Changing software philosophy for ACIS operations as Chandra ages

N.R. Adams, et al

Changing software philosophy for ACIS operations as Chandra ages

Nancy R Adams-Wolk, Paul P. Plucinsky and Joseph DePasquale
Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA USA 02138

ABSTRACT

The Chandra X-Ray Observatory is about to start its 10th year of operations. Over the time of the mission, the Science Operations Team, ACIS (Advanced CCD Imaging Spectrometer) Operations group, has participated in spacecraft command load reviews. These reviews ensure the spacecraft commanding is safe for the instrument and the ACIS configuration matches the planned observation. The effectiveness of spacecraft command load reviews for ACIS depends on the ability to adapt the software as operations change in response to the aging of the spacecraft.

We have recently rewritten this software to start incorporating other spacecraft subsystems, including maneuvers and hardware commanding, to ensure the safety of ACIS. In addition, operational changes that optimize the science return of the spacecraft have created new constraints on commanding.

This paper discusses the reorganization of the code and the multiple changes to the philosophy of the code. The result is stronger, more flexible software that will continue to assist us in protecting ACIS throughout the Chandra mission.

Keywords: X-ray Chandra, PERL, Ground based C++

1. INTRODUCTION

The Chandra X-ray Observatory has been in operation for almost 10 years. The Science Operations Team (SOT), ACIS Operations group (ACIS Ops) is charged with several tasks, one of which is to review the planned spacecraft commanding to safeguard the instrument and to maximize the science return. The ACIS Ops group reviews the spacecraft command load file prepared by the Flight Operations Team Mission Planning (FOT MP) and runs a series of checks to confirm instrument configurations, timings and safety.

As the telescope has aged, the requirements for ensuring the health and safety of the instrument have changed. Some of the new requirements resulted in operational changes that in turn required changes or updates to the load review software. After several years, this code became bulky, difficult to read and very inefficient. This past year, we've undertaken the task to redesign the command load review code in order to make future changes fast and maintainable.

1.1. Examining the existing code.

The ACIS Ops team is comprised of several people who are familiar with the operations of the instrument. As part of the research and analysis required for the optimal operation of ACIS, we have all learned different aspects of scripting and coding. Only one person who helps maintain the command load review software was trained in computer science.

The first step in redesigning a code is to review the existing software. In this case, the software was a series of software scripts written in PERL. Originally written shortly after launch, these scripts handled the separate tasks required and the main calling script was the glue that held it all together. The command line script is lr, a script that uses the command load name to create the directory structure needed for the load review, opens an ftp connection to collect the tar archive from the secure machines at the Operations Control Center, and extracts the needed files. Then the acis_backstop.pl script executes. This script is the heart of the load review and where most of the updated software was placed. The purpose of this script is to: parse each line in the commanding backstop file, write the required hardware and software commands to an output file, extract information from the observation database, and review the commanding for timing errors or missing commands. This script became what is known as spaghetti code; a tangled mess of code fragments that were written by several people. Some of the fragments were in separate scripts and some were embedded directly in the main of the PERL script.

For each script we identified items that were either outdated by changes in the operation of the instrument or inefficient from a coding perspective. These items were highlighted for change. Then reviewed requests to improve the user efficiency and to reduce errors. We found that more autonomous checks were needed. Again, we perused each script,

keeping these items in mind, looking for areas where we could reduce errors and increase efficiency while adding new autonomous comparisons.

1.2. Functionality and upgradeability

The existing code was a simple while loop that contained a series of if statements. Each spacecraft command trickled down through these statements and actions were taken as needed. Some spacecraft commands would have multiple if statements that were fulfilled. Other spacecraft commands were not used, but still checked through each if statement. This made any upgrades very difficult to find. Variables had non-identifying names and were all within the global scope. Several blocks of code could act on the same variable, which made it difficult to debug. The original comments were sparse and following the local became difficult. The main contained only one function. Error flags were numbered and not clear where they were set or cleared.

All of these items were identified for change. To be able to react to a new standard operating procedure requires the review code to be changed quickly and cleanly. The code needed to be split into several functions with clearly identifiable variables and function names. Some items needed to be within the local scope of a function, while other items clearly needed to be global. The error collection method needed to be overhauled into something far more functional and easy to maintain.

