Astronomical Data Analysis Software and Systems IV
ASP Conference Series, Vol. 77, 1995
R. A. Shaw, H. E. Payne, and J. J. E. Hayes, eds.

The ASC Pipeline: Concept to Prototype

A. Mistry

TRW/AXAF Science Center, 60 Garden St., Cambridge, MA 02138

David Plummer, Robert Zacher

Smithsonian Astrophysical Observatory/AXAF Science Center, 60 Garden St., Cambridge, MA 02138

Abstract. Discusses the role of a pipeline in the *AXAF* Science Center Data System. Due to the complexity of the application, a prototyping effort was initiated to evaluate the risks involved with portability, open architecture, error handling/recovery, and distributed processing. Processes in the pipeline are initiated and monitored by a process control application developed using Perl. Currently, the process control application can attach IRAF, IDL, shell scripts, and various third party executables to create a number of data processing flows. An example process flow involves the acquisition of raw telemetry data from a data simulator to an image file. An error logging library was also developed to form the foundation for handling both minor and fatal errors.

1. Design Concept

The purpose of the *AXAF* Science Center Data System (ASC DS) is to provide the science community with useful *AXAF* data and the tools to process the data. The role of the pipeline is to provide the framework for creating data processing flows which produce the standard data products, and to provide an easy to use mechanism that enables scientists to create their own customized processing flows. Several basic items need to be incorporated in order to meet the needs of the ASC DS and the science community: the processing flows must be *programmable* to meet the needs of the user; the system must be *open* to external applications, either commercial or custom; the system must be *portable* to all platforms supported by the ASC DS; the system must be *reliable*, in that it must be fault tolerant and preserve data integrity; and the system must have *good performance*, in that it must perform its task in a reasonable amount of time. Finally, the system must be *easy to use*, in that it must provide a clear interface for the novice user as well as short-cuts for the experienced user.

2. Prototype Activity

The purpose of the prototyping activity was to determine if the design concept can be achieved before the project enters full scale design and development.

After careful consideration of the goals that were set forth for the pipeline, a set of risk areas were defined. Those areas are: portability, error handling and recovery, open architecture, and distributed processing.

The prototype activity for the pipeline started in March and was completed by the end of July. During that time, a Process Control application was developed to allow us to assess the risk factors. The following section discusses how each of the risk areas were resolved.

2.1. Portability

A study of the portability issue was performed before any coding was done. From that research, two pieces of information were gathered: portability is simply the degree to which an application can be compiled and executed on a variety of different hardware platforms. As well, the language used for developing an application is irrelevant in determining its portability; provided that the application was developed using one of the popular development languages, and that no system-dependent routines were used.

The development language for the pipeline has not been selected yet. However, we are leaning towards a POSIX-complient system. POSIX is a group of standards focusing on hardware and software portability (Lewine 1994). The script language Perl was selected for the prototype because of its inherent portability and file/string manipulation capabilities (Wall & Schwartz 1992).

2.2. Error Handling and Recovery

The key to running the pipeline effectively and reliably will be its ability to detect and recover from errors. Through the prototyping effort, we learned that it is possible to trap the errors of an application regardless of what language it was developed under. The methodology is simple. When an application creates a child process, the parent will receive all error conditions that the child can not handle through `stderr` (Stevens 1992). Also, it is possible for the parent to trap the error condition before it affects the child. This scheme works since the Process Control is the parent of all processes that run in a pipeline. The recovery process is then simply a matter of querying the user for an action, or executing the proper error handling routine to correct the situation.

2.3. Open Architecture

The only way the Process Control application can be useful outside of the ASC environment and be flexible enough to adapt to changing technology is for the application to be open. The definition of an open system is:

Open System A system in which the specifications are made public (Lewine 1994).

Since users only need to know how to interface with the Process Control, and not how it works, we would like to add to that definition as follows:

Open System A system in which the *interface* specifications are made public.

An interface standard has not yet been developed, but work is in progress. The interface will be kept neat and simple. Once an interface standard has been published, all applications which follow that standard will operate with the Process Control, with no surprises.

2.4. Distributed Processing

The performance of a pipeline would be greatly increased if the processes were distributed over a number of machines (Jain 1991). The methodology is once again straightforward, in that a `daemon` process will be running on all machines that have been allocated for the Process Control. The Process Control will then assign a task in the pipeline to the available machines for execution. The data transfer from module to module will occur through sockets. In some cases the tasks themselves may be further distributed, but that responsibility will fall on the tasks themselves.

3. Conclusion

The purpose of any prototyping effort is to gather knowledge, to assess the risk factors, and to determine if a proposed design is possible. A fair amount of knowledge has been gained regarding open systems, portability, and distributed processing, through research of available literature and *trial and error*. Our knowledge base has grown to the point where we believe we can create a solid design and produce a reliable and easy to use product.

All of the above mentioned risk factors were assessed in the prototype. None of these factors seems to pose unreasonable risks. We learned several interesting lessons, which remain as concerns for the future:

Running IRAF in the Command Line mode. Pipes or sockets cannot be used in this case. If an IRAF task crashes, an error is reported via STDOUT, and no error code is returned to the shell. This makes it difficult for the process control to recover from a fatal error.

IDL. The number of IDL analysis tasks is limited to the number of site licenses. However remote procedure calls should solve the problem.

Perl. Perl lacks the ability to create complex data structures, which could make development of a complex system more challenging.

In summary the prototype proved that the original design concept is feasible.

References

Stevens, W. R. 1992, Advanced Programming in the UNIX Environment (Reading, Addison-Wesley)

Wall, L., Schwartz, R. 1992, Programming Perl (Sebastopol, O'Reilly & Associates)

Lewine, D. 1994, POSIX Programmer's Guide (Sebastopol, O'Reilly & Associates)

Jain, R. 1991, The Art of Computer Systems Performance Analysis (New York, Wiley)