Astronomical Data Analysis Software and Systems VIII
ASP Conference Series, Vol. 172, 1999
D. M. Mehringer, R. L. Plante, and D. A. Roberts, eds.

A Pipeline Tool for CCD Image Processing

Jon F. Bell, Peter J. Young, William H. Roberts, Kim M. Sebo

Mount Stromlo and Siding Spring Observatories, Private Bag, Weston Creek, ACT, 2611, Australia

Abstract. MSSSO is part of a collaboration developing a wide field imaging CCD mosaic (WFI). As part of this project, we have developed a GUI based pipeline tool that is an integrated part of MSSSO's CICADA data acquisition environment and processes CCD FITS images as they are acquired. The tool is also designed to run as a stand alone program to process previously acquired data. IRAF tasks are used as the central engine, including the new NOAO mscred package for processing multi-extension FITS files. The STScI OPUS pipeline environment may be used to manage data and process scheduling. The Motif GUI was developed using SUN Visual Workshop. C++ classes were written to facilitate launching of IRAF and OPUS tasks. While this first version implements calibration processing up to and including flat field corrections, there is scope to extend it to other processing.

1. Context

This pipeline tool[1] was originally developed for the Wide Field Imager (WFI)[2] being developed jointly by MSSSO, AAO and the University of Melbourne, but it can be used to process other CCD image data. It has been developed in a general manner so that other processing tasks contained in IRAF[3] can be easily added. One might envisage adding facilities for calibration processing of 1 D spectra for example. While it has been developed to run in conjunction with CICADA[1] (Configurable Instrument Control and Data Acquisition), software developed at MSSSO, it may also be used offline in a stand alone mode. Figure 1 shows how this tool fits into the overall process.

2. Architecture

To build a pipeline we needed 3 main components: A processing engine, a data managing and scheduling tool, and a graphical user interface. All of these components had to mesh neatly with the existing CICADA environment, shown

[1] `http://msowww.anu.edu.au/computing/cicada`

[2] `http://msowww.anu.edu.au/observing/wfi/`

[3] `http://iraf.noao.edu/`

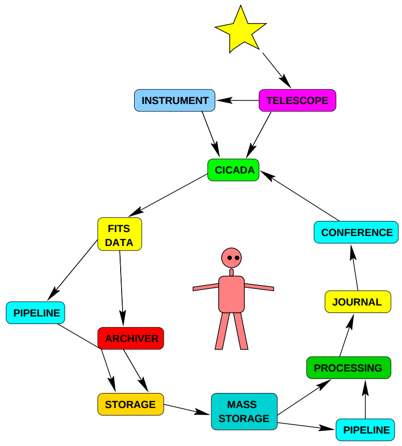

Figure 1. This pipeline tool in context.

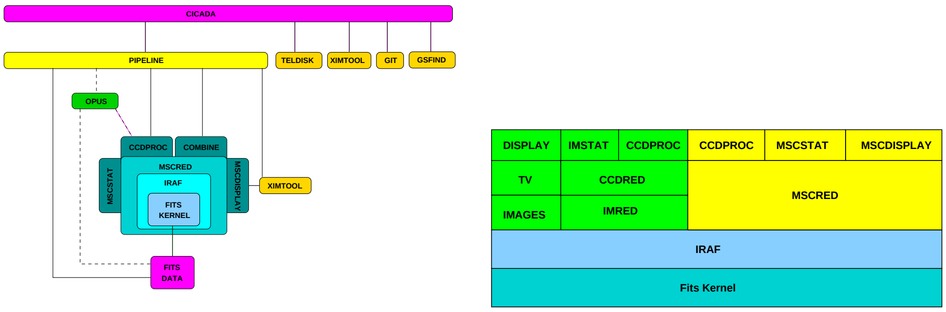

Figure 2. Architecture of Pipeline and Cicada environment (*left*). Inheritance structure of IRAF classes (*right*).

at the top of Fig. 2. They also had to be readily and freely available to all groups involved with the development of WFI. We chose IRAF (see §3.) for the central engine and built a Motif GUI (examples below) with a similar look and feel to CICADA. For a data scheduling and managing tool we experimented with both OPUS and our own code and at present either may be used (as indicated by the dashed lines in Fig. 2). While OPUS performed well, we have only made use of a small subset of its capabilities. Therefore future versions will be based on our own simple scheduler, reducing the complexity of this pipeline tool.

