The Chandra automated processing system: challenges, design enhancements, and lessons learned

David Plummer[a], John Grier[a] , and Sreelatha Masters[a]

[a]Smithsonian Astrophysical Observatory, 60 Garden St., MS-81, Cambridge, MA 02138

ABSTRACT

Chandra standard data processing involves hundreds of different types of data products and pipelines. Pipelines are initiated by different types of events or notifications and may depend upon many other pipelines for input data. The Chandra automated processing system (AP) was designed to handle the various notifications and orchestrate the pipeline processing. Certain data sets may require "special" handling that deviates slightly from the standard processing thread. Also, bulk reprocessing of data often involves new processing requirements. Most recently, a new type of processing to produce source catalogs has introduced requirements not anticipated by the original AP design. Managing these complex dependencies and evolving processing requirements in an efficient, flexible, and automated fashion presents many challenges. This paper describes the most significant of these challenges, the AP design changes required to address these issues and the lessons learned along the way.

Keywords: data processing, automated processing, pipelines, system architecture

1. INTRODUCTION

The Chandra X-ray Observatory was launched in the summer of 1999 to observe the high energy regions of the universe in the X-ray spectrum. It is the most sophisticated and sensitive X-ray observatory built to date.[1] The Chandra X-Ray Center Data System (CXCDS)[2] develops and maintains the software that is used by operations[3] to process, archive and distribute Chandra data. Sophisticated hardware requires a sophisticated processing system to take the data from raw spacecraft telemetry to archived data products useful for astronomical research. To accomplish this task in a timely manner the processing system must be automated, yet still allow operators the ability to intervene at the appropriate stage of processing to recover from errors or handle "anomalous" situations.

Like any complex system, the Chandra standard data processing system (SDP) is divided into layers. The low level details of scientific processing are captured in stand-alone programs that make up the data analysis and science operations toolkits. A Chandra pipeline is a sequence of tools assembled to perform a standard reduction and analysis of data to produce a set of archived data products. The Chandra automated processing system (AP)[4] is the component of the CXCDS that provides an infrastructure layer above the pipelines. AP initiates pipeline processing at the appropriate time, provides input data and run-time arguments, and sends output products to the archive.

The current design of the AP system differs from the initial design presented at the NASA "Critical Design Review" in some very significant ways. The AP design has "evolved" in response to a few key milestones. The most significant milestone was the end-to-end ground calibration and test at the Marshall Space Flight Center (MSFC). The calibration data products were different from flight products in many ways but were still processed by pipelines and archived so that it provided a good test of the processing system under realistic operational conditions. The end-to-end test resulted in some significant AP design changes that were implemented before launch. After launch, the need to re-process data and handle "special" situations presented challenges requiring another set of AP enhancements. The latest set of new processing requirements is associated with the production of a Chandra source catalog. The catalog processing is very CPU intensive and requires a new hardware platform to handle all the computations.

The following sections will focus on the evolution of the AP design in response to the various challenges encountered during the life of the Chandra mission. The AP design changes embody a set of "lessons learned". The components that were dropped illustrate approaches that did not work well. The components that remain or were added will (hopefully) continue to illustrate what does work. We will also high-light a few future enhancements that illustrate what could still be improved.

2. CHARACTERISTICS OF CHANDRA DATA AND PROCESSING

The very nature of the Chandra data imposes some challenging requirements on the processing system. Chandra carries four science instruments, two X-Ray detectors and two grating arrays which can be moved into the path of the X-Rays to obtain high resolution energy spectra. X-Ray data is collected photon by photon as opposed to integrating over time to produce an image. Each X-Ray photon is tagged with a time, energy and position on the detector. This section provides an overview of the Chandra data emphasizing those aspects that impact the design of the processing system.

2.1. Telemetry

Chandra telemetry is transmitted to the ground in a format defined by NASA (HOSC-standard format, as defined in AMO-2050). A telemetry record is called a minor frame and represents a time span of 0.25625 seconds. Each subsystem of the spacecraft inserts information in the telemetry stream in a manner that maximizes the amount of data that can be transmitted via the limited bandwidth.

Telemetry is received in the form of a file of arbitrary length (usually containing about 8 hours of data). Each file overlaps the previous file to ensure that no data is lost. Occasionally, data transmission from the spacecraft encounters a problem and the telemetry files contain gaps. There are 24 bits devoted to the minor frame counter and so every 50 days the counter will roll-over back to zero. The minor frame counter can also be reset to zero during a safe-mode event as the spacecraft recovers from an anomalous situation. Telemetry Data Receipt must keep track of the records already processed (accounting for gaps, roll-overs and resets) and trim off the overlapping data so no duplicate records are sent to Telemetry Processing.

