6.8 A JAVA AND WORLD WIDE WEB IMPLEMENTATION OF VisAD

William Hibbard *, John Anderson and Brian Paul
Space Science and Engineering Center
University of Wisconsin - Madison

1. INTRODUCTION

The VisAD system (Visualization for Algorithm Development) was designed to enable scientists to interactively and visually experiment with their computations [Hibbard, Dyer and Paul, 1992; VisAD Web Page]. It is distinguished by its abstract view of data and visualization that can be adapted to virtually any scientific computations [Hibbard, Paul, Santek, Dyer, Battaiola, and Voidrot-Martinez, 1994; Hibbard and Paul, 1995]. However, the system runs on only one vendor's workstations (Silicon Graphics) and provides its own unique programming language for scientists to use to express their computations (although programs in this language can call functions written in Fortran and C). Furthermore, like many scientific analysis and visualization systems, VisAD has a generic user interface that many scientists find difficult to adapt to their specific disciplines.

2. JAVA

We are addressing these problems by rewriting the VisAD system as a class library in the object-oriented Java programming language [Java Web Page; Flanagan, 1996] (the previous implementation used the C programming language). Implementation as a Java class library will:

1. Make the system portable across virtually all hardware platforms and operating systems.
2. Enable scientists to express their computations in the Java programming language.
3. Support a variety of discipline-specific user interfaces.
4. Exploit the capability of Java and the World Wide Web for giving users access to network information without needing to know details about moving data between physical computers and between different software systems.

As we argue in another paper in these proceedings, Java and the Web are the right choice for implementing interactive scientific software systems [Hibbard, Anderson, and Paul, 1997].

* Corresponding author address: William Hibbard, SSEC, 1225 W. Dayton St., Madison, WI 53706; email address: whibbard@macc.wisc.edu

3. DATA MODEL

Traditional data base management systems are based on data models involving discrete entities (e.g., people, departments, addresses, salaries, etc.) and relations between them. Scientific data are different because they deal with continuous phenomena. Scientists model nature using infinite precision real numbers and functions with infinite domains. However, these mathematical objects are represented inside computers by data objects which are finite and therefore approximate.

The VisAD data model is based the idea that data objects are approximations to mathematical objects [Hibbard, Dyer and Paul, 1994; Hibbard, 1995]. It is partitioned into a collection of data types. The *primitive data types* of the VisAD data model include approximations to real numbers, integers, and text strings. The *complex data types* of the VisAD data model include finite *tuples* of other types, finite representations of *sets* over domains defined by tuples of real types, and finite samplings of functions from domain types to range types. These finite samplings are called *fields* [Haber, Lucas and Collins, 1991]. A field's domain type must be a real type or a tuple of real types.

The VisAD data model integrates *metadata* that document how data objects approximate mathematical objects. These include:

1. *Missing* data indicators, interpreted as the least precise approximation to mathematical objects.
2. *Numerical precision* information for approximations to real numbers. Examples are single and double precision floating point, and rational intervals (i.e., lower and upper bounds are defined as ratios of infinite-precision integers).
3. *Sampling information* for fields and sets. For example, a satellite image of the earth is a field that finitely samples a continuous radiance function whose domain is a set of 2-dimensional Earth coordinates. Sampling information for such a field is essentially the satellite navigation metadata that assigns Earth locations to image pixels.

The VisAD data model includes other forms of metadata, such as data source attribution, user-defined names and units for primitive data types, and indicators for primitive types used as spatial and temporal coordinates. These indicators enable the system to translate between different spatial and temporal coordinate systems.

The data model supports arbitrary hierarchies of tuples, sets and fields, allowing very complex scientific data structures to be expressed. For example, a multi-band image

can be expressed as a 2-D field of tuples, where the field domain is a set of pixel locations in an appropriate 2-D coordinate system (e.g., planet latitude and longitude for planetary images, or celestial coordinates for sky images) and the field range type is a tuple of the multiple band values for each pixel. A time sequence of such images can be expressed as a 1-D field of images, where the field domain is a set of time samples and the field range type is the 2-D image field type. A set of discrete observations can be expressed as a 1-D field of tuples, where the field domain is a set of integers from 1 to N (where there are N observations) and the field range type is a tuple of observed quantities, such as time, latitude, longitude, altitude, temperature, pressure, etc. Map boundary data may be represented as a set in the domain of latitude / longitude tuples, represented as a finite set of line segments.

