Interactive Analysis and Scripting in CIAO 2.0

S. Doe, M. Noble, and R. Smith

Harvard-Smithsonian Center for Astrophysics, MS 81, 60 Garden Street, Cambridge, MA 02138, Email: sdoe@head-cfa.harvard.edu

Abstract. Interpreted scripting languages are now recognized as essential components in the programmer's (and user's) tool chest, and as amply demonstrated at ADASS 1999, have infiltrated the scientific community with great effect.

In this paper we discuss the utilization of the S–Lang interpreted language within the Chandra Data Analysis System (CIAO, or Chandra Interactive Analysis of Observations). In only a few months, with substantial reuse and comparatively little manpower and code bloat, this effort has increased by an order of magnitude the analytical power and extensibility of CIAO.

We summarize our design and implementation, and show brief fitting, modeling, and visualization threads that demonstrate capabilities roughly comparable with those of commercial packages. Finally, we present a beta version of the CIAO spectroscopic analysis module, GUIDE – largely a collection of S–Lang scripts, glued with C++ enhancements to Sherpa and ChIPS – to illustrate in more depth the range of new functionality and the rapid prototyping now available in CIAO.

1. Extending CIAO Applications with S–Lang

S–Lang is an interpreted language created by John Davis of the Center for Space Research at MIT. S–Lang is a popular language, with several hundred users, and several applications that use it as an extension language. Embedding such a language into an application can enhance its flexibility and power by allowing users to more easily extend its capabilities. We have added S–Lang to CIAO, where it is currently used most heavily in ChIPS (Chandra Imaging and Plotting Software) and Sherpa (the CIAO fitting application).

2. Reasons for Choosing S–Lang

Several candidate languages were evaluated before S–Lang was chosen, among them Python, Tcl and Glish. Some of the reasons we did not choose one of these alternatives are:

- Tcl, a string based language, historically underperforms for numerical work
- Python does not support multi-dimensional arrays, so other packages (such as NumericalPython) also need to be installed

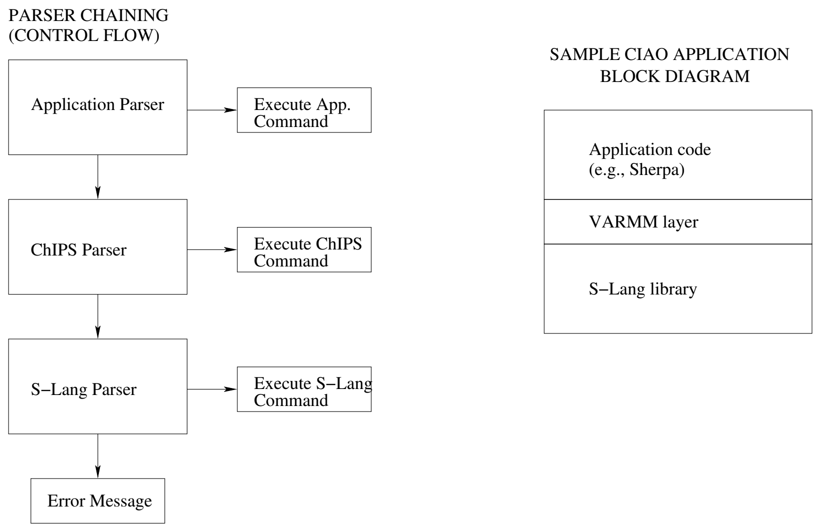

Figure 1. CIAO Application Structure. CIAO's new VARMM library is a thin layer between S–Lang and an application. The application also allows ChIPS and S–Lang commands to be parsed.

- Python is typically not used as an embedded language; it has a larger footprint, and would be more complicated to integrate with CIAO
- Glish at heart is an event dispatcher, and CIAO has already adopted XPA for event/message communication between CIAO processes; also, the Glish parser is not meant to be embedded within another parser.

What makes S–Lang attractive for CIAO are its:

- built-in multi-dimensional array handling
- well-defined grammar
- ease of installation and integration with CIAO
- lighter footprint, and fast execution
- intrinsic arithmetic and mathematical functions

3. VARMM – Interface Between S–Lang and C++

In addition to embedding S–Lang into CIAO applications, we have created a new library that acts as an interface between C++ code and S–Lang. This is the VARMM (Variables, Math and Macros) library. As shown in Figure 1, the VARMM library is a thin layer, providing a clean, simple API, and useful classes for representing data. The VARMM library provides several benefits:

- shields application code from explicit knowledge of interpreted language
- a clean, focused interface that simplifies application development
- OO abstractions that naturally model the data usage patterns
- adds S–Lang to new CIAO applications with less developer effort

The VARMM library provides two classes for accessing data stored in S–Lang variables. S–Lang variables may store scalars, arrays, or structures that encapsulate several scalars and/or arrays. The Varmm class provides access to the contents of a S–Lang variable; for uniformity, Varmm treats all S–Lang variables as arrays. A Varmm object implicitly synchronizes with its S–Lang variable, so that changes to variables in S–Lang scope are automatically reflected in the C++ scoped object when S–Lang variable data members are accessed.