2.1.1. Subsystems (components of telemetry)

The Chandra telemetry is composed of the following subsystems:

- ACIS - Science Instrument - X-Ray CCD Imaging Spectrometer
- HRC - Science Instrument - High Resolution X-Ray Camera
- Engineering Data - spacecraft temperatures, pressures, voltages, etc.
- Science Instrument Module (SIM) - positions X-Ray detectors in the focal plane
- EPHIN - Background detector (Electron Proton Helium INstrument)
- PCAD - Pointing Control and Attitude Determination

The telemetry processors read a set of template files that define exactly where and how often each data element is placed in the telemetry stream.

2.1.2. Atomic units

Data from each subsystem is sampled at different rates and can be inserted into different parts of the telemetry stream. The "atomic unit" of a given subsystem is the smallest number of minor frames required to obtain the full data set for a given sampling period. For example, the engineering data in one minor frame is complete so the atomic unit of engineering data is 1 minor frame (equals .25625 seconds). However, the SIM requires 4096 minor frames to complete 1 data set and therefore has an atomic unit of 4096 minor frames (equals 1049.6 seconds).

As each minor frame is handled by the telemetry processors, data is added to the appropriate subsystem buffer. An output record can be written out to a file only when the buffer is complete. With different subsystem atomic units, there is always some data left in the buffer when the telemetry processors come down. Therefore, the telemetry processors must keep track of the records written for each subsystem in a set of persistence files so that they can start up where they left off when coming back up.

2.1.3. ACIS science runs

The ACIS instrument operates in a way that is fundamentally different from the rest of the spacecraft. ACIS data come in various types of "packets" and are packaged into a "science run" that can be arbitrarily long. Each science run begins with a "start science" packet and ends with a "stop science" packet. Processing data at the end of a science run depends on information transmitted at the beginning of a science run. So telemetry processing must maintain information regarding the state of the ACIS instrument.

The requirement to handle different subsystem atomic units and maintain ACIS state information means that telemetry processing can not jump around from file to file. The standard processing thread must be fed a continuous stream of telemetry, in time order, and with no duplicate records. We can handle the small gaps in the telemetry stream that occur on occasion. However, if we tried to skip over a day or more of telemetry (for some operational reason) we could combine two different ACIS science runs and pollute the processed data by using the wrong ACIS state information.

2.2. Ancillary Data

Ancillary data consists of information necessary for data processing that is not contained in the telemetry stream. The three main sources of ancillary data are listed below:

- Observation data - Observation information is ingested into the archive as part of the Mission Planning process. Observation parameters are then extracted from the archive as part of the SDP thread.
- Ephemerides - Ephemeris data consist of position data and velocity vectors for the sun, moon, planets and the spacecraft.
- Clock Correlations - Spacecraft Clock Correlation Files (SCCF) allow the processing system to accurately time-tag Chandra data. The SCCFs map the spacecraft counters (minor frames) to absolute time.

2.3. Processing Levels and Data Products

The standard data processing thread (SDP) and archived data products are divided into different levels. In general, higher level processing takes the lower level data products as inputs. These are the Chandra processing levels[5]:

- Level 0 : Telemetry decommutation and ancillary data processing. Data is converted into useful data products (Chandra uses the FITS format).
- Level 0.5 : Determines the start and stop time of Observation Intervals (OBIs); data are cleaned or formatted as preparation for Level 1.

