Astronomical Data Analysis Software and Systems XV
ASP Conference Series, Vol. 351, 2006
C. Gabriel, C. Arviset, D. Ponz and E. Solano, eds.

Sherpa: Goals and Design for Chandra and Beyond

S. Doe, D. Nguyen, C. Stawarz, A. Siemiginowska, D. Burke, I. Evans, J. Evans, J. McDowell, B. Refsdal

Smithsonian Astrophysical Observatory, 60 Garden Street, Cambridge, MA 02138

J. Houck, M. Nowak

MIT Kavli Institute for Astrophysics and Space Research, 77 Massachusetts Avenue, 37-287, Cambridge, MA 02139

Abstract. We describe a new, modernized design for *Sherpa*, the CIAO fitting and modeling application, planned for the CIAO4.0 release. *Sherpa* can be used for general modeling and fitting of astronomical data. It is not restricted to analyzing X-ray spectra; *Sherpa* has been used to analyze images and spectra from many X-ray missions (e.g., Chandra, XMM, ROSAT), and even from optical missions such as Hubble. *Sherpa* provides a number of different models, fit statistics and optimization methods, and is extensible with the S–Lang programming language. However, integration with S–Lang is not complete as it was a late addition to *Sherpa*; and as a monolithic application, *Sherpa* is difficult to link to other tools and programming environments. The new design calls for separating the models, fit statistics, optimization methods and science functions into several standalone modules, that can be loaded into a new *Sherpa* application, other S–Lang applications, or C/C++ programs. The design will make it straightforward for users to embed *Sherpa* modules into other languages, such as Python or Perl. The *Sherpa* application will be a new S–Lang application, providing the full capabilities of S–Lang, and which will more smoothly load and run user extensions written in S–Lang, C/C++, or Fortran. Such an environment provides a flexible way for users to add functionality to solve new data analysis problems. In this paper we discuss how the new design will better support analysis of data from Chandra and many other missions (including non-X-ray missions); and how the new design will allow us to better accommodate future requirements, such as new optimization techniques, Bayesian statistics, distributed computing with *Sherpa*, and analysis of data sets of more than two dimensions (e.g., data cubes combining spatial and spectral information).

1. Design Goals

Sherpa is the modeling and fitting application developed by our group at the Chandra X-ray Center (CXC) for the CIAO software package (Chandra Interactive Analysis of Observations). The current version of *Sherpa* already provides

access to many different models, optimization methods and statistics. It allows analysis of images and spectra, and is extensible with the S–Lang[1] programming language. As far as providing functionality goes, many initial goals have already been met.

But in time we identified new requirements. Adding new functions to meet these requirements stressed the initial design. Tools that link to the *Sherpa* library often need only a small subset of functions, and yet must link in the entire library. *Sherpa* links to the ChIPS (Chandra Imaging and Plotting System) and S–Lang parsers, and must go through contortions so that (usually) the correct command gets to the correct parser. The *Sherpa* interface to user-written models—whether written in C, Fortran or S–Lang—has become too cumbersome.

This led us to consider a new design and new implementation of *Sherpa*. The goals of the new design are to retain all the good features of the current *Sherpa*, as well as to meet new goals of better modularity, extensibility and flexibility. Our goals include providing separate data, model, optimization method, science function, and statistic libraries; improving extensibility by making the S–Lang interpreter the only application parser; and increasing flexibility by implementing the application in three distinct layers. These three layers are the base layer of the various *Sherpa* libraries, the application layer binding together data sets and models for fitting, and a UI layer that provides interfaces to functions in the application layer.

2. Base Libraries

The first part of the design has been to identify the pieces of *Sherpa* that should go into separate libraries. To date, we have designed, and partially implemented, data set, model, optimization method, and statistic libraries.

The data set library contains classes that encapsulate data sets to be fit. A DataSet object contains storage for data x- and y-values, model y-values (to be compared to the data), statistical and systematic errors, and weights.

The model library contains classes that encapsulate models to be compared to data. A model consists principally of two things: model parameters, that can be varied during a fit, and a model function that, given parameter values and an x-value, can calculate the corresponding model y-value. The library contains Model classes that encapsulate particular types of models (e.g., such as Gaussian or power-law functions). The library also contains composite model, or CompModel, classes that contain collections of Model objects, and combine their output (e.g., to model a spectrum as the sum of several Gaussian models).

The optimization method library contains the methods *Sherpa* provides. Examples of such methods are the Levenberg-Marquardt and simplex optimization methods. The library contains a base class, OptMethod, and derived classes encapsulating each of the optimization methods.

The statistic library contains various statistics that can compare data and model values, to measure how well a set of model values fit the data values. The

