Proceedings of the American Control Conference
Philadelphia, Pennsylvania • June 1998

An Overview of the Fault Protection Design for the Attitude Control Subsystem of the Cassini Spacecraft

G. Mark Brown, Sue A. Johnson

Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California

ABSTRACT

This paper describes Cassini's fault tolerance objectives, and how those objectives have influenced the design of its Attitude Control Subsystem (ACS). Particular emphasis is placed on the architecture of the software algorithms that are used to detect , locate, and recover from ACS faults, and the integration of these algorithms into the rest of the object-oriented ACS flight software. The interactions of these ACS "fault protection" algorithms with Cassini's system-level fault protection algorithms is also described. We believe that the architecture of the ACS fault protection algorithms provides significant performance and operability improvements over the fault protection algorithms that have flown on previous interplanetary spacecraft.

THE CASSINI MISSION AND SPACECRAFT DESIGN

The Cassini spacecraft, shown in Figure 1, was launched in October 1997 by a Titan IV launch vehicle with a Centaur upper stage. Cassini will reach Saturn in June 2004 after a seven year journey that will include gravity assists from Earth, Venus, and Jupiter. After establishing an orbit about the planet, Cassini will tour the Saturnian system through June 2008, using its suite of twelve science instruments to acquire detailed images and sense the local environments. Cassini will also release the Huygens probe on the first or second of many planned encounters with Titan, the only moon in the solar system with a substantial atmosphere.

Cassini is a three-axis stabilized spacecraft with rigidly mounted antennas and rigidly mounted science instruments. At close solar distances, the spacecraft shades its thermally sensitive elements by pointing its four meter diameter High Gain Antenna directly at the Sun, while communicating with the Earth via one of two available Low Gain Antennas. During the Saturn orbital tour, the spacecraft attitude is much less constrained, and can therefore be dictated by scientific objectives for periods of up to sixteen hours per day. During the remaining eight hours per day, the spacecraft points its High Gain Antenna to Earth and "plays back" recently gathered scientific data from a four gigabit onboard recorder.

Three radioisotope thermoelectric generators (RTGs) deliver approximately 825 watts at the beginning of the mission, decaying to 650 watts at the end of the mission. All onboard customers receive switched power from this source via a regulated 30 volt power bus. The onboard avionics are distributed in several engineering subsystems, including:

The *Command and Data Subsystem* (CDS), which accepts ground-provided command sequences, and then distributes stored commands at the appropriate times to the engineering subsystems and science instruments. The CDS also coordinates all the onboard intercommunications, and collects and formats the science and engineering telemetry.

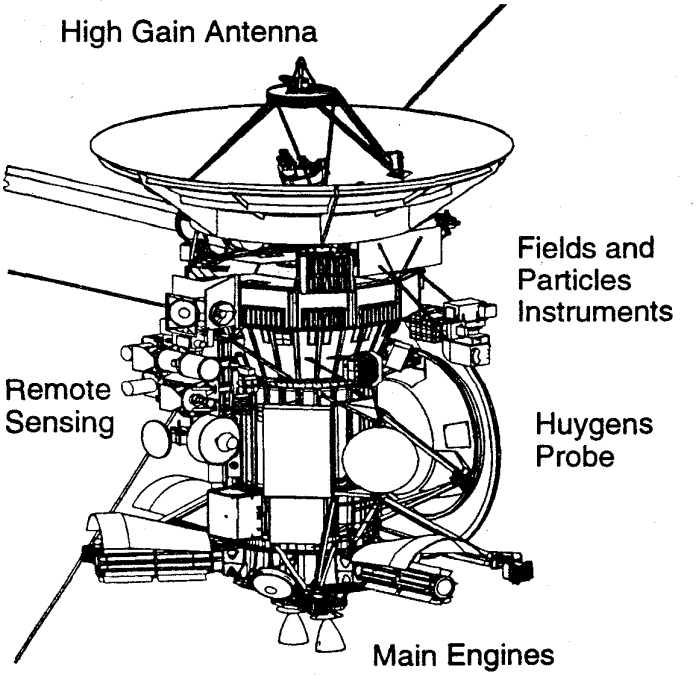

Figure 1: The Cassini Spacecraft

0-7803-4530-4/98 $10.00 © 1998 AACC

The *Radio Frequency Subsystem* (RFS), which receives ground-transmitted commands for subsequent storage and execution by the CDS, and transmits CDS-formatted telemetry to the ground.

The *Attitude Control Subsystem* (ACS), which is described in the following section.

CASSINI'S ATTITUDE CONTROL SUBSYSTEM

Cassini's Attitude Control Subsystem (ACS) estimates and controls the spacecraft attitude, responding to ground-provided pointing goals for the spacecraft's science instruments and communications antennas with respect to targets of interest (e.g. Saturn, Earth). The ACS also executes ground-commanded spacecraft velocity changes.

Figure 2 is a functional block diagram of the Cassini ACS hardware. Each rounded box represents a separately powerable ACS assembly. Stacked groups of these boxes indicate the available redundancy. Circles indicate propulsion system elements that are controlled by the ACS.