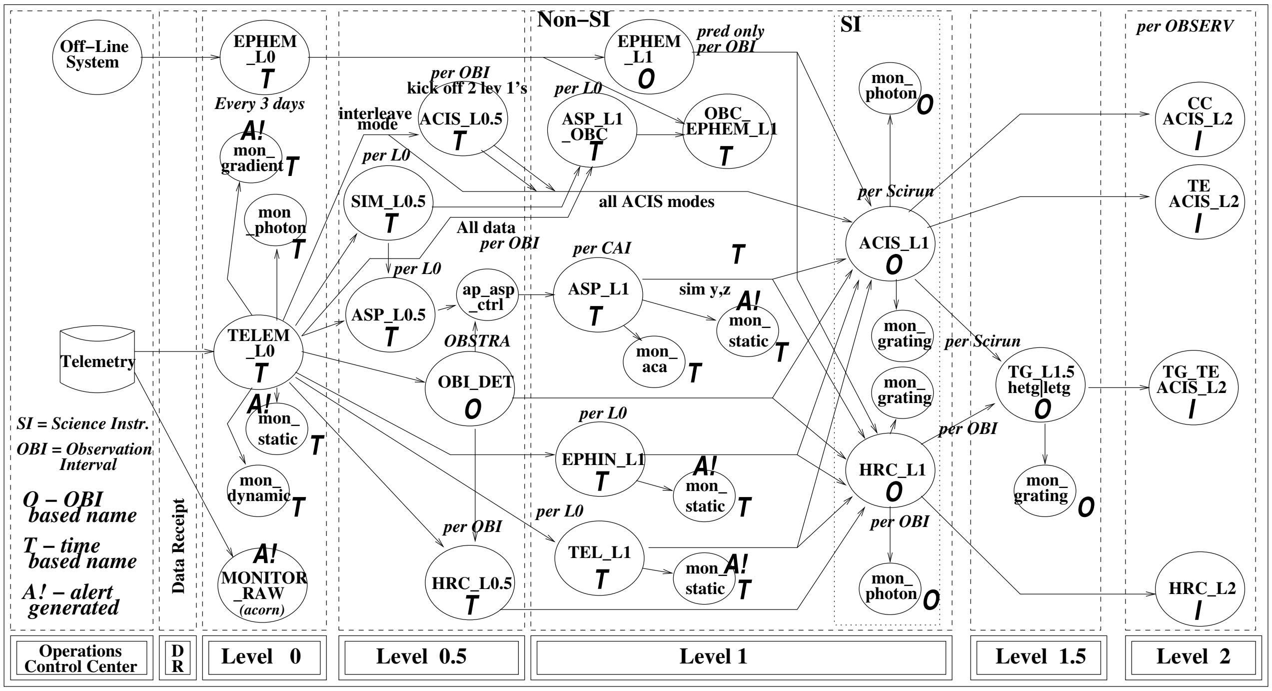

Figure 1. Chandra Standard Processing Data Flow Diagram.

- Level 1 : Data processed by Observation Interval (OBI) – calibrations and coordinates applied to science data; determination of Good Time Intervals (GTIs) by analyzing monitoring systems and comparing them to acceptable limits.
- Level 1.5 : Data processing for observations with gratings. Determine grating coordinates and apply calibrations.
- Level 2 : Data are processed by Observation, possibly combining multiple OBIs. Data from bad GTIs are screened out. Sources are detected and other basic science analysis tasks are performed.
- Level 3 : Data processed by source, possibly combining multiple observations to generate a Chandra source catalog (still in development).

2.4. Pipelines

Figure 1 represents the flow of data through the series of pipelines that are run during standard Chandra data processing. Each circle is a pipeline (or class of pipeline) connected to other pipelines via input and output data products. The TELEM_L0 pipelines produce data products that correspond to the various subsystems of the spacecraft listed above in section 2.1.1. Data from different subsystems will follow different paths through the system. Paths diverge as a given data product is used as input to many other pipelines. Paths converge at a pipeline that requires the output of many previous pipelines. Some pipelines are run on arbitrary time boundaries (as data become available) and others are run with a start and stop time corresponding to a particular observation interval (as determined by the Level 0.5 OBI_DET pipeline). Some pipelines monitor the health and safety of the spacecraft and run on a given data product. A diagram containing all the processing details would be much more complex because each circle actually represents a related set of pipelines. For example, the ACIS_L1 pipeline must be run in 27 different ways to accommodate the different modes of the ACIS detector. Figure 1 illustrates the complex data dependencies between pipelines. Managing this complexity is one of the greatest design challenges of the processing system.

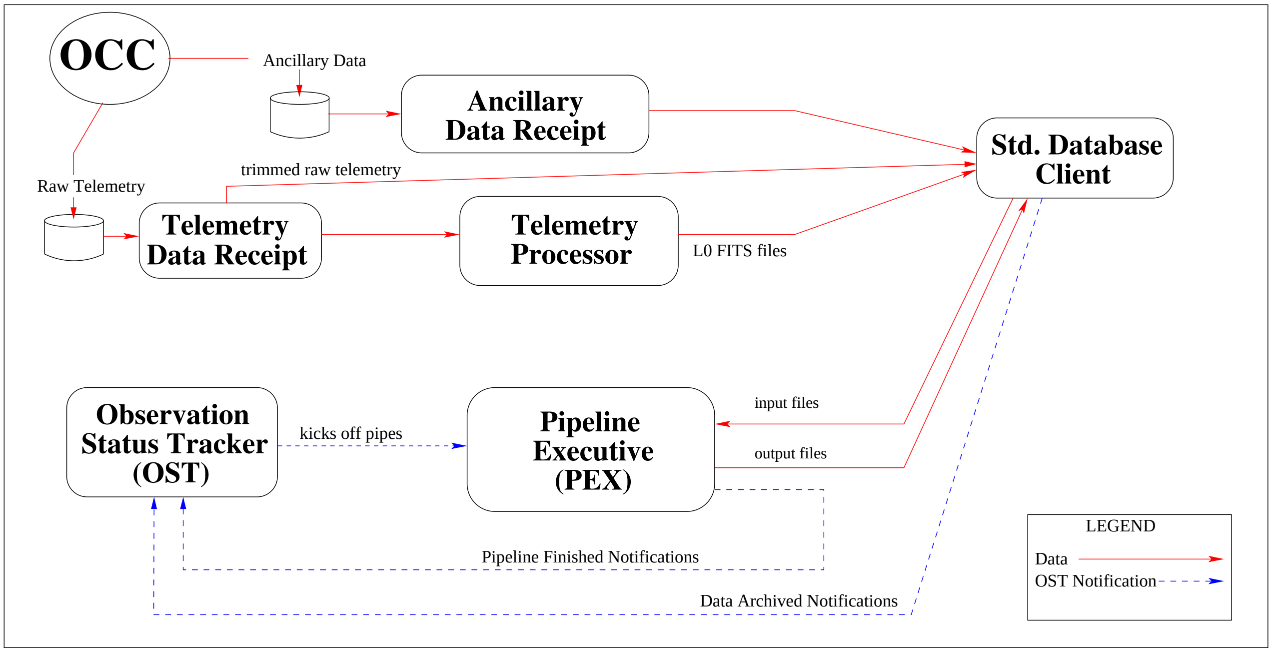

Figure 2. Chandra Automated Processing (initial design)

