Redesign and Reimplementation of XSPEC

Ben Dorman and Keith Arnaud

Laboratory for High Energy Astrophysics, Code 664, NASA/Goddard Space Flight Center, Greenbelt, MD 20771 [1,2]

Abstract. We present a progress report on the redesign of XSPEC. Work has been underway since late 1998 to produce a new version of XSPEC with a code base written in ANSI C++ and with modern design techniques. The new version (XSPEC 12) is expected to be ready for release in 2002.

The new version of the program will implement most of the features of the current release, XSPEC 11. The design allows for both the current tcl command line interface as well as an optional GUI. The existing scheme whereby users can write theoretical model components in FORTRAN 77 will, however, continue to be fully supported.

Scientifically, the major change will be to support simultaneous fitting to spectral data containing multiple sources, required for coded mask instruments (such as Integral/SPI that are able in principle to separate superposed X-Ray sources). The focus of the project is however to update the program to take advantages of the progress in programming technology that has been made in the field of Computer Science over the last 15 years since XSPEC was originally released.

The main software engineering benefits of the redesign project are an extensible implementation that allows for (a) loosely coupled modules [providing ease of maintenance], including CCfits, a new ANSI C++ interface to the `cfitsio` library developed as a by-product; (b) extension of the supported data formats by 'add-in' user-loadable modules; (c) an independent user interface implementation (d) robustness guarantees such as exception safety.

1. Motivation

XSPEC, originally developed by Rick Shafer at Cambridge University in the early 1980s (see Arnaud 1996 and references therein), was designed to be a mission-independent general purpose analysis program for X-Ray spectral data. XSPEC's general mission support requires the adoption by X-Ray missions of

---

[1] Raytheon Information Technology & Scientific Services and Emergent Information Technology, Inc.

[2] Department of Astronomy, University of Maryland, College Park, MD 20742

a set of standard formats for representing X-Ray data that are based on the instruments and requirements of the last fifteen years of science spacecraft.

XSPEC (current version is 11.0.1) manipulates theoretical models, X-Ray source and background data, and calibration data. Each of these form natural objects for the Object-Oriented programming paradigm, as they have state, behavior, and identity. Encapsulating the abstract behavior of data and models allows the extension of XSPEC capabilities to provide general mission support well into the future. As well, XSPEC's internal memory model is somewhat limited by its origins as a statically allocated FORTRAN program, which makes extension to some future needs (e.g. simultaneous fitting of composite data sets to multiple theoretical models) technically difficult and error-prone.

It was decided in late 1998 to re-engineer XSPEC's data structure handling in ANSI C++. C++ supports Object Oriented programming, integration with legacy FORTRAN code, and support for generic algorithms and support for numeric computation (the `valarray` and related classes). We expect, with improved compiler implementations expected in future, these will become at least as efficient, if not more so, than FORTRAN implementations.

Modernization of XSPEC's user interface started with XSPEC 10's adoption of the `tcl` scripting language as its command interpreter. This allows a natural interface to a GUI written with `tk` widgets (as used for the AIPS++ package). ANSI C++'s stream IO facilities allow a user interface which is largely decoupled from internal workings of the program, allowing GUI development side-by-side with the existing command line interface.

This article describes some of the techniques incorporated in the design for the re-engineered XSPEC 12, which is expected to be released in 2002.

2. Methods and Tools

We have adopted Solaris 7 as our development platform because of the availability of a near ANSI-compliant C++ development environment (Workshop 6.0/C++ 5.1), tools such as Rational Purify for efficient resource and access checking, and Rational Rose for design. Rose has the advantage of making class hierarchies and interclass interactions simple to develop and maintain where needed, which we have found has a strong positive effect on code design. Disadvantages include expense and a degree of unreliability, at least in the current Unix implementations.

3. Code Features

3.1. Data Package

The XSPEC data library design makes use of three design patterns (Gamma et. al 1995) in order to allow a flexible upgradeable capability to read and process X-Ray data. Design patterns are described as "elements of reusable object-oriented software" and represent general solutions to common design needs.

The needs XSPEC has in data ingress are:

- Execute a discrete set of steps in processing a data file: read, perform grouping and quality selection operations, select and read background and

response data, and compute statistical quantities (variance, systematic errors) for the analysis.
- Read a set of files with conforming data format.
- Allow for the possibility of adding new data formats without the necessity of modifying the existing code.