The ACS supervisory, estimation, and control algorithms are executed by one of two block-redundant ACS Flight Computers (AFCs). Each AFC houses a 1750A microprocessor with 512 Kwords of Random Access Memory (RAM). The object-oriented flight software is written in Ada; see Reference 1 for more information on the ACS flight software architecture.

The AFCs are remote terminals on the spacecraft's MIL-STD-1553B Command and Data Bus. Each AFC receives ground commands from the CDS via this bus, and provides telemetry and other ancillary data to the CDS for subsequent distribution to ground and onboard customers. For more details on Cassini's CDS and the Command and Data Bus, consult Reference 2.

The prime AFC coordinates the activities of all the ACS peripheral assemblies via two-way transactions on a separately dedicated ACS Bus. The ACS Bus conforms to MIL-STD-1553B electrical standards, but uses a custom protocol that provides more than 32 remote terminal addresses and supports message lengths of more than 32 words.

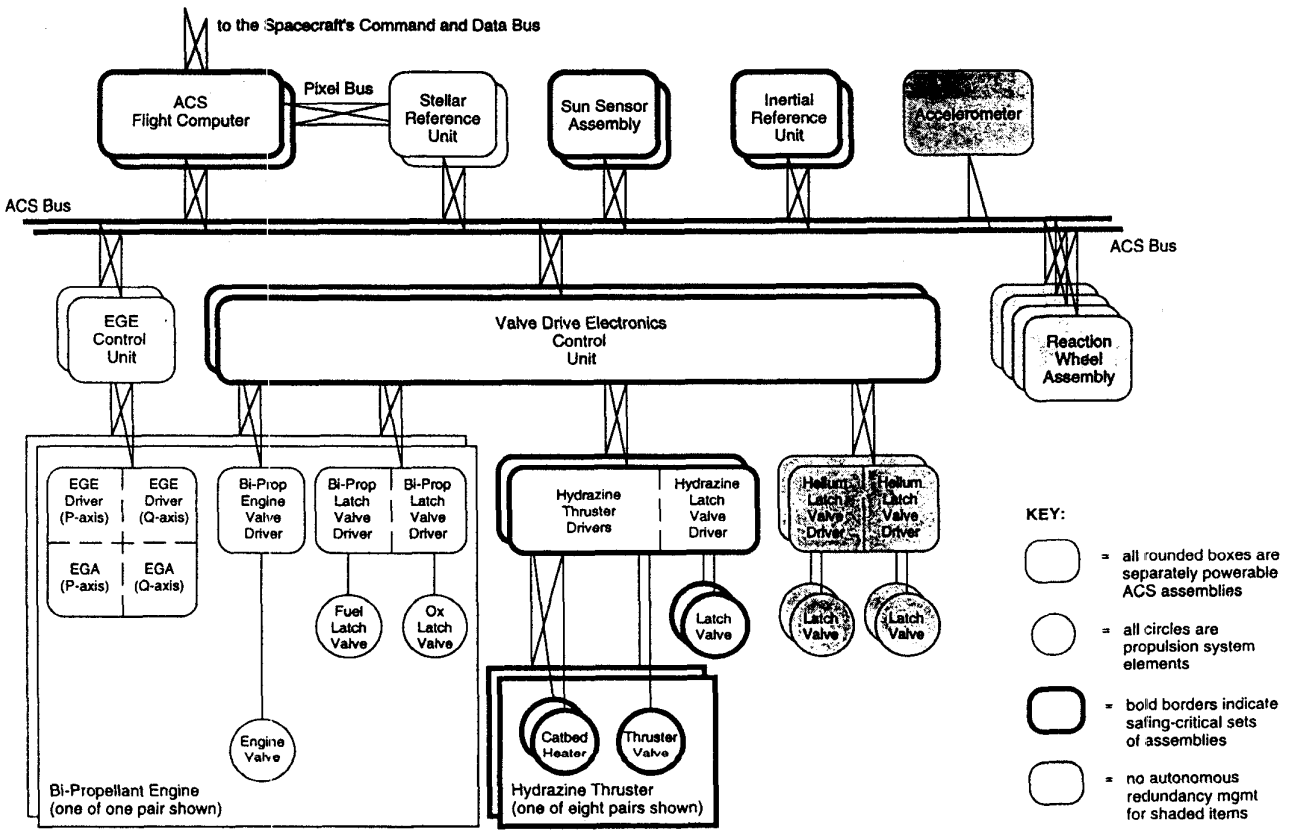

Figure 2: Cassini ACS Functional Block Diagram

Three-axis attitude sensing is provided by block-redundant Inertial Reference Units (IRUs) and block-redundant Stellar Reference Units (SRUs). Each IRU contains four hemispheric resonator gyroscopes, arranged in an orthogonal-triad-plus-skew configuration. Each SRU is a 15-degree square field-of-view star tracker that can provide either AFC with up to 50,000 pixels per second through a dedicated Pixel Bus. AFC-resident software algorithms are able to establish and maintain stellar reference by comparing incoming pixel frames to an onboard catalog of approximately 5000 stars. Three to five stars are commonly tracked at any one time, and tracking can continue at turn rates of less than 0.25 deg/sec.