3. INITIAL AP DESIGN

The design of the AP system at the NASA Critical Design Review (CDR) was based on the experience of many engineers who had worked on previous X-Ray telescope missions. Many "lessons learned" had already been incorporated into the AP design. Figure 2 illustrates the "baseline" design accepted at CDR.

Processing is initiated when data is received from the Operations Control Center (OCC). The AP modules that handle the data are described in the following sections.

3.1. Data Receipt

Data Receipt polls an input directory and picks up new files as they are sent by the OCC. Ancillary Data Receipt simply sends data to the archive. Telemetry Data Receipt checks minor frame counters, trims off duplicated data and streams the data over a socket to the telemetry processors.

3.2. Telemetry Processing

Telemetry Processing decommutates and scales the raw data and produces level 0 FITS files for the various subsystem components. It also notes changes in the observation ID tag and keeps track of telemetry gaps.

3.3. Standard Database Client

The Chandra databases support client access via a command line prompt and a C++ API library. The database client implements all the security measures (like usernames and passwords) required to protect proprietary data. Data can be browsed, retrieved or ingested by the client.

3.4. Observation Status Tracker

The Observation Status Tracker (OST) keeps track of data segments received and pipeline processing results. When the appropriate inputs are received it sends a notification message to the Pipeline Executive to execute a pipeline.

Figure 3. The CXCDS Pipeline Processing Infrastructure

3.5. Pipeline Executive

The Pipeline Executive (PEX) keeps track of the list of pipelines to be executed. It retrieves pipeline inputs and calibration files, runs the pipelines by invoking the Pipeline Controller (Pctr), and ingests output data products.

3.6. Pipeline Processing Infrastructure

The pipeline processing infrastructure allows the general automated processing system to remain insulated from the scientific details of processing Chandra data. In addition to running in the AP system, pipelines can be run by developers and scientists in an analysis or test environment. Figure 3 illustrates the three components that make up the pipeline processing infrastructure.

3.6.1. Pipeline profiles

A CXCDS pipeline is defined by an ASCII profile template that contains a list of tools to run. Tool arguments can contain variables that are given actual values at run time.

3.6.2. Pbuilder

When a pipeline is ready to run, a pipeline run-time profile is generated by the profile builder tool, pbuilder. It replaces the argument variables with the actual run-time parameters (e.g., input/output directory, file name root, etc.).

3.6.3. Pipeline controller

The run-time profile is executed by the Pipeline Controller (pctr). The pipeline profiles and pctr support conditional execution of tools and branching and converging of threads. The Pipeline Controller produces a standard log-file output containing the profile, list of run-time tools, arguments, exit status, parameter files and run-time output.

4. GROUND CALIBRATION - LESSONS LEARNED

An end-to-end test and calibration of the Chandra instruments was performed before the components were integrated into the spacecraft. This was a very useful exercise for the CXCDS at many levels. We were able to exercise operational procedures and processing software under conditions that closely matched flight operations.

4.1. XRCF Overview

Ground calibration of Chandra was conducted at the Marshall Space Flight Center (MSFC), X-Ray Calibration Facility (XRCF). All Chandra subsystem were aligned in the XRCF and the chamber was pumped down to vacuum. The instruments were exposed to X-Rays and data was received and processed by the CXCDS under simulated flight conditions. In addition to the scientific calibrations and experience gained at XRCF, the CXCDS learned some valuable data processing lessons from the challenges and issues we encountered.

4.2. Module Coupling

We discovered that the modules in the AP system were too tightly coupled. For example, Data Receipt could not process the next data file until the previous file had been ingested into the archive. Archive ingest involves opening each data file, extracting information and populating database tables with meta-data associated with each file. At one point during our testing a certain quantity was sampled at ten times the rate we anticipated and produced 10 times the files we expected. The whole processing system was backed up because one data type was taking too long to ingest.

We encountered other situations where an unanticipated event would cause an error or slow-down in a particular module. We needed a way to allow modules to operate more independently so they could fall behind and catch up (or recover) without slowing down the rest of the system.

4.3. Inter-process Communication

The main method of communication between AP modules at XRCF utilized UNIX semaphores and message queues. On occasion we would encounter a problem because the message queue had reached its maximum size. These communication methods persist through hardware shutdowns, but do not easily allow for swapping out a machine due to hardware issues.

We needed a more flexible means of module communication that did not severely restrict the size of the messages or the queue and would still persist through software shutdowns and errors.