1.3. Functionality changes.

The ACIS Ops team brainstormed for items that could supplement the current spacecraft command load review and eliminate mistake prone manual checks. Among these were adding calculations of the telemetry buffer space when there is a change in spacecraft telemetry format, reading and comparing the diagnostic tests during the perigee passages, commanded and requested window block checks and Next In Line (NIL) commanding checks when ACIS is not the instrument in the field of view of the telescope. Most of these items had previously been done as manual checks. These are more prone to errors. One of goals was to move the load review code toward a more autonomous tool with less human intervention.

Additional functionalities that are a function of the spacecraft age are: identifying bright X-ray sources crossing the ACIS detectors during maneuvers or within the field of view of an observation and tracking the Earth's solid angle on the ACIS radiator. Earlier in the mission, these items were not as strong of a concern. Now that the spacecraft has aged and we have more constraints on scheduling targets, following these items became necessary.

acis-backstop.pl Flow:

```
declare all variables for the script
open each file needed for input and output
read the continuity files (also known as history files)
set flags for status of the spacecraft
while (read the input file)
    If (command is command a)
        <act on command a>
    If (command is command b)
<act on command b>
…<commands not fulfilling an expression fall through>
end while
check for error flags set (one check per flag)
print errors to out file and standard out
close files.
```

Figure 1. Original *acis-backstop.pl* pseudo code flow

2. Changing the code efficiency

Once all of the code had been reviewed for problem spots, a basic redesign was required. The original code followed a very simple model. Figure 1 displays the basic flow in pseudo code.

This arrangement was undesirable for several reasons:

- Every command in the backstop file runs through every if statement in the main while section. Several of the commands in the backstop file have nothing to do with the load review and can be ignored.
- Setting an error flag and then checking the flags later made the code both difficult to read and difficult to maintain. Instead of acting on errors as they occurred, the code reported one or more errors had occurred, making it more difficult to find in the output product. Error flags were also created with variable names that did not reflect the actual error. Maintaining the code become nearly impossible.
- Several of the actions performed where in separate codes. These codes often made system calls and wrote their own output files to temporary space. The worst example of an inefficient script was the archive script. This PERL script would access the database for a single observation and report the output to a file. Then the acis-backstop.pl script would open that file and copy the contents to the output of acis-backstop.pl. A typical load can have 20 or more observations; this means there are 60 or more file actions that are not necessary.
- The main script made liberal use of system calls instead of using more efficient built in PERL commands.
- The original script only contained one function, a function to parse the input file lines. All other code was in-line. Again, this made maintenance and updates almost impossible.
- Many lines of code had been copied and reused. This would be much easier to maintain within a function where a single change would be reflected in several function calls.

To address the weaknesses of the original code, we developed a new design. We created a set of guidelines to follow

- Remove unnecessary system calls.
- Replace external script calls with functions if possible.
- Replace ambiguous variable names with clear and easily identifiable names.
- Remove the error flag section and replace with a linked list or array of errors.
- Replace extraneous code with functions to optimize the maintenance of the code.

3. Changes to the code design.

Changing the original code into a more efficient code was done in a series of steps. The process was broken into three steps: streamlining the original code, adding missing functionality and adding new external codes. At each step, the code was tested against several command loads that had been run in the past. The code was under configuration management using the concurrent version system (cvs) and each major section of the redesign was checked into the repository.

3.1. Streamlining the existing code

Following the above design guidelines, the original code was streamlined. Where possible, existing scripts were replaced with functions and the number of disk accesses was reduced. Several sections of the code were broken down into functions to reduce repeated code. We also utilized the PERL associated array to create hashes for spacecraft hardware and software commands that we were interested in. Now when parsing the input file, the script determines at the beginning of the while loop if this is a command that requires further action. If not, the next line of the input file is read and tested. Each type of spacecraft command was bundled into a function for processing. These functions are named to indicate the subsystem of the spacecraft and/or the type of commanding that is being processed. Maintenance and upgrading code will be much faster since it will be easier to find where a change will occur.

Some of the existing scripts were already streamlined, but simply replacing them, as a function was not always a viable option. One example was the script that calls the archive database to find the requested parameters for an observation. The script would open a database connection, collect the information for one Obsid and then place the results in a temporary file to be used by the main script. Instead of creating a function that would connect to the database each time it was called, we created a single database connection that stays open until the input file has been completely processed.. This reduced the number of calls to the database and increased the execution time of the code.