[1] `http://www.s-lang.org/`

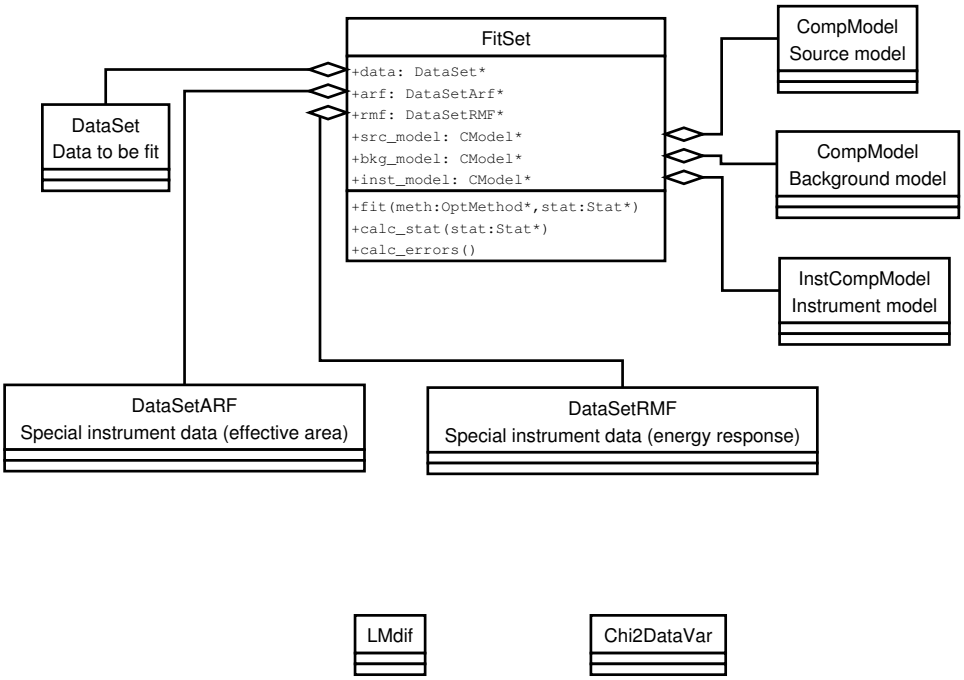

Figure 1. The FitSet class has references to a DataSet object, and various CompModel objects, which are used to calculate models to compare to the data.

library contains a base class, Stat, and derived classes that encapsulate statistics such as the Cash, χ^2, and C-statistics.

3. Application Structure and UI Layer

The *Sherpa* application needs data structures that bind together data, model, optimization method and statistics code to provide all the fitting and modeling capabilities *Sherpa* is designed to offer. Figure 1 shows the main data structure used in the application, the FitSet class. The FitSet is a container that collects together DataSet objects with the CompModel object used in a fit. In the figure, the FitSet contains data in a DataSet object, instrument data in special DataSet derived classes, and CompModel objects that model emission from the source of interest, emission from the background, and an instrument CompModel that models how the instrument affects the observation. The FitSet can also accept references to OptMethod and Stat objects for fitting. The application will contain a collection of FitSet objects (one per data set to be fit). A fit can be performed on a single FitSet, or all FitSets simultaneously.

Access to these FitSets and to functions to manipulate them is provided by S–Lang functions. But we also intend to make it possible for users to replace the S–Lang interface with interfaces to other languages (e.g., Perl, Python). Thus the structure of the application is in three layers, as shown in Figure 2.

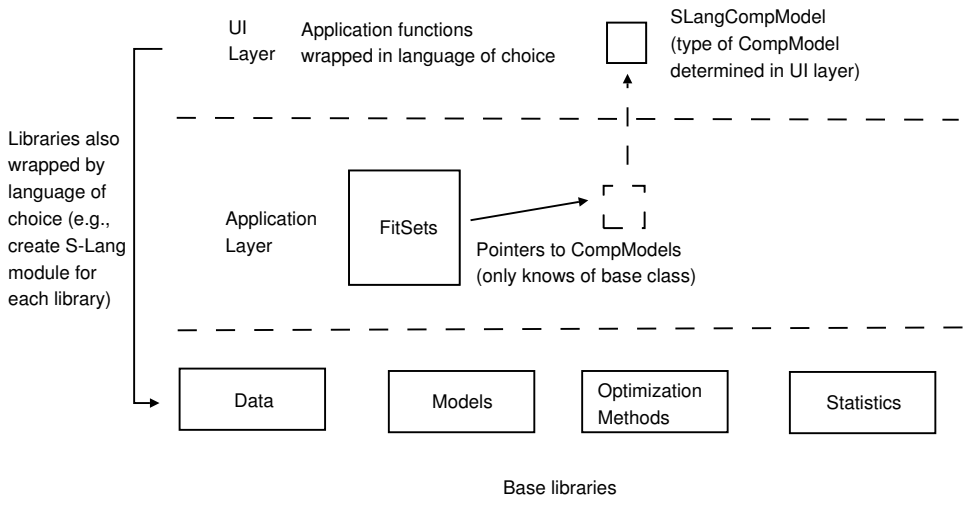

Figure 2. The *Sherpa* application will be built in three layers—a base layer of several libraries, an application layer to bind relevant objects together, and a UI layer.

The base libraries constitute the first layer. Each base library is an independent library; it can be linked to some other piece of software without having to drag along the other libraries.

The second layer is the application layer, which contains all the logic needed for the application to fit models to data (along with other science functions). FitSets containing data and models to be fit to the data exist in this layer.

The third layer is the UI layer. The goal is for the application layer to be completely agnostic when it comes to the UI. We provide the S–Lang interpreter as the command-line interface, and also as the parser to evaluate composite models. FitSets in the application layer only know that they have been assigned CompModel objects. The UI layer creates language-specific model objects (of classes derived from CompModel) and assigns them to FitSets. Users need only replace code and derived classes in the third layer to use *Sherpa* in Perl, Python or other environments. This design also leaves room for us to add a GUI to *Sherpa* at a later date.

Acknowledgments. Support for development of *Sherpa* is provided by the National Aeronautics and Space Administration through the Chandra X-ray Center, which is operated by the Smithsonian Astrophysical Observatory for and on behalf of the National Aeronautics and Space Administration contract NAS8-03060.