4.4. Managing Processing Threads

As with any large project, the first attempt to test and integrate a complex system will encounter many unforeseen issues. Our AP infrastructure was not easily adaptable to new processing requirements and we were often modifying and rewriting wrapper scripts to keep up with the evolving processing requirements.

Mapping a processing thread to a wrapper script did not work well at all. It became clear that we needed an infrastructure that would remain stable while the details of processing were allowed to change.

5. PRE-LAUNCH AP DESIGN ENHANCEMENTS

We were able to fold the lessons learned from XRCF back into the AP design before launch. Figure 4 shows the design of AP currently running in Chandra standard data processing mode. This section describes how the XRCF experience impacted the AP design.

5.1. Decoupling Modules

One of the most significant changes to the AP design was to insert new modules as “buffers” between critical modules. The “buffer” modules allowed the critical modules to operate more independently and remain insulated from operational issues in the other modules.

5.1.1. Archive cache

An archive cache was introduced to allow ingest to occur independent of the processing. The simple database client was replaced by a more sophisticated module (darch) that accesses both the archive and the archive cache. Data are now sent to cache and queued up for ingest allowing other modules to access the data while it is awaiting ingest into the archive. In addition to avoiding backups, this procedure allows us more flexibility in managing the load on the archive servers (ingesting at night when the retrieval load is low, for example). The archive cache also increases efficiency by keeping frequently accessed data online and immediately available to multiple processes.

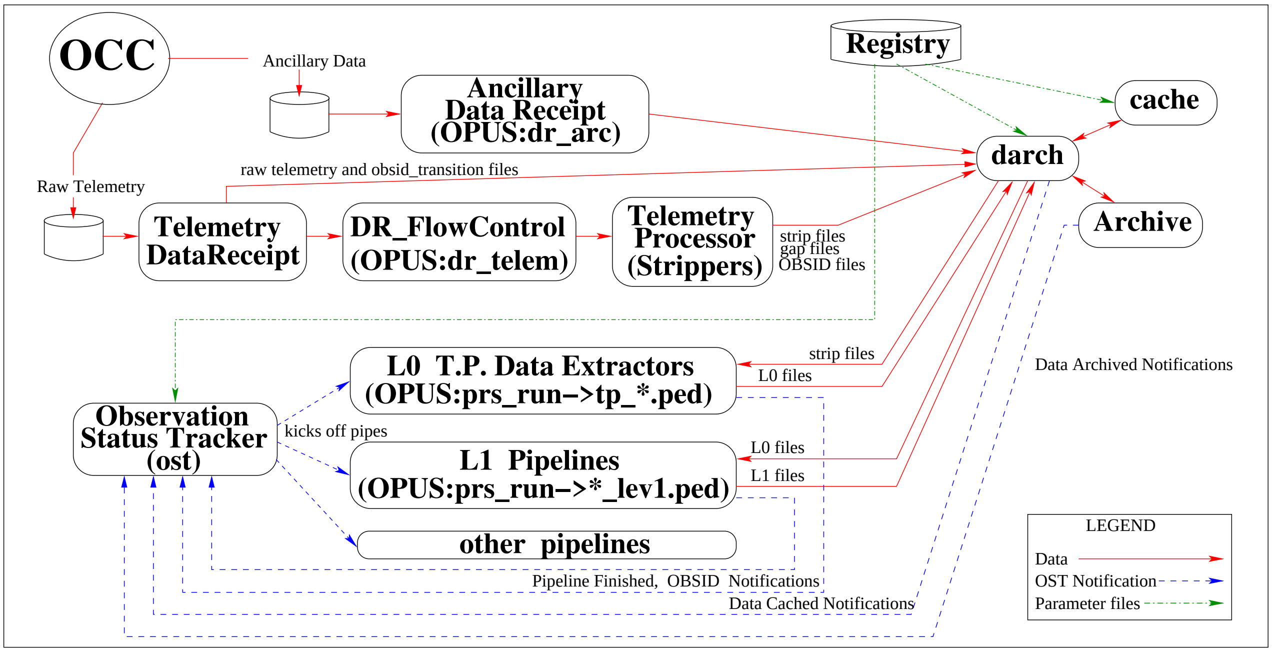

Figure 4. Chandra Automated Processing Design (SDP mode)

5.1.2. Telemetry strippers and extractors

The telemetry processing was split into two stages. The first stage, stripping, simply extracts the subsystem data out of telemetry and produces a raw "strip" file. No decommutation or scaling occurs, but the strippers ensure that all the data necessary for level 0 processing is present in each strip file. This is a fairly simple process and only depends on the format of the telemetry. With strip files in our archive it is often unnecessary to go back to raw telemetry files to re-process.

The second stage, extracting, decommutates and scales the data. The extractors run as pipelines and take one strip file as input and produce one or many level 0 FITS files.

Dividing up the telemetry processing system allows for faster processing. Raw telemetry files must go through DataReceipt and telemetry stripping in a serial fashion. However, an extractor can be run as soon as a strip file is produced. Therefore, the extractors can be run in parallel as pipelines on many machines, producing level 0 FITS files much faster than the XRCF system.