New *acis-backstop.pl* design:

Open each file needed for input and output: via a function
Read the continuity files (also known as history files)
Open a connection to the database
Fill an associated array containing the spacecraft status
While (read the input file)
    Check if a needed command
        Switch:
            Radiation zone command
            <process radiation zone commands>
            Comm. command
            <process communication commands>
            dither command
            <process dither commands>
            grating commands
            <process grating commands
            …

end while
close the database connection
review the array of errors, report as needed
write the continuity file based on the spacecraft status
Close files.

Figure 2. New *acis-backstop pl* flow

3.2. Adding missing functionality

Several items of missing functionality were added after the code was streamlined. The majority of these items had been manually checked by the ACIS Ops reviewer. While Chandra has aged, the staffing levels have decreased. The ACIS Ops team needed to maximize the efficiency of load review code to perform as many consistency checks as possible to minimize the workload of the on-call ACIS Ops team member. We identified many areas that had been considered manual checks in the past and added these to the code.

Items included:

- Reading the engineering request files and comparing the expected state with the commanded state for a Next-in- Line (NIL) observation.
- Reading the diagnostics request file and comparing the expected state of the radiation zone observations with the commanded state.
- Reviewing each commanded ACIS parameter block and confirming that it matched the requested science instrument mode (SI_MODE) in the archive. This included checking for the need for a new bias file, since this results in the same SI_MODE, but a different ACIS parameter block.
- Comparing each parameter in the archive against the parameters in the commanded ACIS parameter block
- Comparing each parameter in the archive against the parameters in the commanded ACIS window block.

- Adding several items to the history information for continuity purposes, see below.
- Checking the time to transfer data from the telemetry buffers given the telemetry format.

ACIS is an extremely configurable instrument. To control ACIS, a parameter block giving the specified parameters is loaded into either a Timed Exposure command slot or a Continuous Clocking command slot. In addition, up to 6 windows can be programmed on each active CCD. These are specified in a window block. Again, there are different slots to record the timed exposure window block versus the continuous clocking window blocks [1]. The software that programs the spacecraft commanding schedule tracks which commanding blocks are stored on board for each command load. When a particular command load contained a window block that was already stored on board, it is not reloaded. This made a very difficult manual check for the ACIS operations person.

To eliminate the need for searching past loads to confirm that the correct window block was stored on board, we added tracking of the stored parameter blocks and window blocks. In addition to storing the parameter and window blocks, we started to record the radiation zone preparation steps. Command loads are often ended near perigee. If this happens after ACIS has been commanded to prepare for the radiation zone passage, we can miss these commands in the next command load. By tracking these items in continuity or history files, we remove one more manual check for the ACIS Ops team.

3.3. Final result

The final result was a script with more autonomous comparisons and command safety checks. The actual length of the PERL code has increased almost by 50%, but the readability of the code and the ease of maintenance are the real benefits. Now it is very easy to insert new code as needed. We were able to add new code, and then remove it when needed when we were testing a change to the instrument operations.

4. New External Code

With the instrument going on 10 years, it was determined the spacecraft configuration was a factor that needed to be considered during our load reviews. We are in the process of adding two new codes to the load review process to aid in tracking the spacecraft orientation and how it affects the ACIS instrument.

4.1. Boresight Angles of Bright X-ray Sources

One concern of any observation is the total mission dose that ACIS receives. CCD cameras do have a finite lifetime and there is cumulative damage from the photons impinging on the detectors [2]. When an observation of a bright X-ray source is planned, the ACIS Ops team often simulates and studies the expected damage. Using clear dose guidelines, we prevent any observation from having more than 1% of the mission dose. One area that we had not been attentive to was the bright X-ray sources during slews and holds. As Chandra slews across the sky, ACIS is exposed to any X-ray photons that may be focused by the mirrors. Since launch, the FOT mission planning team has monitored several bright X-ray sources. Now ACIS has added a new code that records the boresight angle of bright x-ray sources we slew past and report those that are in our field of view during an observation.

