3.5 INTEGRATING SCIENCE METADATA INTO PYTHON USING VISAD

William L. Hibbard*, Thomas M. Whittaker, and Curtis T. Rueden
University of Wisconsin, Madison, Wisconsin

1. THE VISAD DATA MODEL

The initial design motivation of the VisAD system was to enable visualization of any data defined in a scientific program, without the need for users to custom-code display procedures for new data structures they invented as they wrote their programs. The solution was a data model with schemas that could express any data defined in common programming languages, combined with a display model based on mappings from primitive data types to primitive display types that could be used to derive displays of complex data types (Hibbard, Dyer and Paul, 1992; Hibbard, 1995).

The VisAD data schemas were patterned after the two primary data-building features in common programming languages: tuples and arrays. Tuples are fixed length sequences of simpler data types, which may have a variety of different types (the obvious example is the C language *struct*). Arrays are variable length sequences of a simpler data type, where all array elements have the same type (while this is not true in all programming languages, it is true in commonly used scientific languages such as Fortran, C and C++).

In order to adequately support the VisAD display model, the data model had to incorporate certain other information absent in programming-language data structure declarations. For one thing, VisAD primitive types have names such as *time*, l*atitude*, *temperature* and *pressure* rather than simply being declared as *float* or *real*. These primitive names are used to define mappings to primitive display type names (such as *XAxis*, *RGB, Animation* and *IsoContour*). It is a reasonable assumption that values in scientific programs had such names, but they just aren't expressed because programming languages do not provide ways to express them. Primitive type names are one form of metadata integrated into the VisAD data model.

Corresponding author address: William Hibbard, SSEC, 1225 W. Dayton St., Madison, WI 53706

Arrays in scientific programs generally represent finite samplings of continuous functions, but common programming languages provide no way to express this. For example, images and grids are stored in arrays, with one pixel or grid point per array element. These images and grids are really just finite samplings of continuous fields of radiances, temperatures, etc. The VisAD data model uses finite samplings of functions in place of arrays. Thus for example a simple earth image is represented as a finite sampling of a function from *latitude* and *longitude* to *radiance*. This is expressed by the VisAD schema:

$$((latitude, longitude) \rightarrow radiance)$$

This representation allows navigation information defining earth locations to be carried with image pixels, so that an implementation of the VisAD display model knows where to locate pixels on earth-based display coordinates. Note that arrays which are not finite samplings of continuous functions can be represented by samplings of their indices at integer values, which corresponds well to array information in common programming languages. The important thing is that for data like images and grids, spatial and temporal metadata are integrated in VisAD's representation and accessible by the display model.

Earth navigation can also be represented by transforms between coordinate systems in VisAD. For example, an earth image can have the schema:

$$((line, element) \rightarrow radiance)$$

coupled with the invertable coordinate transform:

$$(line, element) \leftrightarrow (latitude, longitude)$$

The VisAD data model allows units to be attached to numerical values, so for example *latitude* and

longitude values may have units of degrees or radians.

A time sequence of earth images may have schema:

(*time* → ((*line*, *element*) → *radiance*))

This representation would include a finite sampling of *time* values, typically with units of seconds since 0 GMT, 1 January 1970. It would also include a finite sampling of (*line*, *element*) values, typically a 2-D integer grid, with an invertiable transform to (*latitude*, *longitude*) for earth navigation. Note that the (*line*, *element*) sampling and the invertible transform to (*latitude*, *longitude*) may vary between *time* samples.

2. THE PYTHON INTERFACE TO VISAD

The VisAD data model integrates metadata not expressed in common programming languages, but it would be useful to make such metadata accessible from scientific programs. VisAD version 1, implemented in C, solved this problem by defining its own programming language based on the data model. However, this private language was not popular with users. VisAD version 2, implemented in Java, solved this problem by defining a library accessible from ordinary Java programs. However, this does not integrate access to the VisAD data model with ordinary Java syntax of operations on data. Rather, all access is via library method calls.

Now we have integrated the Python programming language with VisAD in order to provide users with a simple scripting language interface to the VisAD library, and have been able to integrate access to VisAD metadata with Python syntax. We have integrated Python with VisAD via Jython, which is a Java implementation of Python that enables access to any Java objects and methods from Python. Jython provides a set of method signatures that can be implemented by any Java object to define the meaning of Python syntax such as +, -, * and / for arithmetical operations, and [.] for array access. Implementations of this syntax can include implicit access to metadata. Unit conversions, coordinate transforms and resamplings can all be done implicitly during arithmetical operations. Given two *latitude* values, *lat1* in degrees and *lat2* in radians, the implementation of + in "*lat1* + *lat2*" converts the *lat2* value to degrees before adding the two values.

Although images and grids are Java objects, implementations of appropriate Jython methods allow them to be accessed using Python array syntax. Thus the pixels of an earth image can be accessed with the syntax *image*[*i*] where *i* is an integer. Furthermore, (*latitude*, *longitude*) pairs are objects. If the variable *lat_lon* references such an object, the earth image can also be accessed by the syntax *image*[*lat_lon*]. This syntax can be used to write the Python program:

```
area = load("AREA2001")
map = load("OUTLSUPW")
i = 100
print "area = ", area[map[i]], " at ", map[i]
```

The first two lines of this program read McIDAS image and map files into the variables *area* and *map*. The third sets the variable *i* to 100, and the fourth prints something like:

area = (78.12268) at (53.75, -9.5)

The popular Matlab and IDL scripting languages allow arithmetical operations on arrays, and scientists use this to work with entire images represented by arrays of pixel values. That is, given two arrays of pixels referenced by *image1* and *image2*, they can be added by the syntax:

image3 = *image1* + *image2*

In Matlab and IDL, these are just arrays of pixel values without any associated units or earth navigation. So if the images have different units or earth projections, the system cannot adjust for these differences. Rather, it is the responsibility of the programmer to explicitly manage units and earth projections. But for the same addition of images in a Python script using VisAD, units will be converted before the addition if *image1* and *image2* use different units, and a coordinate transform and/or resampling will be applied implicitly if the two images have different earth projections.

3. CONCLUSION

A current goal of the VisAD system is to combine:

1. A scripting language in the style of Matlab and IDL for interactive analysis.

2. Displays that overlay data from multiple sources geographically and temporally, in the style of McIDAS.
3. 3-D displays of model data in the style of Vis5D, plus an easy ability to define novel 3-D displays for new types of data.
4. Remote collaboration in the style of VisitView.

Combining the capabilities of these systems into a single system will be useful to environmental scientists. The integration of science metadata into Python syntax is important for combining the first two items on this list, the capabilities of Matlab/IDL and McIDAS.

4. REFERENCES

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., 1995; Visualizing Scientific Computations: A System based on Lattice-Structured Data and Display Models. *PhD Thesis*. Univ. of Wisc. Comp. Sci. Dept. Tech. Report #1226.