The ACS acquires stellar reference by first locating the Sun and then Sun-pointing the High Gain Antenna using feedback from one of two available Sun Sensor Assemblies (SSAs). Each SSA provides two-axis digital gray-coded output over a 64-degree square field-of-view.

The desired telecommunications rates require antenna pointing accuracies of a few milliradians. The ACS provides this level of pointing control via infrequent firings of 0.9 Newton hydrazine thrusters. The sixteen available thrusters and the other propulsion elements are all managed by one of two block-redundant Valve Drive Electronics (VDE) control units.

Scientific observations during the Saturn orbital tour require higher pointing accuracy and stability, as well as frequent spacecraft repositioning. During this mission phase, the ACS employs three orthogonally oriented Reaction Wheel Assemblies (RWAs), and the thrusters are used primarily for momentum management. Turn rates during both the cruise and orbital phases are kept below 0.75 deg/sec.

Small velocity corrections are accomplished by timed firings of the hydrazine thrusters. For large velocity corrections such as Saturn Orbit Insertion (SOI), the ACS employs one of two available 450 Newton bi-propellant engines. Each engine nozzle is articulated by two physically-dedicated Engine Gimbal Actuators (EGAs), whose extensions are managed by either of two available Engine Gimbal Electronics (EGE) control units. Engine burns are terminated autonomously based on feedback from a single accelerometer (ACC).

CASSINI'S FAULT TOLERANCE OBJECTIVES

Cassini has two fundamental fault tolerance objectives:

1) During all mission phases, the spacecraft must be able to *fail safe* by autonomously locating and isolating any single failure, recovering to a thermally safe and commandable attitude, and then waiting for further instructions from the ground. Since ground operators will only attempt to make contact with the spacecraft once per week during low-activity mission phases, the spacecraft must be able to keep itself safe for at least two weeks.

2) During a few selected time-critical activities (e.g. Launch, SOI, and Probe Relay), the spacecraft cannot afford to simply isolate a failure and wait for ground assistance. At these times, the spacecraft must be able to *fail operational* by autonomously recovering a much larger set of its capabilities and then proceeding with a previously uplinked "critical" command sequence.

It is important to note that the ground-generated command sequence for each time-critical activity is an integral part of Cassin's "fail operational" capability. Following an autonomously detected failure in a critical sequence, the CDS will temporarily suspend the sequence, coordinate an autonomous fail-safe response, and then restart the sequence from the last achieved "checkpoint". Each critical sequence has several checkpoints, each of which enforces any necessary differences between the fail-safe configuration and the operational configuration for subsequent critical sequence activities.

ACS FAULT PROTECTION ARCHITECTURE

Figure 3 illustrates the architecture of the AFC-resident flight software algorithms that perform autonomous detection, isolation, and recovery from failures of ACS assemblies and ACS-controlled propulsion elements. These algorithms, which will be subsequently referred to as the ACS fault protection algorithms, also coordinate the ACS participation in system-level fault responses. The primary architectural goals, all of which were motivated by recognized shortcomings of the fault protection algorithms on previous interplanetary spacecraft, are to:

1) Improve the diagnostic accuracy of the algorithms by creating and making use of explicit "goodness" indications in addition to the typically available "badness" indications.

2) Improve the operability of the spacecraft by using knowledge of the subsystem's present goals to choose an appropriate level of response.

3) Decompose the autonomous tasks in a natural manner to facilitate the development phase analysis and testing, as well as any eventual modifications that might be required during the eleven-year operations phase.

The primary architectural components are:

Error Monitors, which test local performance measures against expectations, apply discriminating filters, and then output a color-coded opinion. The available output colors and their interpretations are:

Black:	no opinion
Green:	performance meets expectations
Yellow:	performance is unexpected, but does not merit autonomous response
Red:	performance is unacceptable, and merits immediate autonomous response

Activation Rules, which evaluate subsets of the color-coded error monitor outputs in the context of the subsystem's current hardware configuration and activity goals, diagnose the most likely causes of anomalous behaviors, and then activate one or more appropriate response scripts.

Response Scripts, which isolate failed equipment and recover a desired level of subsystem functionality. They do so by issuing commands directly to the subsystem, directing the activities of autonomous repair managers, and/or requesting external assistance via CDS-collected alert messages.

Repair Managers, which track the success or failure of past corrective measures for each piece of equipment, and then determine the most appropriate corrective measure to try next. Repair Managers use the same command interface as Response Scripts, and thus they can be thought of as special-purpose subroutines that are exercised by the Response Scripts.