The library also provides the VarmmDSet class, which is an associated array of Varmms. The VarmmDSet class can be used to represent data read from file; the data from each column of a table are stored in Varmms, and additional information about the file (e.g., the file name, the path, etc.) is stored in other VarmmDSet data members. Data from particular file types, for which we need to store additional information (images, PHA and RMF files), can be stored in objects of classes which derive from the VarmmDSet class. S–Lang variables can be accessed by an application via the Varmm and VarmmDSet public functions.

4. Other VARMM Functions

VARMM provides the programmer with additional functions, to a) create, copy and destroy Varmms and VarmmDSets from within an application; b) get a reference to the data to which a Varmm points; c) get references to Varmms and VarmmDSets given the names of the corresponding S–Lang variables; and d) pass commands to the S–Lang parser (see Figure 1).

The VARMM library also provides a number of user functions for reading data from ASCII and FITS files, and assigning the data to S-Lang variables. These functions are: `readfile, readascii, readbintab, readpha, readarf, readrmf, readimage, writeascii` (to read from and write to various file formats) and `print, printarr` (to print the contents of variables).

5. An Example

Here is a simple example, in which we use a S–Lang variable in ChIPS:

```
chips> pldata = readfile("phas.dat")
chips> print(pldata)
filename        =  phas.dat
path            =  /home/sdoe/
filter          =  NULL
ncols           =  2
nrows           =  124
col1            =  Float_Type[124]
col2            =  Float_Type[124]
chips> plot x pldata.col1 y pldata.col2
chips> pldata.col2 = log(pldata.col2 + 1.0) + 10
chips> redraw
```

A S–Lang variable called `pldata` is created, and data are read from a file named "phas.dat". The ChIPS `plot` command creates a plot, using data from `pldata`. The data in `pldata` can then be manipulated via S–Lang commands at the ChIPS command line. When the plot is drawn again, the updated y-values are immediately apparent.

In addition, the user can customize ChIPS and Sherpa with user–defined scripts. For example, the user could ignore points below a certain y-value by creating the following script:

```
define ignoreLowPoints(data, floor) {
   data.col1 = where(data.col2 > floor);
   data.col2 = where(data.col2 > floor);
   return data;
}
```

The script could then be used in ChIPS:

```
chips> evalfile("ignoreLowPoints.sl")
chips> ignoreLowCounts(pldata, 10)
chips> redraw
```

The new plot would contain only those points where the y-value exceeded 10.

6. GUIDE

Embedding S–Lang in ChIPS and Sherpa not only opens up the data for the user, but also allows the user to create and use scripts to further extend CIAO capabilities. Such scripts may consist of Sherpa, ChIPS and S–Lang commands combined.

For CIAO 2.0, we present a package of such scripts, to facilitate analysis of Chandra high–resolution spectra. This package is called GUIDE (Grating User Interactive Data Extension) and is a modular extension to Sherpa.

GUIDE extends Sherpa in several ways. First, we provide a new file format for storing the state of a Sherpa session. More importantly, GUIDE provides many functions for analysis of high–resolution Chandra spectra. These include functions to identify emission features; to provide access to collisional plasma models, such as the APEC or Raymond & Smith models (to get predicted line emissivities); and to use the modeled emissivities to fit a differential emission measure model to the line fluxes.

While GUIDE is beta software, we already see that embedding S–Lang in ChIPS and Sherpa has enabled us to provide tools for spectroscopic analysis, to do so much more rapidly than we would have otherwise, and to involve scientists more directly in their creation. We believe this approach will be useful for future CIAO releases and for users who develop their own S–Lang packages for use with CIAO software.

Acknowledgments. This project is supported by the Chandra X-ray Center under NASA contract NAS8-39073.

References

S–Lang Web site: http://space.mit.edu/~davis/slang/

Python: http://www.python.org/

Glish: http://aips2.nrao.edu/docs/glish/glish.html