Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

A Framework for the Development of Multi-Mission Software

J. E. Davis

Center for Space Research/CXC, Massachusetts Institute of Technology Cambridge MA, 02139

Abstract. A model for producing more maintainable multi-mission software is presented in the context of `mkarf`, the program that produces ancillary response files (ARFs) for the Chandra X-Ray telescope. The basic strategy employs the methods of object-oriented programming, and consists of encapsulating the mission-dependent pieces into subclasses that derive from more mission-independent abstract base classes. In particular, `mkarf` interacts with mission-specific calibration data via a library called `ardlib`, which provides a mission-independent class interface to objects that represent effective areas, quantum efficiencies, and so on. For this reason, much of the discussion centers upon the design of `ardlib` itself.

1. Introduction

One of the problems that I encountered in writing `mkarf`, the Chandra ARF generation program, was the difficulty in adapting the existing ARF generation software from other missions to produce ARFs for Chandra X-Ray telescope. Unfortunately, this obstacle proved insurmountable and I was left with no alternative but to create yet another program to produce ARFs. However, I decided from the outset that this one would be different. It would be one that could be easily adapted to other missions, whether they be future, past, or present. In other words, I wanted to write a mission-independent ARF generator.

An ARF (George et al. 1998; Davis 1999) is actually a very complicated object that represents the energy-dependent effective area of the telescope. This value not only depends upon several important pieces of instrument-specific calibration information, it also depends upon off-axis angle as well as the detailed dither of the telescope during the observation. Given these dependencies, it is no small wonder that there seems to be a separate ARF program for each mission and instrument combination. Nevertheless, I feel that the goal of producing a reasonably mission-independent ARF generation program has been reached.

The approach that I took with `mkarf` was to bury anything explicitly mission or instrument-specific inside mission-independent libraries. Although this sounds like a paradox, the basic ideas are not new and fall under the guise of object-oriented programming. In this paradigm, the design and implementation of mission-independent libraries that perform mission-dependent calculations take center stage.

The next part of this presentation discusses the design of one of the most important libraries used by `mkarf`, namely `ardlib`, which is a library that deals with the calibration data. Following the description of the library is a discussion of some of the problems that a mission independent library introduces and how these problems are dealt with by `ardlib`. A brief conclusion appears at the end.

2. Ardlib: a Mission-Independent Calibration Library

`Ardlib` is the library that `mkarf` uses to compute quantities such as mirror effective areas, detector quantum efficiencies (QEs), grating efficiencies, bad pixel maps, and so forth. It was designed from the outset to support multiple missions in a clean and mission-independent manner.

The fundamental idea is to recognize that objects such as effective areas and QEs have meanings that are independent of any specific instrument or mission. For example, both a Chandra ACIS quantum efficiency and an ASCA GIS quantum efficiency are representatives of a QE. It is for this reason, using the terminology of object-oriented programming, that `ardlib` implements an instrument-specific object as a *derived class* of some more fundamental *abstract base class*. The `ardlib` application programming interface, or API, contains references only to the abstract base classes but not to the mission or instrument specific derived classes. In other words, the `ardlib` API is manifestly mission-independent.

Although I have used the parlance of object oriented programming to describe the implementation of these objects, for maximum portability the code itself is written in ANSI C, with the abstract base classes appearing as opaque pointers in the API. Inside the library, a base class is implemented as a structure that contains function pointers that get set upon instantiation of the derived class. Since the source code to `ardlib` is freely available as part of the Chandra Data System, the actual details of the implementation will not be discussed here.