4.1.1. Why track bright sources?

ACIS is sensitive to the maximum dose that the instrument can receive over the lifetime of the mission. In the past, we have tried to determine the dose of photons incident on the detector (which is not the same as photons detected) based on observations. However, the damage the CCDs receive is cumulative. We are now at the point in the mission where it is desirable to track as many bright X-ray sources as possible. There are two issues of concern at this point. The first involves the slew of the telescope. To prevent any single pixel from receiving too many photons, the telescope moves in a small lissajue, a dither pattern that is user defined. Once the data are on the ground, the aspect solution is reconstructed and the data are undithered, bringing the photons back into a point source. This system is described in the proposers guide and in Weisskopf et al. [3]. During a slew, there are no guide stars to lock on and the dither is not active. Any bright X-ray source that happens to fall on axis will be focused onto one or two pixels of the CCDs and can cause significant damage. After several discussions and simulations, we determined that only SCO X-1 would be damaging during a slew at the maximum acceleration of the telescope. However, there are several other bright X-ray sources that may cause damage if they are near a target and the telescope has a slower slew rate.

The second issue is a bright X-ray source in the field of view while observing another X-ray source. In most cases, we expect that any bright X-ray source that was identified as a threat to ACIS will not be observed. It is entirely possible that a source nearby may be observed and that the bright X-ray source falls within the field of view of ACIS. The expectation is the source will be far enough off axis, that the point spread function will dilute the photon dose on any given pixel. The goal is to identify any known bright X-ray source that is within a 35 arc minute radius of an observation and confirm that it will not damage the instrument.

We downloaded all of the available All Sky Map data from the Rossi X-Ray Timing Explorer (RXTE)[3]. Objects with count rates that were at least 0.5 crab for more than 10 samples were flagged. Then a human reviewed all of these sources to determine which objects are most likely to be a threat to ACIS. The final list contained 58 sources.

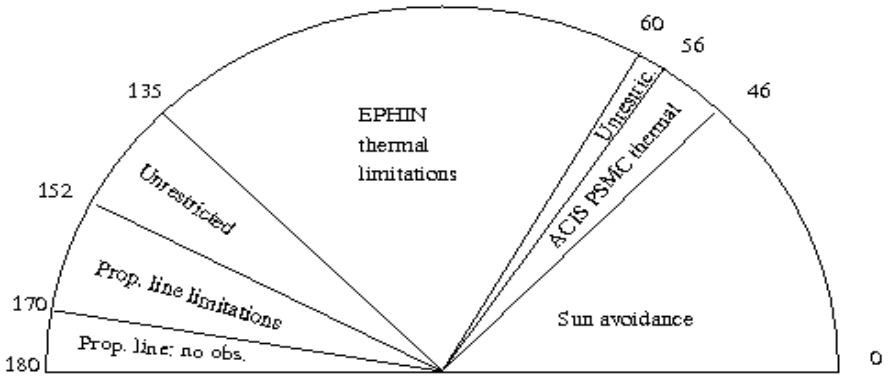

Figure 3. Constraints on Spacecraft Solar Array Angles

4.1.2. Code Source

This code was modeled after a MATLAB code written for the FOT. Since our work is done on a Solaris machine, we needed a Solaris executable. Unfortunately, the MATLAB code could not be converted to a Solaris binary due to licensing issues. In reviewing the MATLAB code, we decided to rewrite the code in object oriented C++. This allowed for code reused for the solid angle code.
Both the solid angle and the bright source code require the maneuver information. The spacecraft maneuvers are recorded with a unit quaternion. The starting and final quaternions and the time of the slew are distributed as part of the backstop tar package. By taking the quaternion, we can determine where the spacecraft is pointing, and we can determine the boresight angle to any position in the sky in RA and Dec.
Part of writing the C++ code for the bright-source code meant writing a class that would read the time in several different formats and storing it in one format. This time class could then be used to match the time stamps for several different sources of information.

4.2. Earth Solid Angle

The ACIS thermal control system includes two passive radiators, referred to the warm and cold radiators [1]. The warm radiator cools the detector housing and the cold radiator cools the focal plane.

When the spacecraft is oriented so that the earth is in the field of view for the cold radiator, there is actually a warming due to the albedo and infrared emissions from the earth. As the mission continues, Chandra's orbit is changing such that the perigee will be significantly lower than at any previous point in the mission. This results in a larger solid angle of the

Earth on the cold radiator and a larger spike in the focal plane temperature during perigee passages. An increase of focal plane temperature is undesirable. From an operations standpoint, we have made changes to the operating procedures and we are in the process of studying new ways to maintain a stable focal plane temperature.