3. Central Engine – IRAF

A key requirement of the processing engine in this pipeline was the ability to process multi-extension FITS (MEF) files. Our investigations revealed that the best means of doing this was to use the recently developed IRAF package mscred. It provides multi-extension analogs of most of the tasks found in the ccdred package of IRAF, as well as a number of mosaic specific tasks. While IRAF tasks can be used in an automated fashion by running the package executables directly, we found it much simpler and more robust to use scripts to run the

IRAF cl directly. The structure of IRAF lends itself very nicely to a multiply inherited OO class structure, in the sense that a task or sub package inherits all the properties of its parent. In order to implement the use of IRAF in our pipeline, we developed a multiply inherited C++ class structure as shown in Fig. 2. This has been done in a very general way, so other tasks and packages can easily be added. As shown in Fig. 2, there is a base class for the FITS kernel, which is inherited by the IRAF class. Packages such as mscred then inherit the IRAF base class, thereby also inheriting the FITS kernel base class. In the present implementation the functions in these classes receive all the necessary information from the GUI or configuration files and write out scripts containing all the necessary details to run each task. Getting information back from IRAF can be another tricky issue when it is run in an automated way. We have approached this issue, by turning the IRAF logging facilities on and using event handlers and timeouts to return information from the log files. When Open IRAF is available it would be possible to use that to directly call IRAF tasks, rather than the method noted above.

4. Typical Processing Steps

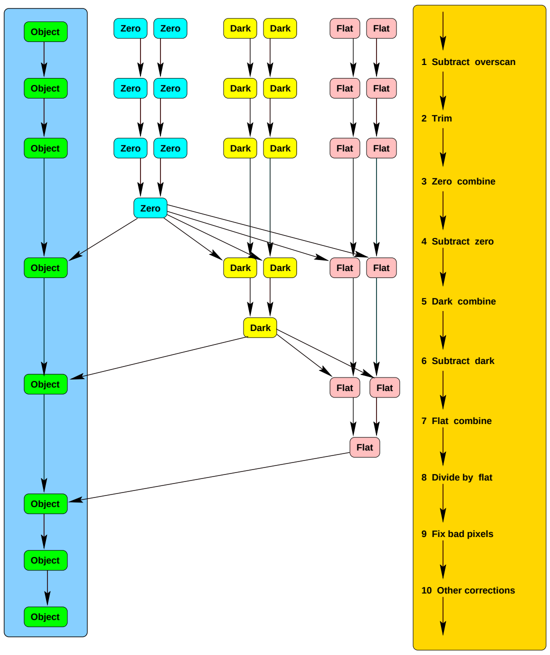

Figure 3. Processing steps presently implemented.

The processing sequence that is presently implemented is shown in Fig. 3, starting at the top and flowing down. The right most box describes the steps involved. Following the successful processing of each image, the image is displayed, along with image statistics, so the user can quickly verify that the processing is proceeding as intended. At present, due to the complicated nature of preparing flat field images for WFI and other wide field CCD mosaics, it is assumed that the calibration images have been pre-prepared (or obtained from a library) and the path on the right is followed for images containing science data. When the formation of flat fields becomes a more reliable process, it will be possible to automate pipelining of the steps shown above as IRAF already supports this.

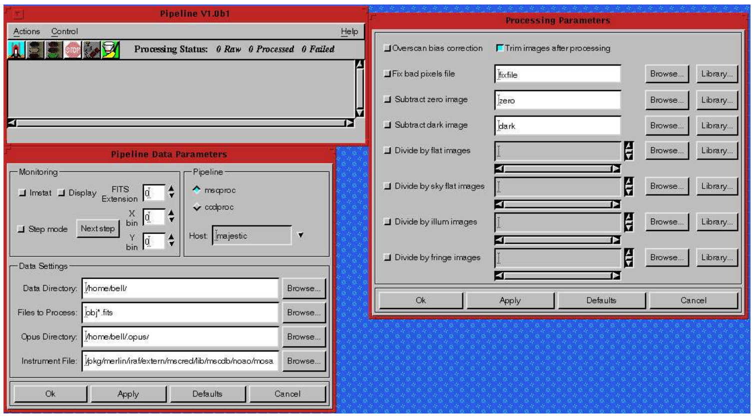

Figure 4. Snapshots of the GUIs.

5. Development Tools and Platform

The GUI was developed using the SUN Workshop Visual Interface Builder V2.0[4], generating C++ code. Extensive use was made of the Tools.h++ 4.0.1[5] class library as well as other locally developed libraries for process communication, exception handling and automatic setting up of OPUS[6]. All modules were built and tested using the Sparc C++ compiler on Solaris 2.6. Versions of third party software used were: IRAF: V2.11.1, MSCRED: V1.1 and OPUS: V1.0. The system resources required by this pipeline are dominated by the data.

Acknowledgments. We gratefully acknowledge very helpful advice and support from Frank Valdes and other IRAF developers at NOAO and Jim Rose and other OPUS developers at STSCI. The WFI software working group also contributed many useful ideas and suggestions.

---

[4] `http://www.sun.com/workshop/visual/`

[5] `http://www.inmark.com/products/`

[6] `http://www.stsci.edu/software/OPUS/`