All four of the above-described components are exercised during each ACS computation cycle (i.e. every 125 msec). The Error Monitors are distributed throughout the flight software, usually at the earliest possible test point. Tests are performed and opinions are generated throughout each computation cycle. The Activation Rules, Response Scripts, and Repair Managers are all centralized and are executed during reserved time slices at the end of each computation cycle. Response Scripts are prioritized via a simple ordered list. Two or more Response Scripts can be simultaneously active; however

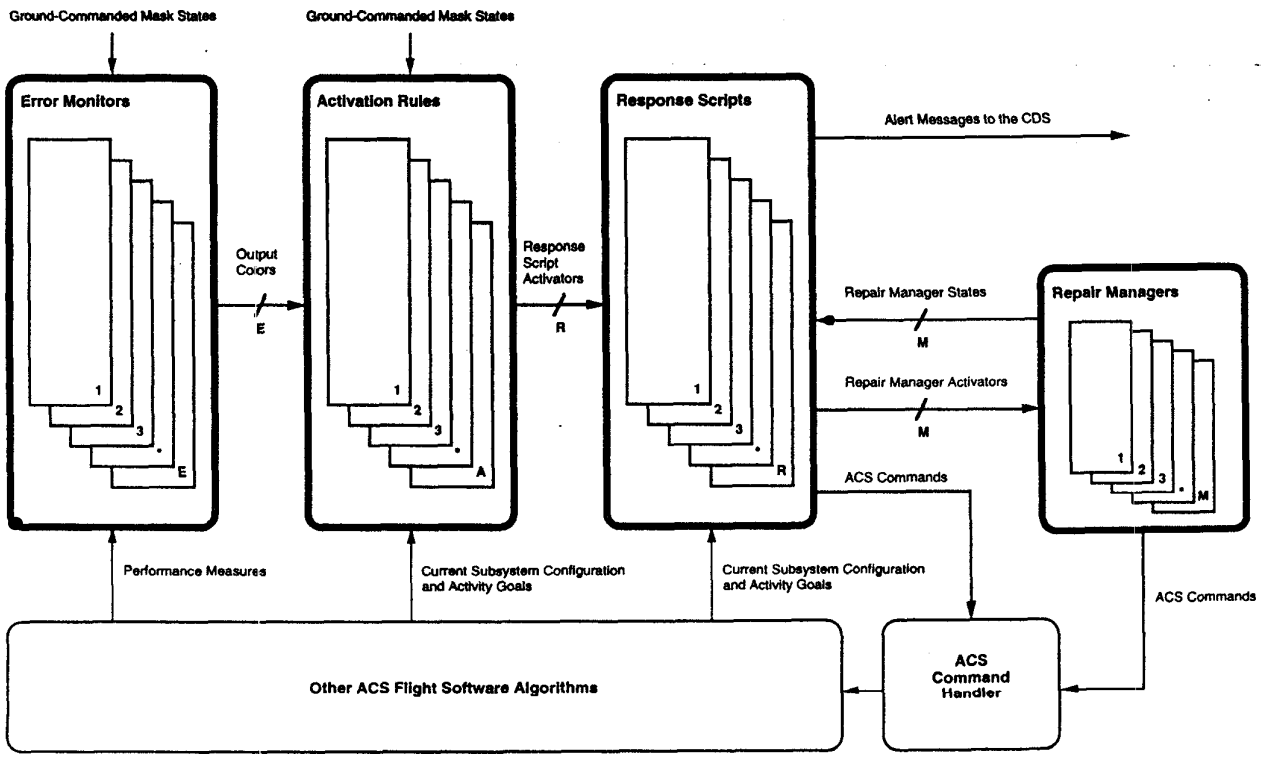

Figure 3: Architecture of the ACS Fault Protection Algorithms

the actions of lower priority scripts are delayed when higher priority scripts consume all of the available computation time.

Error Monitor Architecture

Each error monitor is composed of three parts as shown in Figure 4. An error test compares a variable or event to an expected value or range of values. The output of the error test is expressed as an opinion, either *"Expected"*, *"Tolerable"*, or *"Unacceptable"*. *"No Opinion"* is expressed when sufficient conditions for testing the variable are not met. For example, all of the error monitor opinions associated with an assembly are set to *"No Opinion"* when that assembly is turned off.

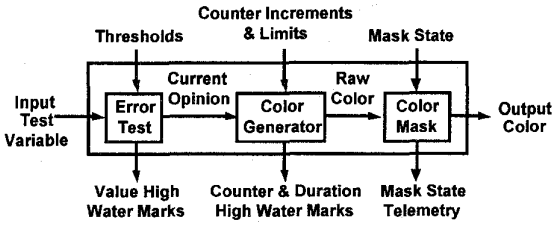

Figure 4: Error Monitor Architecture

A color generator filters the opinions produced by an error test, incrementing or decrementing a persistence counter during each computation cycle. Table 1 shows how a typical color generator operates on a persistence counter called redCount. The color generator will not decrement the counter below zero, and will not increment the counter above a ground-prescribed Red limit.

Table 1:
Color Generator Operations on redCount

Current Opinion	Red Count Update
No Opinion	redCount = 0
Expected	redCount = redCount - RedDec
Tolerable	redCount = redCount - RedDec
Unacceptable	redCount = redCount + RedInc

There are four types of color generators. The *Standard* type has a redCount that is governed by a Red limit, a Red Increment, and a Red Decrement. The raw color of the monitor is Green when the redCount is zero, is Red when the redCount is equal to the Red limit, and is Yellow at all values in between. After reaching the Red limit, an expected opinion will cause the redCount to decrement, returning the color to Yellow, or eventually to Green. A *Latched* type differs from the *Standard* type by latching the redCount at the Red limit, even if the opinion eventually improves. A *Latched* error monitor must be reset to re-detect the failure. A *Caution* type of error monitor is used in situations where continuing in the presence of a failure is preferable to terminating an event based on a weak indication. A *Caution* monitor cannot output a Red color remaining Yellow even after the redCount reaches the redLimit. The fourth type of error monitor is a *Bipolar* monitor. An example of this type is documented in Reference 10. A *Bipolar* error monitor uses two test thresholds and two persistence counters.