Complicating the matter is the pitch constraints on the spacecraft. As the telescope has aged, we have found that certain pitch angles affect the thermal environment of the spacecraft. The FOT mission planning team must plan observations with the health and safety of the spacecraft in mind. Figure 3 displays the constraints on pitch with respect to which subsystem is affected. We have found that the pitch between 120 to 160 degrees warms the focal plane (Figure 4). Unfortunately, this is one of the few regions of solar array angle pitch that has few constraints.

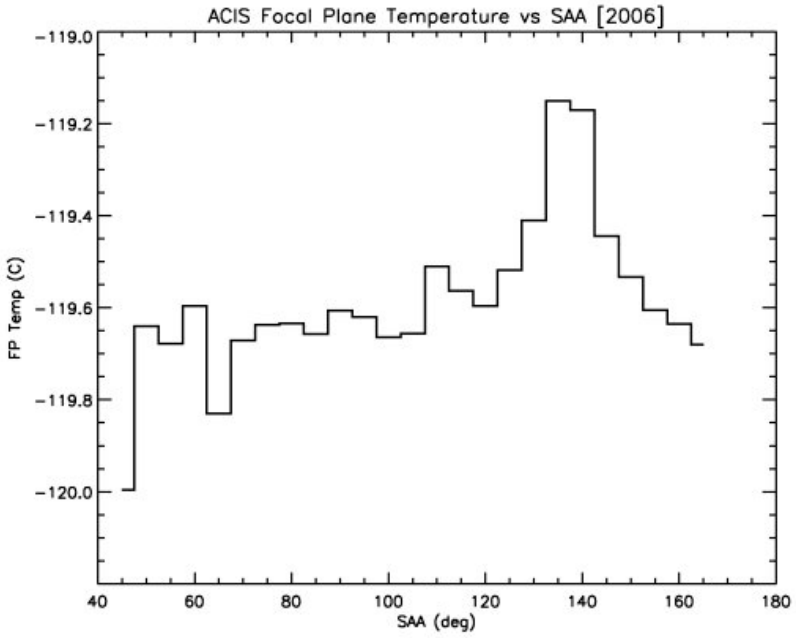

Figure 4. ACIS Focal Plane Temperature as a function of the Solar Array Angle (pitch).

4.2.1. Identifying the warm observations

A warm and variable FP temperature is undesirable since the performance of the CCDs is a sensitive function of the temperature. The best performance is achieved with a cold and stable focal plan temperature. Unfortunately, as Chandra has aged, this is not always possible. We are in the process of writing a C++ code that will review the spacecraft pointing and maneuvers and the Chandra ephemeris to determine when we have the possibility of a warm observation. Any earth in the cold radiator's field of view will be identified as a warm observation as well as those with pitch angles greater than 120 degrees (Figure 4). This "tail-Sun" orientation and the Earth-shine on the radiator can both increase the focal plane temperature, resulting in a less accurate calibration.

Major elements of the bright-source code will be reused. We can easily use the classes for Time and Maneuvers. In addition, an Ephemeris class will be added. This class will read in an ephermis and interpolate the ephemeris to match

the maneuvers so we can determine the solid angle of the earth on the cold radiator. The output will contain the start and stop times of warm observations.

5. Conclusions

Over the past 10 years, the code used for ACIS Ops command load reviews has been modified to address the changing needs of the spacecraft. At this point in time, we have redesigned the software to allow for easier software updates to match changing requirements of spacecraft operations, and a clean design to assist in maintenance of the code. We have also started programming for issues of concern for the instrument such as the bright X-ray source avoidance and the solid angle of the Earth in the ACIS cold radiator's field of view. These elements will allow us to prevent damage to the instrument and help plan a better science product by keep the instrument within desired thermal operating limits.

REFERENCES:

[1] Nousek, J. A.,"Science Instrument Operations Handbook for the AXAF CCD Imaging Spectrometer (ACIS)", ACIS-PSU_SOP_01, Nov. 20, 1997.

[2] Butt, Y.,"Operational Limits for X-ray Photon Dose on ACIS", CXC Memo, May 3, 2002.

[3] Weisskopf, M. C.; Aldcroft, T. L.; Bautz, M.; Cameron, R. A.; Dewey, D.; Drake, J. J.; Grant, C. E.; Marshall, H. L.; Murray, S. S. Experimental Astronomy, v. 16, Issue 1, p. 1-68 (2003).