

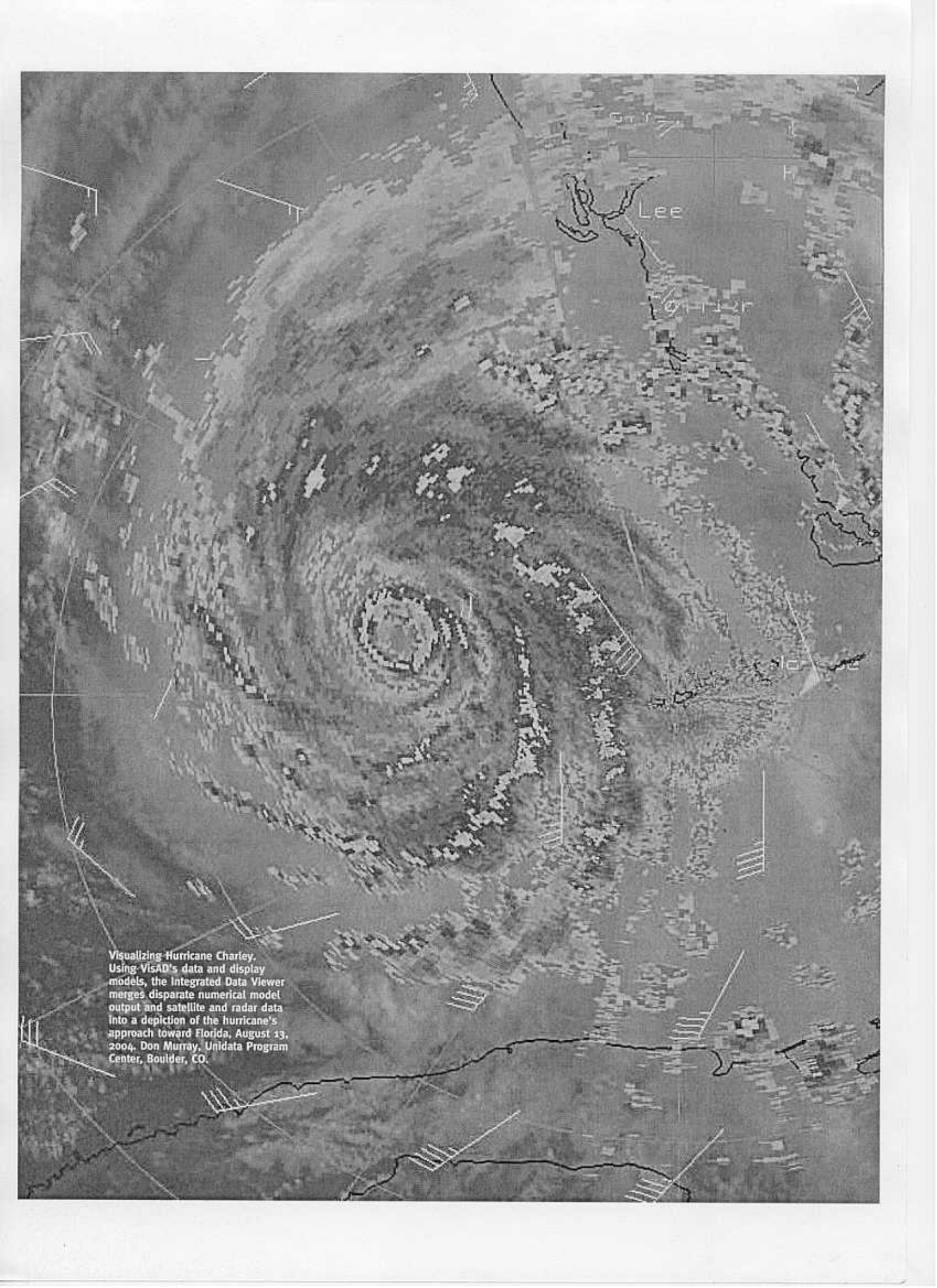
By WILLIAM HIBBARD,
CURTIS RUEDEN, STEVE EMMERSON,
TOM RINK, DAVID GLOWACKI,
TOM WHITTAKER, DON MURRAY,
DAVID FULKER, AND
JOHN ANDERSON

JAVA DISTRIBUTED COMPONENTS FOR NUMERICAL VISUALIZATION IN VISAD

*Combining a flexible
data model and
distributed objects,
they support the
sharing of data,
visualizations, and
user interfaces
among multiple
data sources,
computers, and
scientific disciplines.*