A color mask is applied to the raw color produced by the color generator. The state of the color mask is controlled by the operations team, as described in the "User Interfaces" section of this paper. If a monitor has been masked by the operations team, its raw color is unchanged, but its output color is forced to Black.

A simple example of an error monitor is MDC_Sun_Search_Timeout. Whenever a sun search is in progress, the opinion provided by the error test is "Unacceptable", and the redCount increments by one count during each computation cycle. If the redCount is non-zero but less than the Red limit, the raw color is Yellow. After 30 minutes, the duration of a full sky search, the redCount reaches the Red limit and the raw color becomes Red. If the Sun is detected at any time before the redCount reaches the Red limit, the search is declared complete, the opinion of this monitor immediately reverts to "No Opinion", and the redCount immediately returns to zero.

There are a total of 313 error monitors in the ACS flight software.

Activation Rule Architecture

Activation rules use the current error monitor colors, the current ACS mode, the current ACS configuration and the commanded ACS goals to decide to activate one or more response scripts. Each activation rule requires at least one Red error monitor. This enables a short-cut in the flight software processing: if there are no Red error monitors, none of the activation rules are evaluated.

Each activation rules starts with a diagnosis, which combines all the relevant information into a simple true/false test. A diagnosis takes the form of the following example, where the index "id" shows that there is one rule specific to SRU_A, and another rule specific to SRU_B:

```
vital_pixel_link_failure(id) =
        sru_is_vital(id)
    && anyRed(sru_frame_completion_error[id],
              sru_pixel_transfer_error[id],
              pixel_outside_range)
```

The expression "sru_is_vital(id)" indicates whether the SRU referred to by "id" is considered vital. To be vital that SRU would have to be the prime SRU and the ACS would have to be in a mode that requires celestial reference. The expression "anyRed()" indicates whether there are one or more Red colors among the listed set of error monitors. Two of the listed monitors are indexed by [id], indicating that there is an error monitor of this type for each SRU. There is only one pixel_outside_range error monitor, which is shared by the two SRUs.

Each activation rule also identifies one or more response scripts that are to be activated when the diagnosis is true. The activation rule that uses the above-described diagnosis is:

```
elseif (vital_pixel_link_failure[id])
    {
    terminate_burn.activate(non_critical_burn);
    repair_vital_sru.activate(id);
    }
```

Groups of activation rules are organized as lists of "if diagnosis A is true, activate B, elseif diagnosis C is true, activate D, and so on". Each group of activation rules addresses the possible failures within a single assembly (such as SRU_A), or a single ACS function, such as attitude control. The order of the activation rules within each if-elseif-elseif list is prioritized. The diagnoses requiring the most aggressive responses are usually listed first. In the example above, when there is a detected pixel link failure on a vital SRU, two response scripts are immediately activated. The first response script will stop any noncritical burn that may be in progress, and the second response script will initiate repairs to the vital SRU.

There are a total of 310 activation rules in the fault protection software.

Response Script Architecture

Response scripts respond to "activate" commands from the activation rules. Once activated, a response script directs itself to completion, restoring some aspect of the subsystem functionality by issuing appropriate commands and alert messages, and/or directing the actions of a repair manager.

As an example, the state transition diagram for the unsupported_safing response script is shown Figure 5. This script responds to an apparent CDS failure, by commanding ACS to stop whatever it is doing and turn to a safe attitude that will support Earth communications. Once activated, this response script transitions to the first state, where it commands the ACS to a stable flight mode. The script then transitions to its second state, where it waits until the commanded mode is achieved. Once the commanded mode has been achieved, the script transitions to its third state, where it commands a turn to an onboard-stored default safe attitude.

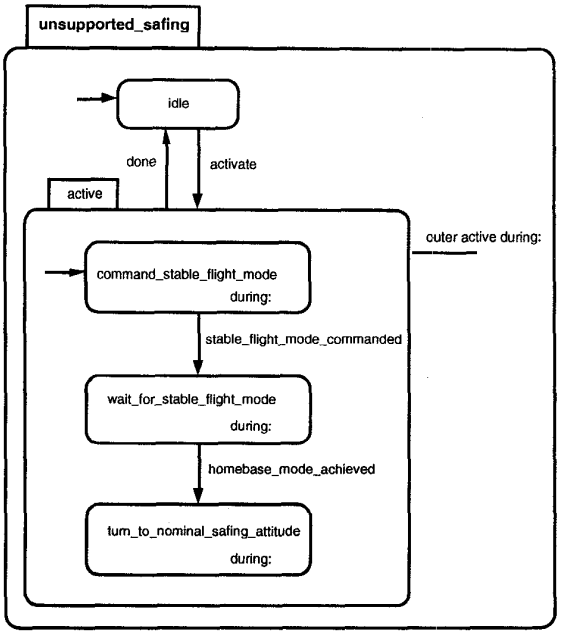

Figure 5:
Response Script State Transition Diagram

This response script declares itself "done" upon completion of the turn to the safing attitude, or upon a resumption of CDS services. Resumption of CDS services is monitored by the script's "outer active during", which is executed during each computational cycle that the script is active.