Since data objects are interpreted as approximations to mathematical objects, computational and visualization operations on data objects are interpreted as approximations to corresponding operations on mathematical objects. This interpretation integrates metadata into data semantics. For example, arithmetical operations with missing operands produce missing values (e.g., *missing* + 1.0 = *missing*), whereas complex statistical operations generally ignore missing data (e.g., the mean of {1.0, 2.0, *missing*} = 1.5). Arithmetical operations on rational intervals are defined by interval arithmetic [Moore, 1966]. Arithmetical operations on fields are similar to array operations supported by interpreted vector languages such as IDL, except that field operations include resampling and interpolation from the sample locations of one field to the sample locations of another field. For example, two images (i.e., 2-D fields) of the Earth are geographically aligned before they are added or subtracted. Similarly, Earth images are geographically aligned before they are displayed.

Metadata about names and units of primitive values are also integrated into computational and visualization operations. They are used to define coordinate systems for displays, and unit conversions may be implicit in arithmetical operations.

4. DISPLAY MODEL

Each VisAD data object has a data type which is defined in terms of a set of primitive data types with user-defined application-specific names (e.g., temperature, pressure, latitude, longitude, time, etc.). Similarly, VisAD displays have complex types defined in terms of a set of *primitive display types* (e.g., 2-D or 3-D display coordinates, color components, transparency, index into an animation sequence, etc.). A VisAD display is defined by a set of mappings from primitive data types to primitive display types, called a *display frame of reference*. For example, a time sequence of satellite images is usually displayed in a frame of reference that maps Earth coordinates to 2-D display coordinates, maps satellite radiance to color, and maps time to an animation index.

The mappings from primitive data types to primitive display types include continuous functions that transform tuples of values of data primitives into tuples of values of display primitives. This may be used, for example, to create logarithmic display coordinates or to display 2-D planetary images on the surface of a 3-D sphere or ellipsoid.

Whereas the C implementation of VisAD defined a fixed set of primitive display types, the Java implementation will enable users to define new primitive display types and new rendering algorithms appropriate for those types. For example, a primitive display type could be defined for the set of chemical elements, along with an algorithm that renders elements with spatial locations as balls labeled with chemical symbols. Since complex data types and display frames of reference defined by users are unpredictable, rendering algorithms may decline to render certain data objects (e.g., an element renderer may decline to render a data structure that does not specify spatial locations for atoms).

Organizing VisAD display rendering as a set of algorithms, each oriented to particular primitive display types and each deciding which data objects it can render, provides a nice approach to some of the tricky problems in the C implementation of the VisAD display model. For example, various techniques are available for rendering flow fields or for rendering iso-contours of scalar fields, and the choice of technique may depend on the sampling associated with the fields. Fields with samples on a rectangular grid and fields with irregular samples are best rendered by different techniques. The new approach provides a flexible way to handle such decisions.

The design for the Java implementation of VisAD includes a general facility for *direct manipulation*, where users interact directly with graphical depictions of data objects. In particular, dragging points in displays causes changes in the data values depicted by the dragged points. Direct manipulation can be very useful for computational steering, where users interactively control parameters of computations by reaching into graphical depictions of those parameters.

5. CLASS LIBRARY

The VisAD interpretation of data objects as approximations to mathematical objects is natural for an object-oriented implementation. Abstract classes are defined for mathematical types, such as real numbers, vectors, sets and functions, and these are extended to subclasses for various kinds of approximations to these mathematical types. The abstract classes declare abstract methods for arithmetical operations and their subclasses implement those methods using algorithms appropriate for their approximations. For example, subclasses of the real number abstract class may be defined for floating point numbers, for double precision floats, and for rational intervals. The methods of these subclasses define various approximations to real arithmetic, such as the IEEE standards for single and double precision floating point numbers, and interval arithmetic for rational intervals. Similarly, subclasses of a real function abstract class are defined for approximations by finite samplings (i.e., fields), and methods are implemented which approximate

arithmetical operations on functions using various resampling and interpolation algorithms.

The VisAD abstract data classes declare abstract methods for all the unary and binary mathematical operations of the Java language, as well as all the special functions of the java.lang.Math class [Flanagan, 1996]. VisAD binary operations may take operands of different types, as long as some reasonable interpretation is available. For example, a real number added to a field is interpreted as addition of a constant to a function.

An object-oriented implementation of the VisAD data model provides a procedural interface to data that can hide the details of how data objects are stored. For example, very large data objects may be stored across multiple volumes, across multiple servers, or in tertiary storage. Data objects may also represent computations that are deferred until values are accessed.