The Web is the Internet's killer application because it enables the widespread sharing of information. Within 10 years, the numerical computing environment will enhance sharing among scientists by adding a new structure to the Web. It will consist of a persistent, active network of numerical data and computational, display, and user-interface components distributed across the Internet. As with the Web, users and even application programs will be largely unaware of physical computers but will instead have access to a worldwide shared network of logical components. Users will explore the network through

browsers and add new components to it. For example, atmospheric chemists might use a browser to locate a weather simulation data set or even a running weather model, connect it as input to their chemistry models, then connect a display component to visualize each model's computations. Weather-modeling colleagues might then clone that display in their own browsers and connect user-interface components to collaborate on experiments with the coupled models.



Visualizing Hurricane Charley. Using VisAD's data and display models, the Integrated Data Viewer merges disparate numerical model output and satellite and radar data into a depiction of the hurricane's approach toward Florida, August 13, 2004. Don Murray, Unidata Program Center, Boulder, CO.

Such an environment will challenge several long-held assumptions about the way programmers write numerical software, including:

- *Single machines or fixed partitions across machines.* Numerical software runs (and data resides) on a single machine or is partitioned across multiple machines in a fixed configuration; the challenge comes from grid computing [1, 2] and distributed object technology [7];
- *Structure and properties of data.* The structure and properties of data are known ahead of time by programmers who write software for accessing the data; the challenge comes from abstract numerical data models [3, 4, 6]; and
- *Uses of software.* These uses are known ahead of time by the programmers writing the software; the challenge comes from reusable components [5].

We are developing the Visualization for Algorithm Development, or VisAD, library of Java components to overcome the limitations these assumptions impose on numerical visualization, defining four types of Java components:

Data. Implements an abstract data model that defines a schema grammar for organizing numerical and text values. It also defines associated metadata (such as units, coordinate systems, sampling geometries and topologies, missing data indicators, and error estimates). The schema grammar and metadata can express images, 3D grids, time series, map boundaries, simple real numbers, and practically any other numerical data. The data-component API hides interfaces to a variety of file and server data formats, movement of data between

disk and memory, movement of data across the network, and partitions of large data components across processor clusters.

Display. Implements an abstract display model that enables applications to define data depictions descriptively—via mappings from primitive data values to primitive display values—rather than procedurally. The display-component API (such as Java3D and Java2D) hides the graphics library used to render depictions, as well as whether depictions are rendered in windows on workstation screens, in browsers, or in immersive virtual reality displays.

Computation. Uses code supplied by applications to compute values for data components or manipulate display components based on the values of other data components. The computational-component API hides the programming language used for application-supplied code.

User interface. Includes the familiar screen icons (such as buttons and sliders) linking user actions (such as mouse clicks) to values of data components and to library calls.

Networks of such components span multiple computers via Java Remote Method Invocation (RMI)