A response script can also be activated by another response script, or by a ground command. Multiple response scripts may run concurrently. Resource competitions between response scripts are managed directly at the point of conflict. For example, the propulsion system control software will not allow more than one latch valve to be actuated during a single computation cycle. If two response scripts attempt to actuate two latch valves in the same computation cycle, the second actuation request will be queued by the propulsion system control software, and held until the next computation cycle.

There are a total of 213 response scripts in the ACS flight software.

Repair Manager Architecture

A repair manager performs corrective actions on a particular physical assembly, based on a record of its prior attempts to repair that assembly. A response script that addresses a hardware failure can direct a repair manager as part of its response. There are eight different types of repair managers in the ACS fault protection, one for each of the fundamentally different assembly types.

One type of repair manager is the Remote Terminal repair manager. There are sixteen such repair managers, one for each assembly that communicates on the on the ACS bus. A Remote Terminal repair manager can perform three types of operations: "get", "isolate" and "repair". A response script must tell the repair manager which to do. A "get" operation is essentially a power cycle, first powering the assembly off and then on. An "isolate" operation will power off the assembly. A Remote Terminal repair manager will only attempt a "get" or an "isolate" operation three times before it becomes exhausted, refusing to attempt these operations anymore. A Remote Terminal repair manager's first "repair" operation is to reset the remote terminal. If this is not successful and a second "repair" is requested, the repair manager will switch all remote terminal communications to the redundant ACS Bus. At this point, the Remote Terminal repair manager becomes exhausted and refuses to attempt any more repair operations.

Another type of repair manager is the Latch Valve repair manager. There are six such repair managers, one for each of the propulsion system latch valves that need to be autonomously opened and closed by the ACS. A Latch Valve repair manager can perform three types of operations: "open", "close", and "suspend". The first two operations are self-explanatory. The "suspend" operation is used only in the case where continuous power is being applied to the latch valve by its drive electronics. This condition could overheat the latch valve. The overall fault protection response is to remove power from the drive electronics, and to suspend the repair manager. A latch valve whose repair manager has been suspended can not be opened or closed by fault protection until the repair manager has been reset.

The other six repair manager types are the repair managers for the propulsion valve drive electronics, the engine gimbal electronics, the SRU heaters, SRU temperature control, the catalyst bed heaters and the bus controller. Each of these assemblies have slightly different attributes that require slight differences in their repair manager operations.

USER INTERFACES TO THE ACS FAULT PROTECTION

Behavioral Controls

Operability concerns were addressed in the design of the ACS fault protection algorithms. Although very little "care and feeding" of these algorithms is required, the operations team can choose to change the behavior of these algorithms at any of several levels. Behavioral controls include:

Error Monitor Masks. A command can be used to mask or unmask any single error monitor. The color of a masked error monitor appears "Black" to all the Activation Rules in which it is used. Groups of commands can be used to assert any desired combination of error monitor masks.

Activation Rule Masks. A command can be used to mask or unmask any single activation rule. A masked activation rule cannot activate a response script. Groups of commands can be used to assert any desired combination of activation rule masks.

Named Parameters. Special parameter update commands were created for those fault protection parameters (or groups of parameters) that are expected to require an update at some point during the mission. For each named group of parameters, the user simply specifies the new values, and the software finds the locations of these parameters in memory. One example is the command that updates the thresholds for the "excessive thruster firing" error monitors (see Reference 11 for a detailed description of these error monitors). These parameters will need to be loosened during close encounters with the Earth, Venus, and Titan, when the disturbance torques on the spacecraft will be hundreds of times greater than what is normally acceptable.

Unnamed Parameters. Any fault protection parameter can be updated by a generic user command that provides a new value for a specific address. Parameters that can be changed this way include:

- Filter values for error monitors
- Threshold criteria for error monitors
- Persistence criteria for error monitors
- Time-out values for response scripts
- Retry limits for response scripts

Health States. A user command can be used to change the health state of any assembly from "ok" to either "failed" or "dead". A "failed" assembly cannot be made prime, and a "dead" assembly cannot be powered on. The fault protection algorithms will not change the user-controlled health states, and will always check the health

state of a redundant assembly before attempting to bring that assembly online as a replacement for an apparently failing prime assembly. If a redundant assembly has already been declared "failed" or "dead", the algorithms will immediately move on to other corrective actions involving other assemblies that are still healthy.

Real-Time Telemetry

The Cassini spacecraft is rarely monitored by the operations team, and even then then, its telemetry is several minutes old before it reaches the Earth. Therefore, the real-time ACS fault protection telemetry has been designed to provide the operations team with the best possible insight into what *has happened* in the past. There are also a few summary indicators of what is currently happening. Key items in the real-time ACS fault protection telemetry include:

High Water Marks. These contain the maximum deviations and persistences of certain monitored error variables. These give the operations team some insight into transient error monitor activity that hasn't been severe enough (yet) to activate any fault responses. These also give the operations team some insight into how bad the errors got or how long they lasted after corrective action was initiated.