The VisAD class library includes classes for real, integer and text string primitive data types. Instances of these classes include user-defined names and units. There are also classes for complex data types. A tuple instance contains a finite list of other types, a set instance contains a domain type, and a field instance contains domain and range types. A set or field domain type must be a real type or a tuple of real types. Each data object contains a data type (which must be consistent with its class) and an internal representation of the data object's value. The VisAD set class has numerous subclasses, which represent various kinds of finite samplings of n-dimensional real spaces. Each field data object includes an object of a set class to specify the sampling of its domain.

VisAD includes a class for displays, a class for mappings from primitive data types to primitive display types, a class for links between data objects and displays, a class for primitive display types, and classes for rendering algorithms. Each display object contains a set of mapping objects (that define a display frame of reference) and a set of links to data objects. A display object applies its mapping objects to its linked data objects to transform their primitive data type values into primitive display type values, then applies rendering algorithm objects to produce 3-D renderings. Multiple data objects can be displayed in the same display for visual comparison, and a data object can be displayed in multiple displays (i.e., multiple frames of reference).

The methods of the data object classes use the Java synchronization facilities so that multiple computational and display threads may safely share access to data objects.

Users can extend the data classes in order to define alternative forms of approximation. In the initial Java implementation of VisAD, fields are the only type of approximation to functions. However, the class library allows users to define other forms of approximation, such as the spherical harmonics used by some weather models. The field approximation could be extended to distinguish between range values that are point samples and averages.

Users can extend the data classes to define new forms of metadata. This may include virtually any information.

The set classes may be extended by users wishing to support new map projections or wishing to support image navigation standards such as the Spice Kernels and McIDAS.

Users can define new primitive display types and extend the rendering class to define ways of rendering these primitive display types.

6. DISTRIBUTED OBJECTS

The Java Distributed Object Model and its associated Remote Method Invocation (RMI) system provide access to remote objects in a way that is almost indistinguishable from access to local objects [RMI Web Page]. The Java implementation of VisAD uses RMI to support distributed computations and data objects whose components are distributed.

This will enable users at different locations to define display objects and link them to a shared set of data objects. For example, users will be able to visually monitor a shared computation. It will also enable comparisons of different observation sets and comparisons between observations and simulations regardless of their location on the network.

7. USER INTERFACE

Generic user interfaces are the curse of scientific analysis and visualization systems. Thus the VisAD class library does not include a user interface. Rather, the class library is intended to be the basis for various discipline-specific user interfaces, and several of these will be distributed with the class library. The class library may also be used directly by scientists writing software in Java.

We plan to eventually develop a new version of Vis5D based on the VisAD class library. Vis5D is used to visualize simulations of the atmosphere and oceans and is an example of a discipline-specific system [Vis5D Web Page]. The C implementation of VisAD was used to visualize Vis5D data sets in order to demonstrate the flexibility of its data and display models. However, the generic user interface of the C implementation of VisAD was not nearly as well suited to the task as the Vis5D user interface, which was custom-designed for the needs of environmental modelers. The VisAD class library will be adaptable to custom-designed user interfaces like Vis5D's.

The C implementation of VisAD was used for a variety of analyses and visualizations of time sequences of multi-spectral satellite images. We will use the VisAD class library to support user interfaces for a variety of satellite image processing tasks.

With its multiple displays, the VisAD class library will be capable of supporting an image processing spread sheet much like the IISS [Hasler, Dodge, and Woodward 1991; Hasler, Palaniappan, Manyin, and J. Dodge, 1994] and Ynot [Baltuch, 1994] systems. That is, each cell of a spread sheet would have its own VisAD display object to render data objects.

Some discipline-specific systems must support complex analysis computations, and therefore must include a general programming language in their user interface. The VisAD

class library will fit naturally with user interfaces that include the Java programming language. The VisAD distribution will include software tools for linking discipline-specific user interfaces to Java programs created using Java programming environments.

The VisAD class library will be used to support virtual reality user interfaces. We have used Vis5D for experiments with the CAVE virtual reality system [Hibbard, Paul, Battaiola, Tripoli, Pokrandt, and Cohen, 1994; Wheless, Lascara, Valle-Levinson, Brutzman, Sherman, Hibbard, and Paul, 1996], and will apply this experience to develop a CAVE interface for VisAD.