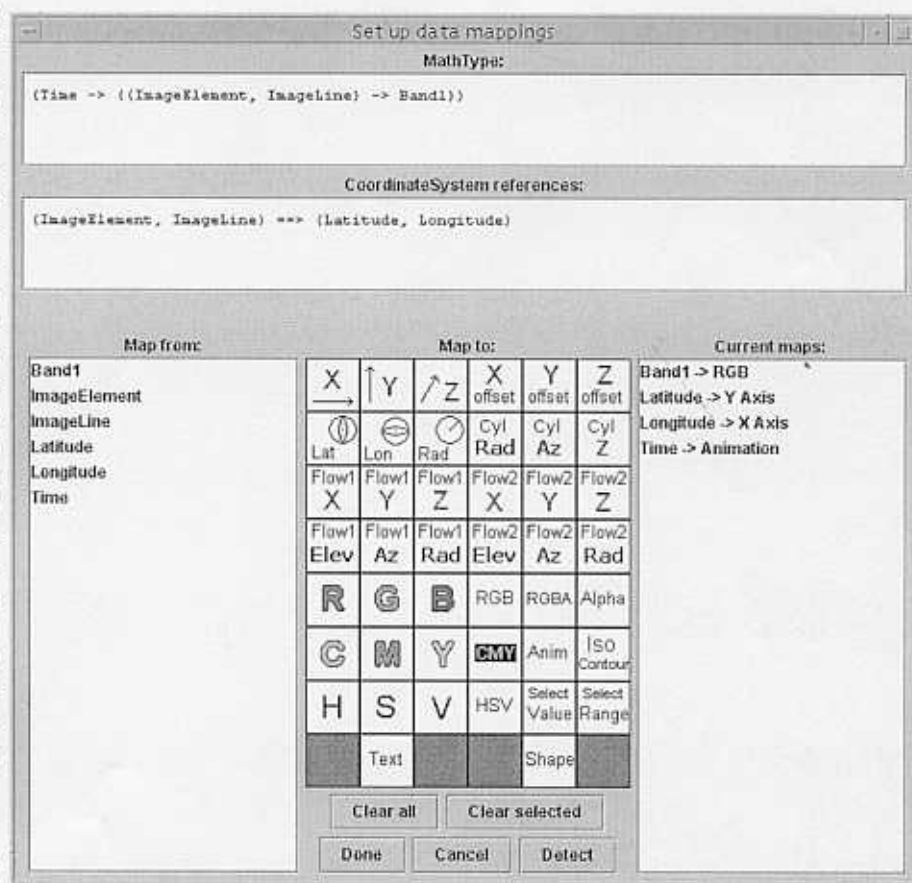


Figure 1. Display mappings dialogue panel.

distributed object technology. They can express various forms of distributed computing (such as client-server, cluster processing, and remote collaboration, as well as whatever programmers and users care to define). Display and computational components may be linked to data components, with their actions (updating data depictions or executing application-supplied code) triggered whenever linked data values change. A data component may be linked to multiple display components with a different depiction in each; multiple data components may be linked to a common display component for visual comparison. Display components may be linked to yet other display components, creating collaborative networks of displays that synchronize their appearance; any change by users or applications is reflected in all linked display components. User-interface components may be linked to data components, enabling users to manipulate data values. Display components can also be used as user-interface components, enabling users to manipulate data values by redrawing their depictions. All these connections may be either local or remote.

Abstract Data and Display Models

Abstraction is the key to reusability. VisAD achieves abstraction for its data components through a

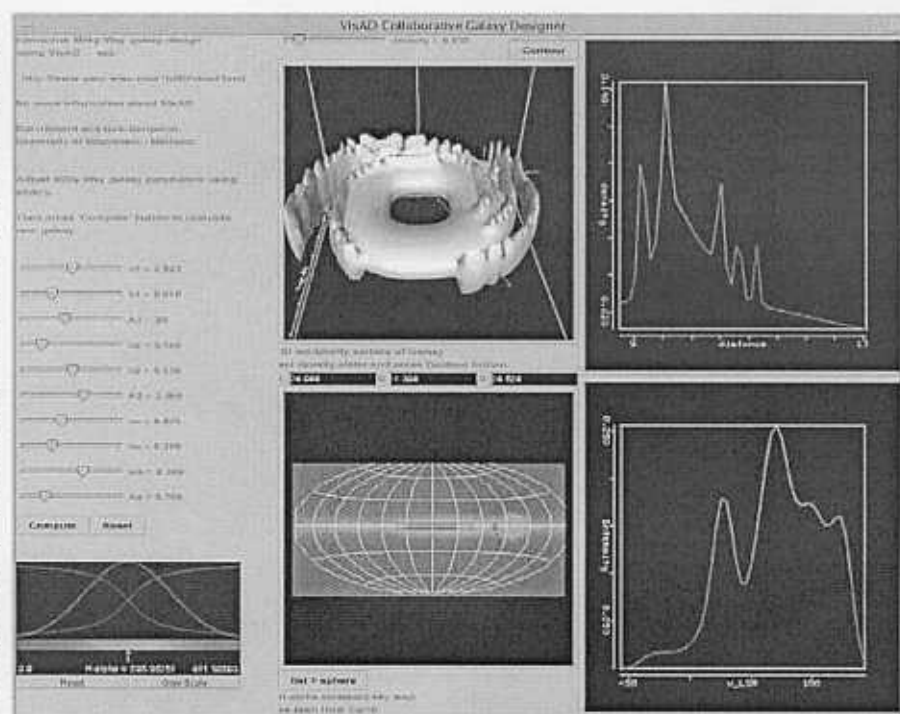


Figure 2. The collaborative Galaxy application, simulating the Milky Way galaxy as it would appear from Earth.