5.1.3. Data receipt flow control

DR_FlowControl allows files to be processed by Data Receipt and queue up for telemetry processing. Raw files will not back up in Data Receipt if there is an issue upstream. DR_FlowControl sends raw files to the telemetry strippers one at a time and is used as an entry point for error recovery or reprocessing.

5.2. File Based Module Communication

To achieve a more flexible means of module communication we adopted a file based system of notifications and queues. We had to handle our own locking mechanism but ended up with a more robust means of communication capable of storing thousands of messages if necessary.

5.2.1. File queue

The file queue is mainly used for archive/cache access. Requests to retrieve or ingest data are placed on a file queue. The AP data archive server (darch) takes requests off the queue and handles them, accessing the archive cache if appropriate. The number of data archive servers can be adjusted to maximize efficiency.

5.2.2. Notification files

The other AP modules communicate via notification files. Each AP module polls its own notification directory for messages. The notification infrastructure provides services to ensure that notification files from different processes do not collide and that notification files are complete before they are picked up by the receiving module.

5.3. Flexible Configuration

At XRCF we saw how quickly a set of wrapper scripts becomes difficult to maintain. For the flight system, the pipeline processing is totally configured with a processing registry. The AP system is now template driven and the infrastructure can remain stable while the registry is easily modified.

5.3.1. Processing registry

The processing registry provides a flexible way to define data products, processing pipelines, pipeline kickoff criteria and data dependencies between pipelines.

We first register all the Chandra input and output data products. We can then capture the processing requirements and inter-dependencies by registering all the pipelines. Data products are registered with a File_ID, file name convention (using regular expressions), method for extracting start/stop times, and archive ingest keywords (detector, level, etc.). Pipelines are registered with a Pipe_ID, pipeline profile name, pbuilder arguments, kickoff criteria (detector in focal plane, gratings in/out, etc.), input and output data products (by File_ID), and method for generating the "root" part of output file names. The current standard data processing registry consists of 579 data products including a log file for each pipeline. There are 198 pipelines in the SDP registry consisting of various instances of 58 different pipeline profiles.

5.3.2. Observation status tracker

The Observation Status Tracker (OST) reads the registry when it first comes up. It then keeps track of what data has been archived. With pipeline data dependency information it can then kick off each pipeline at the appropriate time (when all its inputs are available). The OST is a type of state machine that simply takes in notifications and sends out notifications.

5.4. Software Reuse

5.4.1. Parameter files

All the tools in the Chandra toolkit are provided input via a standard parameter file interface. Each parameter file contains the parameter name, type and value. The parameter file API provides functions to read and write parameter files via C and C++ code as well as via command line functions.

We use parameter files all over the AP system. In addition to providing run-time parameters to each module, they are used to implement the notification file system, the registry information, "special processing" directives, and module persistent data.

5.4.2. OPUS

When implementing our pipeline processing system we looked at a number of OTS systems. We did not find any that would meet all our requirements and therefore developed all the AP modules in-house. After XRCF, we saw the need for some serious redesign work and took another look at the OPUS* system developed by the Space Telescope Science Institute.[6]

Although it did not meet all our requirements, it did help fulfill a number of requirements we had for distributing processing and providing GUIs to operations. So we decided to use OPUS to implement our pipeline processing at a very general level. All our pipelines must do the following:

1. get kicked off

*See http://www.stsci.edu/resources/software_hardware/opus

2. retrieve input data
3. run a set of tools to process data
4. archive output data
5. send some notifications
6. clean itself up

Each "stage" of our OPUS pipeline handles one of the above processing requirements. We still use our pipeline processing infrastructure (see section 3.6) to run the pipelines in the third OPUS stage above. Figure 4 shows different types of pipelines run by the OPUS:prs_run threads. The notification sent by the OST to OPUS contains registry information instructing OPUS which pipeline profile (*.ped file) to use along with the run-time arguments.

OPUS allows the user to start up any number of instances of a given stage. Distributing the pipelines to different nodes and monitoring the processing is handled with the OPUS infrastructure. Balancing the processing load is accomplished by assigning an appropriate number of stages to each node.

Figure 4 shows that we also use OPUS for two other purposes besides pipeline processing. OPUS is used to poll an input directory for ancillary data and monitor data ingest (see OPUS:dr_arc). OPUS is also used to implement the data receipt flow control (see OPUS:dr_telem).

6. POST-LAUNCH AP DESIGN ENHANCEMENTS

As with most missions, many challenges are encountered when the theoretical test cases are replaced with real world scenarios. The standard data processing (SDP) thread is sometimes not adequate to handle these new scenarios. This section describes some post-launch challenges and the enhancements to the AP system we implemented in response to those challenges.

