As a part of Java Media, the Java 3D API is an application programming interface used for writing stand-alone 3D graphics applications or Web-based applets. It gives the developer high level constructs for creating and manipulating geometry and for constructing the structures used in rendering that geometry. Application developers can describe very large virtual worlds using these constructs, which provides Java 3D with enough information to render these worlds efficiently[9].

9. References

[1] K. Mani Chandy, *Caltech Infospheres Project Overview: Information Infrastructures for Task Forces,* via `http://www.infoshperes.caltech.edu/`.

[2] K. Mani Chandy, E. Schooler, *Designing directories in distributed systems: A systematic framework,* IEEE International Symposium on High Performance Distributed Computing (HPDC-5), Syracuse, NY, Aug 1996.

[3] K. Mani Chandy, Adam Rifkin, Paolo A.G. Sivlotti, *A World-Wide Distributed System Using Java and the Internet,* IEEE International Symposium on High Performance Distributed Computing (HPDC-5), Syracuse, NY, Aug 1996.

[4] L. Beca, et. al. *TANGO - a Collaborative Environment for the World-Wide Web*, via `http://www.npac.syr.edu/tango/papers.html`.

[5] NCSA, *Habanero Developer Guide*, via http://ncsa.uiuc.edu/.

[6] Bjorn J.Kvande, *The Java Collaborator Toolset*, Master's Project Report, Department of Computer Science, Old Dominion University, Aug. 1996. or via `http://www.cs.odu.edu/~kvande/Projects/Collaborator`.

[7] K. Arnold, J. Gosling, *The Java Programming Language,* Addison-Wesley, 1st edition, 1996.

[8] J. Gosling, B. Joy, and G. Steele, *The Java Language Specification*, Addison-Wesley Developers Press, Sunsoft Java Series, 1996.

[9] Sun Microsystems, *The Java 3D API White Paper*, via `http://www.javasoft.com/marketing/collateral/3d_api.html`.

[10] Sun Microsystems, *The Java Language White Paper*, via `http://java.sun.com/docs/overviews/java/java-overview-1.html`.

[11] Sun Microsystems, *Object Serialization Specification,* via `http://www.javasoft.com:80/products/jdk/1.1/docs/guide/serialization/index.html`.

[12] Sun Microsystems, *RMI Specification,* via `http://www.javasoft.com:80/products/jdk/1.1/docs/guide/rmi/index.html`.

[13] H. Abdel-Wahab, et. al. *Using Java for Multimedia Collaborative Applications*, Proceedings of the 3rd International Workshop on Protocols for Multimedia Systems (PROMS'96), October 1996, Madrid.

[14] Dyer, D.S., *A Data Flow Tokyoite for Visualization*, IEEE Computer Graphics and Applications, Vol. 10, No 4, July 1990, pp. 60-69

[15] Orfali, R, C*lient/Server programming with Java and CORBA,* 1997 John Wiley& Sons Inc.

[16] Matthew Choptuik, *Ser,* via `ftp://helmholtz.ph.utexas.edu/pub/rvs/rvs/src/`.

[17] The Binary Black Hole Grand Challenge, via `http://www.npac.syr.edu/projects/bh/`.