schema grammar for expressing data organizations, and for its display components through expression of data depictions as mappings from primitive data values to primitive display values. Figure 1 is a VisAD user-interface component that enables users to define display mappings. The top *MathType* window includes an expression in the schema grammar for a time sequence of 2D Earth images—a data component to be displayed. This expression includes names for primitive numerical and text values, groupings of values into vectors, and functional dependencies among values. Below that, the *Coordinate System references* window shows that the data component includes an invertible transform between image coordinates and Earth coordinates. All the numerical values occurring in data components may include associated units and error estimates. There are also usually samplings for the

DISPLAY COMPONENTS CAN BE USED AS
USER-INTERFACE COMPONENTS, ENABLING
USERS TO MANIPULATE DATA VALUES BY
REDRAWING THEIR DEPICTIONS.

Figure 3. VisBio volume rendering of a live *C. elegans* embryo (imaging performed by William Mohler, University of Connecticut Health Center, Farmington, CT).

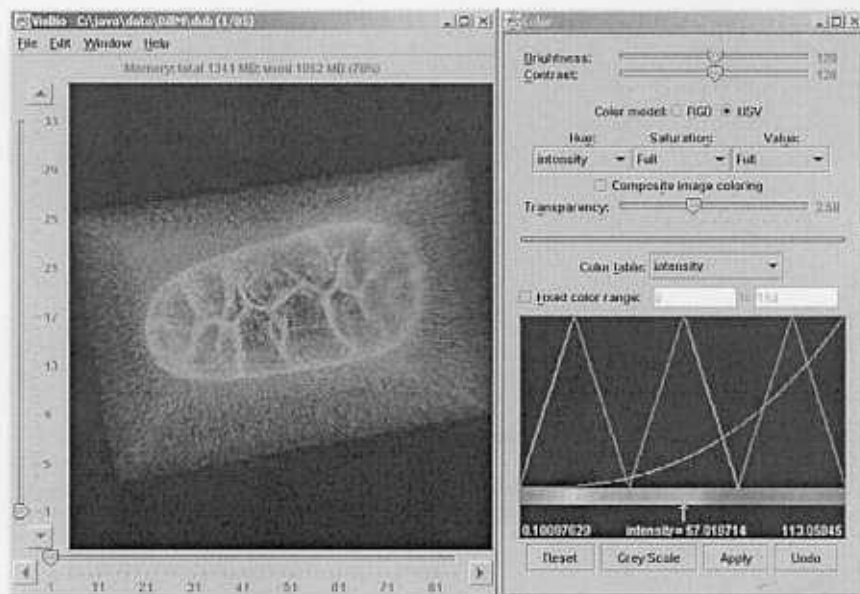
domains of any functional dependencies. Unit conversions, coordinate transforms, resampling, and propagation of missing data and error estimates are all done implicitly as necessary in mathematical and display operations on data components.

In order to define data depictions, primitive data values—in the *Map from* window—are mapped to primitive display values—in the *Map to* window. The *Current maps* window shows the system's first guess at appropriate mappings for the schema in the *Math Type* window. The user can clear these mappings and create new ones by alternately clicking on primitive data value names and display value names. The user interface component in Figure 1 defines mappings via library calls available to any application.

The data schema grammar enables data components to be reused for virtually any numerical and text data. Moreover, the associated metadata enables meaningful comparisons of data from diverse sources, including the spatial and temporal alignment in displays, and is important for increased data sharing on the Internet. The system includes a set of classes for interpreting data file and server formats as data components, implicitly transferring data to a memory cache as needed to execute data-component API calls. VisAD developers have applied these classes to more than 20 common numerical data formats.

The display mappings enable the reuse of display components for virtually any form of data depiction. The fact the display mappings are a descriptive rather than a procedural definition of data depictions enables display components to also be used as user-interface components, with users modifying data values by redrawing data depictions. That is, procedures are difficult to invert, whereas descriptions apply just as well in both the data-to-depiction and the depiction-to-data directions.

Developers can extend the Java classes implementing data components in order to define their own coordinate systems, sampling topologies, and interpo-



lation algorithms. With Java platform independence, these mathematical algorithms can be transferred with data components among various machines; in this way, algorithms can function as data content. Developers can extend classes implementing display components in order to define their own rendering algorithms or even to use a different graphics library.