Last Error. This is a simple one-deep buffer, identifying and time-tagging the last Red transition by an error monitor (i.e. the last time the color of an error monitor changed from not Red to Red).

Error Count. This is the total count of the number of Red transitions by all of the error monitors.

Last Error Response. This is a simple one-deep buffer, identifying and time-tagging the last Response Script activation (i.e. the last time a Response Script transitioned from its Idle state to its Active state).

Active Response Count. This indicates the number of Response Scripts that are currently active.

Event Log

A detailed accounting of all significant ACS events, including error declarations and response milestones, is contained in the Event Log. Each AFC keeps an Event Log in a reserved section of its Direct Access Unit (DAU) memory, which is the memory that it uses to interface with the support equipment during ground testing. Since DAU memory is not crucial to in-flight AFC operations, it does not need to be initialized following a warmboot AFC reset. This allows the Event Log to survive a warmboot reset, capturing significant events across one or more resets.

Each Event Log entry is a fixed-length record of seven 16-bit words. As shown in Figure 6, each entry contains a 32-bit time-tag, providing a time-tag resolution of 125 msec. The third word identifies the event type. The last four words contain type-specific data for each event type. The event types and their associated data are:

- Mode Change, for which the associated data identifies the old mode and the new mode.
- Red Error Monitor, for which the associated data identifies which error monitor became Red.
- Response Script Command, for which the associated data identifies which Response Script issued the command, and what that command was.
- Repair Manager Command, for which the associated data identifies which Repair Manager issued the command, and what that command was.

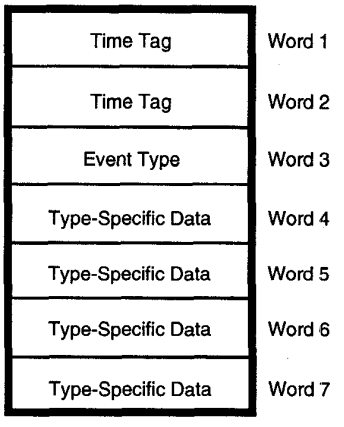

Figure 6: Event Log Entry Format

The Event Log is currently sized to hold 1750 entries. After 1750 entries, the pointer automatically returns to the 151st entry, thereby preserving the first 150 entries. A special user command allows the operations team to reset the pointer back to the starting address.

The contents of the Event Log are slowly trickled out through the real-time telemetry. Users can also downlink any portion of the Event Log more quickly via a special memory readout command.

Post-Mortem Data

Post-Mortem Data provides a detailed snapshot of the state of the AFC and the flight software executive just prior to the last warmboot AFC reset. The flight software updates the Post-Mortem data in all of its fatal error interrupt handlers, and also updates the Post-Mortem

Data in the brief period it is given prior to a hardware-asserted warmboot AFC reset. Similar to the above-described Event Log, each AFC keeps its Post-Mortem Data in a reserved section of DAU memory, allowing this data to survive a warmboot AFC reset.

Post-Mortem Data is a single fixed-length record of 176 16-bit words. This data includes:

- The identity of the last error interrupt
- The time of the last error interrupt
- The total number of error interrupts since reset
- Total number of resets
- The contents of error registers
- The contents of chip configuration registers
- The time when the post-mortem data was written

Real-time telemetry includes similar information on the AFC's *current* status, but it does not include the actual Post-Mortem Data snapshot at the time of the last reset. Users can retrieve any portion of the Post-Mortem Data via a special memory readout command.

ACS PARTICIPATION IN SYSTEM-LEVEL FAULT PROTECTION

The ACS participates in several System-level fault responses that are coordinated by the CDS prime string. For detailed information on Cassini's System-level fault protection requirements and design, see Reference 7. The interfaces between the ACS fault protection algorithms and the System-level fault protection algorithms are shown in Figure 7. Bold arrows identify the messages that are passed between the subsystems via the CDS Bus. Light arrows show System-level responses that are activated by other System-level responses within the CDS. As shown in Figure 7, the System-level fault responses send commands to one or both AFCs, using the same command formats and protocols that are used by the normal, ground-provided stored sequences.

Spacecraft Safing is the general-purpose System-level response that transitions the spacecraft from an arbitrary initial state to a thermally safe, commandable, low activity state. As shown in Figure 7, most of the other System-level responses issue particular commands that address particular failures, and then activate the Spacecraft Safing response.

Heartbeat Messages

Each AFC generates a heartbeat message during each computation cycle. Each heartbeat message indicates whether that AFC is operating its flight software program, whether that AFC believes it is the prime or backup AFC, and whether that AFC believes it is healthy enough to serve as the prime AFC. A counter is incremented in each heartbeat message, in order to prove that the message is fresh.

As shown in Figure 7, a system-level fault protection algorithm examines the heartbeat messages, and initiates corrective action if it sees heartbeat messages indicating anything other than one healthy prime AFC. If the prime AFC stops providing fresh heartbeat messages, or if its heartbeat messages indicate that it does not wish to remain prime, the CDS will issue commands to reset that AFC and reload its flight software, or to assert the backup AFC as the new prime. The CDS is also responsible for correcting the unforeseen condition where both AFCs believe they are prime at the same time.

