Astronomical Data Analysis Software and Systems IV
ASP Conference Series, Vol. 77, 1995
R. A. Shaw, H. E. Payne, and J. J. E. Hayes, eds.

Propagating Uncertainties and Units in Data Structures

J. McDowell and M. Elvis

AXAF Science Center/Smithsonian Astrophysical Observatory, 60 Garden Street, Cambridge, MA 02138

Abstract. We describe a possible data structure designed to improve propagation of physics information in an analysis system, and an associated subroutine library for combining physical units.

1. Rationale

The Advanced X-ray Astrophysics Facility (AXAF) is a sophisticated X-ray observatory scheduled for launch in 1998. As part of the combined AXAF pipeline processing and data analysis system being developed at the ASC (AXAF Science Center), we are investigating possible data structures for the ASC data system. Our goal is to provide a system which treats scientific data as rigorously as possible, maintaining the integrity of the auxiliary information supplied with the data. Some existing systems propagate some kinds of auxiliary information (e.g., coordinate systems), with varying degrees of success; the multiwaveband quasar database analysis system developed by one of us (JCM) demonstrated the need for flexibility in handling and converting heterogeneous data in various units and coordinate systems when datasets taken in different wavebands are to be combined. Making the software do the bookkeeping reduces workload on the astronomer and improves the archival quality of the data products. These considerations led us to concentrate on some object-oriented concepts.

We now introduce a definition of a Physical Data Object. This prototype data structure is optimized for the requirements of our X-ray data analysis system, but has been designed with multiwaveband support in mind. It should be emphasized that this prototype is one of a number of possible approaches and does not represent our final design. We have proven some of the concepts discussed here via rapid prototyping and will implement full versions over the next year.

2. The Physical Data Object

The accompanying entity-relationship diagram shows the full structure of the Physical Data Object (PDO) prototype concept. (The diagram shows entities as rectangles, attributes as circles, two-way relationships as diamonds, and 'is-a' (A is an example of B) relationships as rounded rectangles.) The PDO is considered to be a 1-dimensional vector of length N each element of which is an M-dimensional array, together with associated attributes. The most common cases are M=0, i.e., a 1-D vector of scalars, and (N=1,M=2), i.e., an image.

A "Physical Data" Table is a collection of PDOs of the same length which share the same data subspace (see below); it is the analog of the FITS binary table. Any attribute of the object may have the value 'Not Present' and software will deal with that case. Support for Null (NaN) values and upper limit flags for numerical quantities will be provided at a low level. Arrays of data could be stored either explicitly or implicitly as coefficients of a functional expansion, as done in the UK's Asterix/NDF system. We would support three different kinds of uncertainty (statistical, and systematic scale and offset) to allow a greater degree of control over error propagation. Each data value is considered to be stored in a 'local' coordinate system, and the WCS (World Coordinate System) allows it to be registered on a 'global' system. We do not allow WCS to provide world coordinates mixing different physical objects; we instead support multidimensional objects so that, for example, X and Y positions are stored as a single 2-D object.

We also introduce a Data Subspace infrastructure which is a little different from WCS; it records from where the data was extracted (including coordinates which are no longer in the data). For X-ray data this will typically include an instrument ID, a set of good time-intervals, a set of detector coordinate regions versus time, and a pulse-height range. The Data Subspace allows one to combine different datasets. Note that in practice the detector coordinate regions will usually be specified with a single region fixed in world coordinates coupled with a WCS that varies with time (the 'aspect history'). The Data Subspace library will also support a Bad Value Mask, an integer vector which can be used to temporarily mask elements of the data vector, allowing rapid re-screening.

The PDO might also support an Exposure Map Array: this array is usually defined implicitly rather than carried around with the data. It has the size and shape of the physical data array, and its pixel values represent the exposure time for that element of the array. It is generated by projecting the data subspace on the relevant axes.

3. Software Library Infrastructure

To use the proposed data objects, we would require a set of libraries to provide services. A PDO I/O library would provide routines to get and put PDOs and each of their sub-objects. A Unit handling library parses physical unit definitions, and provides simple routines to combine them. An Uncertainty handling library handles simple cases of combining uncertainties, and contains output routines to combine available uncertainties into a single quoted error estimate. Applications would still need to provide their own formulation of the correct propagation of uncertainties and units, but the libraries provide a simplified notation for doing so (e.g., a routine unc_rms(A,B) might combine the uncertainties in a stack A of objects and assign the root mean square of the result as the uncertainties for the result object B). An Implicit Array library supplies an interface to make functionally defined data items look like a simple tabulated array. A WCS library will handle propagation of WCS data. Finally, a Data Subspace library would handle adding data subspaces, projecting them and extracting subsets of them.

4. Storing Physical Objects in FITS

We want to be able to archive PDOs in FITS format. One approach would be to use the ability of FITS BINTABLES to store multidimensional objects and keep the uncertainties together with the data values in a single 'Item' column. We feel this would make it harder for simple programs to access the data. We therefore intend to take the approach of storing each axis of the data (e.g., X, Y values) and each uncertainty in separate columns (or keywords for scalars), and encoding the connection between them in special header keywords. This means that other software systems reading the file will lose the PDO superstructure, but will see the actual data in a familiar form. Our goal is for the resulting data files to be valid HEASARC type FITS files where appropriate, with extra header keywords and columns that the current HEASARC software would ignore. Our design would enable backward compatibility with any system that accepts such HEASARC FITS files.