Visualization and Analysis

For new users, the VisAD library is a challenge due to its high level of abstraction. Their first step is usually to visualize their data in the VisAD Spreadsheet, which provides access to much of the library via a GUI. The user-interface component in Figure 1 is part of that GUI, helping users learn about the data schema grammar and the display mappings. They can experiment with data files and sets of display mappings, then see the resulting visualizations. The Spreadsheet GUI consists mainly of a rectangular array of cells (display-component windows), each possibly containing depictions of multiple data components. These components are generated by reading from files or servers or by simple formulas applied to data components in other cells.

When users are ready to program the library, the easiest way to start is by writing Python scripts. A script can be simple, as in the single line `plot(load("filename"))`, which loads and displays a data file. VisAD supports Python via the Jython implementation, providing access to Java objects from Python, and means the entire VisAD library is accessible from Python. Support is also available for mathematical operations on data components

THE VISAD LIBRARY IS BEING USED TO WRITE TRADITIONAL VISUALIZATION APPLICATIONS THAT ASSUME SPECIFIC DATA STRUCTURES AND DEPICTIONS. THESE APPLICATIONS TYPICALLY REQUIRE A FEW HUNDRED TO A FEW THOUSAND LINES OF JAVA.

via Python infix expressions and for specialized displays (such as histograms, scatter plots, contour plots, and image animations) without requiring users to understand the system's display mappings. Python scripts can invoke the plot function to depict data components using a SpreadSheet cell, allowing users to control display mappings via the user-interface component, as in Figure 1.

Collaborative Simulation

The VisAD library is being used to write traditional visualization applications that assume specific data structures and depictions. These applications typically require a few hundred to a few thousand lines of Java. The library distribution includes approximately 12 such applications as examples for new users.

The Galaxy application (its GUI is in Figure 2) is fairly typical, enabling teams of astronomers to collaborate on experiments with physicist Robert Benjamin's simulation of the Milky Way galaxy and see how its H-alpha emission sky map and spectra would look from Earth. Simulation parameters are defined by a set of simple real-number data components users adjust via the slider components shown on the left side of Figure 2. The simulation code—written in Fortran encapsulated in a computational component—produces data components linked to display components generating other windows in Figure 2. The upper-center window in the figure shows an iso-surface of simulated warm gas density; the lower-center window shows the H-alpha sky map as seen from Earth. The red point and green line in the upper-center window depict a vector from Earth to some point outside the Milky Way galaxy. Users drag the red point to manipulate the vector; changes to the vector trigger a second computational component to produce density and spectra along the vector, as in the upper- and lower-right windows.

A user who begins running the first copy of the Galaxy application has all data, computational, display, and user-interface components. However, other users who begin running collaborative copies of the Galaxy application generate only new display and user-interface components linked via RMI to the data and computational components in the first copy of the application.

Exploiting Reusable Components

The VisAD library is being used to support applications (more sophisticated than the Galaxy application) that do not assume specific data structures and depictions. They are continuing projects requiring years of development. The SpreadSheet—the first such application—deals with any data structure, accesses data from remote servers, and is fully collaborative; the user-interface component in Figure 1 enables users to generate any depiction. Multiple users might link their SpreadSheets together, and actions initiated by any individual user are seen identically in the GUIs of all.

The Unidata Program Center, part of the University Corporation for Atmospheric Research in Boulder, CO, is using VisAD to develop the Integrated Data Viewer (IDV) as part of a National Science Foundation-supported mission to supply Earth science data and access software to U.S. universities. The IDV enables users to browse remote servers and combine their data in a common spatial-temporal frame of reference. Due to the diversity of environmental-observing instruments and simulations, Earth science data involves a variety of structures and properties supported by the IDV. In addition to being able to access standard Earth data servers, the IDV provides a Web-browsing user-interface component that recognizes links to numerical data files. Clicking on the links downloads the files into the spatial-temporal visualization window rather than into the browser window.

The Laboratory for Optical and Computational Instrumentation at the University of Wisconsin-Madison uses VisAD to develop the VisBio system for visualizing and analyzing large multidimensional microscopy data sets. Figure 3 shows a VisBio volume rendering of a 3D microscopy image of a live embryo of *C. elegans* (a species of nematode worm). In addition to various forms of image displays, the system defines custom cursors users drag to measure distances in images and movements in time sequences. A number of data schemas are appropriate for a variety of microscopy data sets, depending on whether they include depth (3D vs. 2D), time sequencing, multiple spectra, and multiple optical lifetimes. With up to six independent variables, microscopy data sets can be quite large. Thus VisBio employs progressive refinement rendering (low resolution while the scene is changing, high resolution when change stops) and complex memory management.