6.1. Special Processing

Occasionally the standard processing of an observation will not produce the best data products. For example, a guide star chosen to be part of the "aspect" solution may fall on a bad part of the aspect camera. The resultant products are better when the "bad" guide star is ignored and so the observation needs some "special handling". There are too many different types of cases to handle them all in the software so a mechanism is needed that allows operators to pause processing, do something by hand, and then resume the automated processing thread. We run another instance of AP (SAP) to handle these "Special" cases.

6.1.1. Special processing parameter files

The AP pipeline processing system was enhanced to accommodate the "special processing" parameter files. When an observation needs special handling a parameter file with specific directives can be placed in a "special processing" directory. The AP system checks this directory at key points in the processing thread. If a parameter file for the given observation is found the directions in the parameter files are followed. Any special commands are recorded in the processing log. If the directive is simply "stop", the AP system requires a special processing comment file, describing the manual steps taken, before the automated processing will resume.

6.1.2. Special processing GUI

Sometimes the reprocessing of an observation requires only a subset of the pipelines be rerun. The Chandra Automated Processing Task Interface (Captain) was developed to make this type of reprocessing easier. Captain reads the registry and allows an operator to visualize the pipelines appropriate for a given observation. The operator can then choose the pipelines to rerun and Captain will set up the appropriate special processing parameter files, retrieve the necessary input data and run the pipelines identified in the GUI.

Figure 5 depicts a typical reprocessing run configured in Captain. It shows Observation 01237 configured to start reprocessing at the level 0.5 pipeline, OBIDET. Pipelines are displayed as ellipses, straight lines indicate data dependencies, and curved lines indicate notifications that instantiate a pipeline.

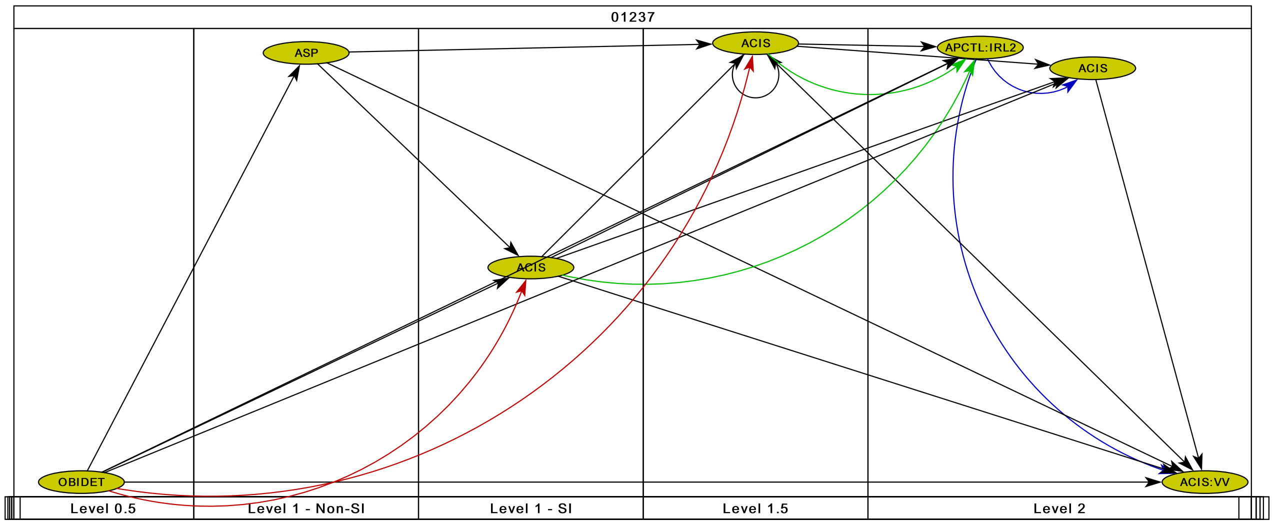

Figure 5. The Captain GUI used in Special Processing

6.2. Reprocessing

Sometimes a reprocessing task must be run on many observations in bulk and using Captain is not practical. The reprocessing thread (REPRO) may differ from the standard thread enough that the SDP infrastructure is not adequate. If a different set of pipelines must be run the registry can simply be constructed with only the relevant pipelines included. For the extra tasks, we have developed a layer that sits on top of OPUS.

6.2.1. Generic OPUS pipes (ad-hoc tasks)

We have configured OPUS for three distinct purposes in the AP infrastructure, ancillary data receipt, telemetry flow control, and CXCDS pipeline processing. It provides great tools for initiating and monitoring the processing. However, configuring OPUS can be a bit complicated and involves setting up many configuration files; we require 10 different configuration files to run pipeline processing.

The Generic OPUS pipes infrastructure allows an ad-hoc processing task to be specified by one configuration parameter file. The infrastructure then builds the OPUS configuration files so OPUS can be used to run the specific task. Any change to the single configuration parameter file is detected when the processing system comes up and the OPUS files are adjusted accordingly.

The Generic OPUS pipes infrastructure was developed to make configuring reprocessing tasks easier for development; the task is specified in one file. It makes it easier for operations to run the task because they use the same procedures and GUIs associated with AP.

