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.

An Abstract Data Interface

D. J. Allan

Department of Physics and Space Research, University of Birmingham, UK

Abstract. The Abstract Data Interface (ADI) is a system within which both abstract data models and their mappings on to file formats can be defined. The data model system is object-oriented and closely follows the Common Lisp Object System (CLOS) object model. Programming interfaces in both C and FORTRAN are supplied, and are designed to be simple enough for use by users with limited software skills.

The prototype system supports access to those FITS formats most commonly used in the X-ray community, as well as the Starlink NDF data format. New interfaces can be rapidly added to the system—these may communicate directly with the file system, other ADI objects or elsewhere (e.g., a network connection).

1. Introduction

The Asterix X-ray analysis package has hitherto dealt solely with data belonging to the Starlink Hierarchical Data System (Warren-Smith & Lawden 1993). In an attempt to broaden the scope of the package it was decided to add FITS (NASA Office of Standards and Technology 1993) compatibility at a low level, rather than simply providing format conversion software. As this addition involved a rewrite of the existing interface library to binned data, a general redesign of the way applications access their data was undertaken.

Two important goals of the data interface redesign were to remove all file-format specific code from general purpose applications, and to make adding new functionality to the data interface less time consuming. In the old system, FORTRAN common blocks would usually be changed, necessitating a rebuild of the entire data interface. It was also important to retain efficiency of access to the various data formats which have different natural modes of access. For example, HDS is poor at representing tabular data and software using HDS tends to be strongly column oriented, whereas FITS more efficiently supports row access. We also needed to improve fault tolerance. When a user has a file whose components do not match what the data interface expects, the system should be able to cope, preferably by the user editing an ASCII text file, rather than altering the dataset. We also wanted to simplify the programming interface to encourage novices to write analysis tasks, and to provide FORTRAN and C interfaces. (The demand for a C interface has been growing steadily.) Finally, we wanted to ensure that the redesign would not have to be repeated too soon!

2. Data Models

ADI is a system for defining "abstract data models." These models generally provide a "view" of some underlying object, such as a data file whose structure we want to conceal from application software. Different views of the same underlying object are possible, but the most useful feature as the ability to support the same view of different objects.

A data model consists of a number of "slots" which have a name and a value. Figure 1 shows a schematic view of the `Array` abstract model. The

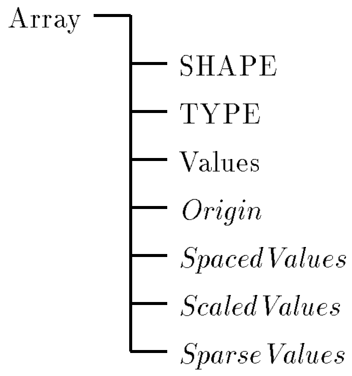

Figure 1. Structure of the `Array` abstract model

names in upper case denote slots which are required to define a new instance of a particular model. The `Array` definition allows for the possibility of unset values, so the `Values` slot does not fall into this category. Names in normal type but capitalized are those slots which a user should expect to be defined for any instance of an `Array`. Names in italics mark slots which need not have defined values for any instances of an `Array`. The slot values may be any of the usual scalar types (integers, floats, strings and logicals), arrays of these scalars, or any other ADI object. ADI can chain together data models to provide further layers of abstraction, e.g., an "image" view of an event table is possible if there exist both spatial coordinate lists for the events and some default spatial bin size.

In Asterix a further layer of abstraction, this time a procedural one, is used to insulate the application from details of a particular abstract model. A data model object which is not linked to anything else exists purely as a construction in memory. This can be very useful as an aid to the application developer, as dynamic objects of arbitrary complexity can be created, obviating the need for the "make the arrays as big as we'll ever need" style of programming. An image processing application, for example, could work by maintaining a stack of X-Y image objects, allowing operations on the data to be undone if required.

3. Implementation

ADI was designed as an object-oriented system where data models are defined as classes. The object model chosen is that of the Common Lisp Object System (CLOS) (Bobrow 1988). The building blocks of the ADI system are primitives, classes, instances, generic functions and methods.

Primitives ADI supports most of the basic types used in FORTRAN and C, as well as arrays of these types with up to 7 dimensions.

Type Name	Description	Short-code
_UBYTE	Unsigned byte	UB
_BYTE	Signed byte	B
_UWORD	Unsigned word	UW
_WORD	Signed word	W
_INTEGER	Signed integer	I
_REAL	Single precision F.P.	R
_DOUBLE	Double precision F.P.	D
_CHAR	Character string	C
_LOGICAL	Logical	L

Table 1. Supported primitive data types.

ADI will perform type conversion when the access type for a primitive does not match its storage type. ADI also allows new primitive types to be defined.

Classes A "class" is a form of data structure which is a collection of "slots." Each slot has a name and a value (which may be null). A particular object of a class is called an "instance." ADI has several built-in classes which it uses to manage its own internal data storage. A class can inherit the slots and behavior of existing classes in the usual CLOS fashion.

ADI data models are implemented as ADI classes which are derived from the class `ADIdataModel` either directly by inclusion in their superclass list, or indirectly by inheriting a class which is thus derived. Inheriting this class enables a new class to act as the viewing object in a data model chain (classes do not have to be so derived to be the rightmost element in the chain).

Generic Functions The "generic" function in ADI is a function definition with no implementation. The term generic is used because the arguments to the function can be of any type. The implementation of a generic function is distributed over a set of "methods."

Methods ADI lets the user define methods to perform all the built in ADI generic functions (which are used to access data models and their slots), as well as defining new generic functions. A method can be specialized to work on particular classes, classes with particular objects to the right in the ADI chain, or even a specific ADI object.

The CLOS method forms "primary," "around," "before," and "after" are all supported. Methods are executed following the CLOS method combination rules (Bobrow et al. 1988; Keene 1989).

4. Status

A general purpose library for defining and using data models has been constructed. ADI is completely configurable at a low-level, but through its support for complex data models can provide a very high-level data interface. It encourages the application programmer to concentrate on the data objects being manipulated and the relationships between them. The result of the changes when applied to Asterix will be package which processes data according to a given set of data models, rather than a package which processes HDS files.

It is anticipated that a stand-alone release of ADI will be available by the end of 1994; releases of Asterix after that date will incorporate ADI in an increasing fraction of the applications.

References

Bobrow, D. G., DeMichiel L. G., Gabriel, R. P., Keene, S. E., Kiczales, G., & Moon, D. A. 1988, Common Lisp Object System Specification, X3J13 Document 88-002R

Keene, S. E. 1989, Object-Oriented Programming in COMMON LISP (Reading, Addison-Wesley)

NASA Office of Standards and Technology 1993, Definition of the Flexible Image Transport System (FITS) (Greenbelt, NASA/OSSA)

Warren-Smith, R. F., & Lawden, M. D. 1993, HDS—Hierarchical Data System, Starlink Document SUN 92