5. The Prototype ASC Units Library

Our prototype units handling library is now being tested. The prototype is written in Fortran and has been tested on a SPARC machine running SunOS Unix. The operations we support are: (1) multiplication and division of units, (2) raising of units to a real power, and (3) reducing compound units to the base units which define them. We use the following syntax to manipulate units:

$$m \times 10^{e_0} u_1^{e_1} u_2^{e_2} ... u_n^{e_n}$$

For example, a simple valid unit is cm and a more complicated one is

```
6.67x10^{-11} m^3 s^{-2} kg^{-1}:
```

The braces are optional, but allow the output string to be directly typeset by TEX or compatible output programs (e.g., the SM plotting package which supports TEX syntax in its axis labels).

Simple use of the program requires no knowledge of the units, treating the individual units as simple tokens. The user may supply a Unit Definition File, simply an ASCII file with a list of known units, some of which are defined in terms of others. This allows the program to parse SI prefixes (without this it doesn't know whether 'pc' is a parsec or a pico-c) and, on request, to resolve compound units.

Since the choice of which units are compound and which are fundamental is specified at run time with a Unit Definition File, the library is very flexible. This is illustrated by one example supplied with the library, in which the unit definition file specifies the base units as 'm' and 'yr', and defines 'pc' and 's' in terms of them. This allows the software to divide the units `1` and `50 km s^-1 Mpc^-1` to obtain the result `1.96x10^{10} yr`, the Hubble time.

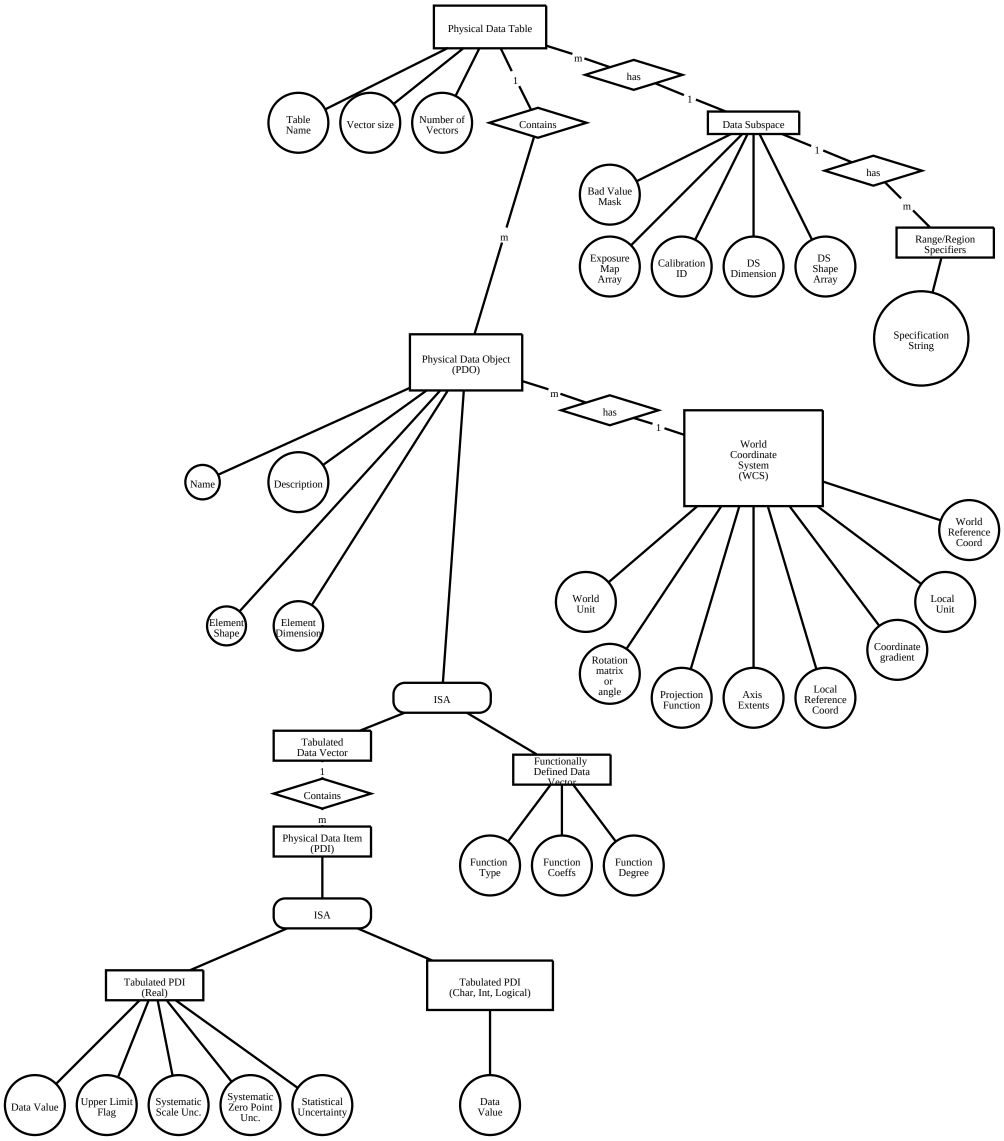

Figure 1. Structure of a Physical Data Object.