These goals are achieved by using respectively the Template, Abstract Factory, and Prototype patterns. Each format is represented by a set of concrete classes, one for each of the files within a given data format (e.g. the widely-used OGIP format has files of type PHA – used for source and background files, RSP – for response files and ARF – for effective areas). These classes have the responsibility of performing the data processing steps. The Abstract Factory technique provides the functionality for minting a set of class instances (PHA, RSP, ARF) with conforming formats. Finally the Prototype pattern is used to store prototypes for formats that the program recognizes. Each file that the user requests to be read is checked against this list, and class instances of the correct type are created and processed. The key advantage of this pattern scheme is that the list of file formats that can be recognized is dynamic. Code can be loaded in shared library modules at run-time, adding to the list of recognized formats without modifying the existing code base. The Template Method specifies and orders the set of operations that must be implemented by the format-specific subclasses.

3.2. Theoretical Models Packages

The theoretical model processing packages comprise class hierarchies dealing with model components and parameters, and parsing classes.

An XSPEC theoretical model is composed of a number of 'additive' Component Groups that represent X-Ray sources or combinations of sources whose radiative spectrum is modified by other physical processes (e.g. absorption, represented by a 'multiplicative' model). Component Groups are composed in turn of objects representing sources and absorption (etc) processes. In the code, these are implemented by a set of classes derived from an abstract Component class. These are instantiated using a parameterized Factory Method, where by the type of the model component as specified in the input file determines the class of the component object created.

The Component Class on construction selects a pointer to a function that generates the modeled flux array from an input set of energies: this function is typically written in FORTRAN 77. Thus, XSPEC12 will continue to support user's ability to extend XSPEC according to their own needs without their needing to learn C++.

The parser class implements the reading of XSPEC's model expressions, and a subclass processes expressions linking parameters. The implementation extends XSPEC's current capabilities in a natural way to allow much more complex nested model expressions and parameter linking capabilities (for example, a parameter may be set as the product of two existing parameters).

3.3. User Interface

The user interface in XSPEC12 is implemented by deriving from the C++ IOstream library. The class XSstream is derived from `std::iostream` and contains an XSstreambuf object derived from `std::streambuf`. These classes add a pointer to an I/O channel class, XSchannel, to the standard streams. This implementation allows the code to communicate with its user interface through the usual shift operators. The XSchannel pointer is a interface class that mandates the implementation of any I/O device. We are implementing `tcl` and `tk` output channels, so that depending only on a flag set by the user, I/O may be performed through a command line or a widget interface.

4. FITS I/O: The CCfits Library

Xspec makes extensive use of FITS format I/O. Each individual file format that XSPEC recognizes may use files of five different types, all of which currently require separate code to be maintained, and also duplicated, if new formats are added.

As part of the project we have designed and (partially) implemented a class library which wraps calls to the widely-used `cfitsio` library developed in our Laboratory. This code is independent of XSPEC and can be build as a shared library. It is implemented with the Standard Library containers and algorithms (the "Standard Template Library" or STL) and is in principle portable to any platform that supports the Standard up to member template functions and template partial specializations.

In the CCfits model, a FITS object is created when a FITS file is opened, and data read from the file is loaded into the object which serves as a memory image of the file. Extensions (Images or Tables) are accessed either by index, extension name, or by match a set of scalar header keys. User code reading FITS extension consists only of supplying a list of keyword and column names to be read.

As simple examples, take the code lines

```
CCfits::FITS dataSource("dSource.fits","HEADER1");
CCfits::FITS dataSource("dSource.fits","HEADER1",true);
```

The first of these will read the following from the file `dSource.fits`: the mandatory primary header keys; the mandatory header keys of the extension `HEADER1` and the column specification for that header. Subsequent calls can be used to obtain the data from the file. The second example will additionally read the primary image and the column data in that extension on initialization.

We expect to be able to release the CCfits library in 2001.

References

Arnaud, K. A. 1996 in ASP Conf. Ser., Vol. 101, Astronomical Data Analysis Software and Systems V, ed. G. H. Jacoby & J. Barnes (San Francisco: ASP), 17

Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995 *Design Patterns* (Reading: Addison-Wesley)