8. COLLABORATION

As discussed in Section 6, the VisAD class library will allow users in multiple locations to see displays of shared data objects. VisAD will also allow users at multiple locations to share control over a single user interface through the use of NCSA Habanero [Habanero Web Page] or Java Media Share [Java Media Web Page] from Sun (to be released in 1997). Habanero and Share enable users at multiple locations to share control of an application by coordinating user interface events among multiple copies of the same graphical user interface.

9. CONCLUSION

The design of the Java implementation of VisAD reflects lessons learned directly from the development of McIDAS, Vis5D and VisAD, as well as lessons shared by other system developers. The design stresses:

1. Platform-independence.
2. Flexible and extensible data and display models.
3. Programming support for users' analysis algorithms.
4. Support for discipline-specific user interfaces.
5. Exploiting the Internet through distributed computing and collaborative user interfaces.

The Java implementation of VisAD is only partially complete at the time of this conference. When it is ready for users, it will be freely available over the Internet and will include documentation.

10. ACKNOWLEDGEMENTS

We wish to thank Dave Fulker, Mitch Baltuch, Russ Rew, Glen Davis, Ian Foster, Tom DeFanti, Bob Wilhelmson, Crystal Shaw, Charlie Catlett, Joe Rueden, Fritz Hasler, K. Palaniappan and Mike Manyin for helpful discussions.

11. REFERENCES

Baltuch, M. S., 1994; Ynot - An Algorithm Development Tool for Spatial Data Analysis and Visualization. Preprints of the Tenth International Conference on Interactive Information and Processing systems for Meteorology, Oceanography and Hydrology. Nashville, Amer. Meteor. Soc., 48-52.

Flanagan, D., 1996; Java in a Nutshell. O'Reilly & Assoc.

Habanero Web Page. http://www.ncsa.uiuc.edu/SDG/Software/Habanero

Haber, R. B., B. Lucas and N. Collins, 1991; A data model for scientific visualization with provisions for regular and irregular grids. Proc. Visualization 91. IEEE. 298-305.

Hasler, A. F., J. Dodge, and R. H. Woodward, 1991; A High Performance Interactive Image Spreadsheet. Preprints of the Seventh International Conference on Interactive Information and Processing systems for Meteorology, Oceanography and Hydrology, New Orleans, Amer. Meteor. Soc., 187-194.

Hasler, A. F., K. Palaniappan, M. Manyin, and J. Dodge, 1994; A high performance Interactive Image SpreadSheet (IISS). Computer in Physics.

Hibbard, W., C. Dyer and B. Paul, 1992; Display of scientific data structures for algorithm visualization. Proc. Visualization '92, Boston, IEEE, 139-146.

Hibbard, W. L., B. E. Paul, D. A. Santek, C. R. Dyer, A. L. Battaiola, and M-F. Voidrot-Martinez, 1994; Interactive visualization of Earth and space science computations. IEEE Computer 27(7), 65-72.

Hibbard, W., C. Dyer and B. Paul, 1994; A lattice model for data display. Proc. Visualization '94, Washington, IEEE, 310-317.

Hibbard, W., B. Paul, A. Battaiola, G. Tripoli, P. Pokrandt, and S. Cohen, 1994; The Siggraph '94 daily weather forecast. Siggraph '94 Visual Proceedings, 263.

Hibbard, W., 1995; Visualizing Scientific Computations: A System based on Lattice-Structured Data and Display Models. *PhD Thesis*. Univ. of Wisc. Comp. Sci. Dept. Tech. Rpt. #1226; ftp://iris.ssec.wisc.edu/pub/lattice/

Hibbard, W., and B. Paul, 1995; Computational Steering with VisAD. Siggraph Video Rev., No. 108, ACM.

Hibbard, W., J. Anderson, and B. Paul, 1997; Java and the World Wide Web: the right choice for interactive systems. Preprints, Conf. Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology. Long Beach, Amer. Meteor. Soc.

Java Web Page. http://java.sun.com/

Java Media Web Page. http://java.sun.com/products/apiOverview.html

Moore, R. E., 1966; Interval Analysis. Prentice Hall.

RMI Web Page. http://chatsubo.javasoft.com/current/rmi/index.html

Vis5D Web Page. http://www.ssec.wisc.edu/~billh/vis5d.html

VisAD Web Page. http://www.ssec.wisc.edu/~billh/visad.html

Wheless, G. H., C. M. Lascara, A. Valle-Levinson, D. P. Brutzman, W. Sherman, W. L. Hibbard, and B. E. Paul, 1996; Virtual Chesapeake Bay: interacting with a coupled physical/biological model. IEEE Computer Graphics and Applications, 16(4), 52-57.