6.3. Catalog (L3) Processing

The latest development effort is associated with the production of a Chandra source catalog that spans multiple observations (level 3 processing).[5] The processing system requires a different approach than standard AP for many reasons. First, the data can not be processed in L3 till it has passed the one year proprietary period. Second, the processing involves some very CPU intensive analysis and takes much longer than standard processing. It is so involved that we decided to purchase a Linux based Beowulf cluster with 14 nodes (28 CPUs) to handle the processing. Third, standard AP starts with raw telemetry and L3 starts with data products already in the archive. And finally, a totally different set of pipelines is run for L3 processing.

6.3.1. Beowulf cluster processing

Most of the data analysis tools are ported to many flavors of Unix (including Linux) as part of the Chandra Interactive Analysis for Observations toolkit (CIAO).[7] So the level 3 tools were already able to run on the Beowulf nodes. The only other executable we had to port was the Pipeline Controller. The rest of the AP infrastructure used in L3 processing runs on the SDP hardware (SUN Solaris workstations). We also had one small enhancement to allow a pipeline to run on a remote machine; this did not affect the SDP thread at all.

6.3.2. Reusing modules

The rest of the infrastructure for running L3 processing consists of modules already developed for SDP, SAP, and REPRO. A generic OPUS pipe (l3proc) manages the whole processing task for each observation as it becomes public. The appropriate data is retrieved from the archive using Captain modules. The L3 pipelines are configured with a separate instance of the processing registry. The pipelines are run and data is queued for ingest with the same modules used for SDP.

Configuring an instance of AP to run L3 processing required very few software enhancements. The ability to reuse existing modules to put together a system with new processing requirements validates the flexibility of the template driven approach in the AP design.[8]

7. FUTURE AP ENHANCEMENTS

Currently, the registry allows data products and pipelines to be registered. The notifications that generate a specific instance of a pipeline are separate objects. Generalizing the concept of notification and allowing notifications to be registered would make the system more flexible. Also allowing pipelines to "inherit" properties from a "base" pipeline would cut down on the amount of duplicated information in the registry and make maintenance easier. Integrating the registry more fully into the Archive system would make for cleaner interfaces.

8. CONCLUSION

The Chandra Automated Processing system began flight operations after five years of active design, development and testing. Enhancements have continued over the course of seven years of flight. Looking back at the development of such a complex system and considering what has worked well and what has not worked well, the following represents a summary of the "lessons learned" over the years:

1. Designing a flexible template driven processing system is well worth the effort when dealing with complex processing requirements.

2. Keep modules loosely coupled to avoid processing bottlenecks and make error recovery easier.

3. Try to partition the processing modules in such a way that the parallel processing can begin as early as possible in the processing stream.

4. Look for places where a data cache could improve reliability and performance.

5. In certain circumstances the simpler "old" technology (like files) performs better than the "newer" technology (like UNIX IPC or CORBA).

6. OTS software (e.g. OPUS) can sometimes be integrated into your system and save lots of development time even if the fit is not perfect. Look for a way to adapt the OTS software and fill a niche.

7. Do not assume external interfaces will be absolutely followed. You must make assumptions when designing your software, but make sure you add checks before passing data along. In other words, "Trust but verify".

8. Consider good regression testing practices when designing your system.

9. Use the few extra bits of telemetry bandwidth to avoid clock counter roll-overs. It will eliminate untold complexity in the processing software.

ACKNOWLEDGMENTS

Support for development of the CXCDS 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 under contract NAS 8-03060.

REFERENCES

1. http://www.chandra.harvard.edu/about/axaf_mission.html.
2. J. D. Evans *et al.*, "The Chandra X-ray Center Data System: supporting the mission of the Chandra X-ray Observatory." (these proceedings).
3. J. Nichols, C. S. Anderson, P. J. Mendygral, and D. L. Morgan, "Chandra Data Processing: Lessons Learned and Challenges Met." (these proceedings).
4. D. Plummer and S. Subramanian, "The Chandra Automatic Data Processing Infrastructure," in *Astronomical Data Analysis Software and Systems X*, F. R. Harnden Jr., F. A. Primini, and H. E. Payne, eds., *ASP Conference Series* **238**, pp. 475–478, 2001.
5. I. Evans *et al.*, "The Chandra X-ray Observatory data processing system." (these proceedings).
6. J. Rose, "OPUS-97: A Generalized Operational Pipeline System," in *Astronomical Data Analysis Software and Systems VII*, R. Albrecht, R. N. Hook, and H. A. Bushouse, eds., *ASP Conference Series* **145**, pp. 344–347, 1998.
7. A. Fruscione, J. McDowell, and M. A. Nowak, "CIAO: Chandra's user analysis system." (these proceedings).
8. J. D. Grier, D. Plummer, and K. Glotfelty, "The Chandra Source Catalog Automatic Data Processing System," in *Astronomical Data Analysis Software and Systems XV*, C. Gabriel, C. Arviset, D. Ponz, and E. Solano, eds., *ASP Conference Series*, 2005.