It is instructive to look at a concrete but simple example that illustrates the use of the `ardlib` API. The following example computes the QE at a specified energy and pixel location for a specified detector and mission. For clarity, error checking has been omitted.

```
int compute_qe (char *mission, char *detector,
                float energy, float xpixel, float ypixel,
                float *qe)
{
   Ardlib_Det_QE_Type *q;

   ardlib_initialize (mission, NULL);
   q = ardlib_open_det_qe (detector);
   ardlib_compute_det_qe (q, x, y, &energy, 1, qe);
   ardlib_close_det_qe (q);
   ardlib_finalize ();
   return 0;
}
```

Although simple, this example illustrates a few points worth noting. First of all, before the library can be used, it must be initialized for a particular mission. This is what the call to `ardlib_initialize` accomplishes. Similarly, the function `ardlib_finalize` is used to de-initialize the library after its services are no longer required. Secondly, the data type `Ardlib_Det_QE_Type` is an opaque type that represents the abstract mission-independent base class. The call to `ardlib_open_det_qe` instantiates an object of the mission-dependent derived class and that object is passed to the `ardlib_compute_det_qe` function which in turn calls the appropriate function to compute the QE for the selected detector. Internally this is handled via the code:

```
int ardlib_compute_det_qe (Ardlib_Det_QE_Type *qet,
                           double x, double y,
                           float *energies, unsigned int num,
                           float *qes)
{
   if (qet == NULL)
     return -1;

   return qet->compute_det_qe (qet, x, y, energies, num, qes);
}
```

Similar functions exist for computing mirror effective areas, grating efficiencies, and determining whether or not a given pixel is bad.

Adding a new mission to `ardlib` is relatively straightforward. Briefly, it involves adding a mission name to a supported mission table along with functions to initialize and de-initialize the mission-dependent code. Beyond that, all that is required is to implement the derived classes.

3. Pros and Cons of Mission Encapsulation

One of the added advantages of the object-oriented methodology described above is that the actual implementation details of a particular mission have been hidden behind the API. For example, in its current implementation, `ardlib` computes a Chandra mirror effective area by interpolating on a set of points stored in an external fits file. If the effective area algorithm were to change to the evaluation of some analytic function representing the effective area, then code that makes use of the library would not have to be modified. In this respect, *external objects such as calibration data files are really reflections of the underlying implementation.*

To ensure proper data encapsulation, `ardlib` has its own parameter file which serves to hide implementation-dependent parameters from application code using the library. Of course the user will want the ability to specify which calibration data to use for a particular calculation. For maximum flexibility, the `ardlib` API contains a function (`ardlib_set_par_file`) that allows the main application to specify which particular parameter file to use. By default, `ardlib` uses `ardlib.par`. The downside is that the user will have to deal with at least two parameter files: the main application's parameter file, e.g. `mkarf.par`, and the `ardlib` parameter file.

The only alternative to the use of multiple parameter files would be to have all the parameters in the parameter file associated with the program. This would not only clutter the parameter file with many parameters that the user is unlikely to change (or even fully comprehend), it also means that if the library's implementation were to change, the application's parameter file would have to be modified to reflect the changes to the library. This leads to a software maintenance problem because, in addition to updating all the parameter files for applications that reference the library, the documentation for those programs would also need to be modified. Hence, the use of multiple parameter files that are closely tied to the libraries which utilize them is a *good* thing because it avoids these pitfalls.

It is important for a program to have the possibility of creating a record, or a history, of what files were used to produce its end product. The disadvantage of data encapsulation is that this information is not always readily accessible because it is hidden by the interface. To get around this problem `ardlib` provides a function as part of its API that the application can use to retrieve a record of the data files and other processing options that were used for its computations.

4. Summary

In this work, I described an object-oriented software design paradigm for the creation of mission-independent software. I have tried to show that mission-independence can be achieved through the proper encapsulation of mission-specific details. An added benefit of this model is that it lends itself to simpler software maintenance.

The `mkarf` program and the mission-independent calibration library `ardlib` are currently part of the Chandra data system release.

Acknowledgments. I would like to thank David Huenemoerder for useful discussions during the design phase of `ardlib`. The software was conceived and developed at the MIT Center for Space Research under Chandra X-Ray Center contract SV1-61010 from the Smithsonian Institution.

References

Davis, J. E. 1999, to be published. A preliminary version is on the web from http://space.mit.edu/%7Edavis/arf99.html

George, I. M., Arnaud, K. A., Pence, B., & Ruamsuwan, L. 1998, OGIP Calibration Memo CAL/GEN/92-002