The Australian Bureau of Meteorology is using VisAD to develop the Australian Integrated Forecast System (AIFS) 2 system, consisting of a number of modules supporting forecaster tasks. Most of these tasks require overlays of data with diverse structures and properties. They also require user manipulation of data by dragging their depictions. Meanwhile, the U.S. National Center for Atmospheric Research in Boulder, CO, is using VisAD to develop its Visual Meteorology Tool (VMET) system for visual meteorology. Like the IDV and AIFS, VMET must display data with diverse structures and properties and produce a variety of data depictions.

VisAD's reusable components are also being used for experiments with visualization techniques. VisAD developers have extended classes implementing data components to support large data components partitioned across processor clusters. They've extended classes implementing display components to exploit parallel processing for visualizing these partitioned data components. And they've also extended classes implementing display components to depict data in the ImmersaDesk virtual reality system, as well as for progressive refinement rendering. The VisAD library has proven itself a useful tool for visualization research because it enables experiments at any level via class extensions. It also provides the necessary infrastructure programmers need to write practical applications that generate evaluations of new techniques by real users.

The VisAD library, including source code, documentation, and application examples, is freely available from www.ssec.wisc.edu/~billh/visad.html. Ugo Taddei of the Institute of Geography at the University of Jena in Germany has contributed a fine online general tutorial to go along with the specialized tuto-

rials on the site. Approximately 15 programmers from a half dozen institutions have contributed code to the library. The much larger community of programmers who use the library are supported by an active mailing list used for online discussion and collaboration. ■

REFERENCES

1. Colwell, R. From terabytes to insights. *Commun. ACM* 46, 7 (July 2003), 25-27.
2. Getov, V., von Laszewski, G., Philippsen, M., and Foster, I. Multiparadigm communications in Java for grid computing. *Commun. ACM* 44, 10 (Oct. 2001), 118-125.
3. Haber, R., Lucas, B., and Collins, N. A data model for scientific visualization with provisions for regular and irregular grids. In *Proceedings of the IEEE Visualization conference* (San Diego, Oct.). IEEE Computer Society Press, Los Alamitos, CA, 1991, 298-305.
4. Hibbard, W., Dyer, D., and Paul, B. Display of scientific data structures for algorithm visualization. In *Proceedings of the IEEE Visualization conference*. IEEE Computer Society Press, Los Alamitos, CA, 1992, 139-146.
5. Meyer, B. On to components. *IEEE Comput.* 32, 1 (Jan. 1999), 139-140.
6. Treinish, L. SIGGRAPH 1990 workshop report: Data structure and access software for scientific visualization. *Comput. Graph.* 25, 2 (May 1991), 104-118.
7. Wollrath, A., Riggs, R., and Waldo, J. A distributed object model for Java. In *Proceedings of the 2nd Conference on Object-Oriented Technologies and Systems (COOTS)* (Toronto, June). USENIX, Berkeley, CA, 1996, 219-231.

WILLIAM HIBBARD (billh@ssec.wisc.edu) is an emeritus senior scientist and principal investigator in the Space Science and Engineering Center at the University of Wisconsin-Madison.

CURTIS RUEDEN (ctrueden@wisc.edu) is a research intern in the Laboratory for Optical and Computational Instrumentation at the University of Wisconsin-Madison.

STEVE EMMERSON (steve@unidata.ucar.edu) is a software engineer in the Unidata Program Center at the University Corporation for Atmospheric Research, Boulder, CO.

TOM RINK (rink@ssec.wisc.edu) is an instrumentation technologist in the Space Science and Engineering Center at the University of Wisconsin-Madison.

DAVID GLOWACKI (dglo@ssec.wisc.edu) is an assistant instrumentation innovator in the Space Science and Engineering Center at the University of Wisconsin-Madison.

TOM WHITTAKER (tomw@ssec.wisc.edu) is a researcher in the Space Science and Engineering Center at the University of Wisconsin-Madison.

DON MURRAY (dmurray@unidata.ucar.edu) is a software engineer in the Unidata Program Center at the University Corporation for Atmospheric Research, Boulder, CO.

DAVID FULKER (fulker@ucar) is a principal investigator in the National Science Digital Library Headquarters of the University Corporation for Atmospheric Research, Boulder, CO.

JOHN ANDERSON (janders@pixar.com) is a senior scientist in studio tools, research at Pixar Animation Studios, Emeryville, CA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.