Alert Messages

Figure 7 also shows that the ACS fault protection algorithms in the prime AFC can activate certain System-level responses by issuing appropriate "alert messages" to the CDS. These alert messages are:

CDS Loss Alert Message. The prime AFC issues this alert message when it detects a loss of the regularly expected CDS Bus transmissions. If the CDS is still operational and is able to collect this alert message, the CDS will assume that the communications problem is on the ACS side of the interface, and will activate its Heartbeat Loss response in an attempt to restore services to the prime AFC. Note that if the CDS is not operational or is unable to collect this alert message, then ACS will eventually activate an ACS response that is appropriate for a long-term CDS failure. This ACS response is described in a later section of this paper, "ACS Responses to Failures in Other Subsystems".

Engine B Pyro Alert Message. The prime AFC issues this alert message when it wants the CDS to open the pyro valves that normally keep Engine B isolated from its bi-propellants. This is an irreversible action that will only be requested for a second attempt at the SOI burn. For more information on SOI redundancy management and multiple SOI burn attempts, see Reference 9.

Spacecraft Safing Alert Message. The prime AFC issues this alert message when it detects an anomaly that could compromise the objectives of the stored sequence, or when it observes excursions of System-level performance metrics beyond agreed-upon safe limits.

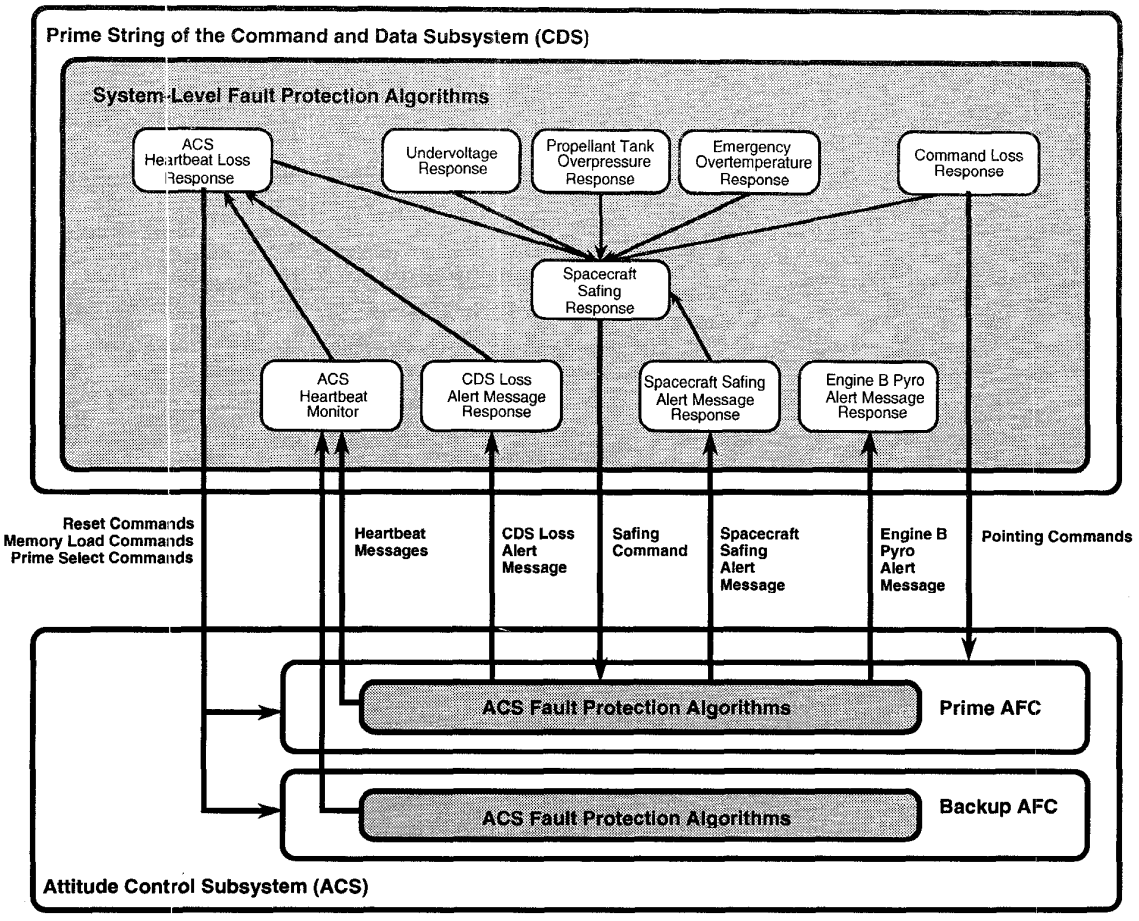

Figure 7: ACS Interfaces with the System-Level Fault Protection Algorithms

Specific criteria for issuing the Spacecraft Safing alert message include:

- Temporary loss of J2000 attitude estimate
- Temporary loss of attitude control capability
- Attitude control error has exceeded threshold
- ACS power consumption needs to be autonomously increased by more than 17 watts
- ACS power consumption has been autonomously increased by any amount for more than three minutes
- The SOI burn has been terminated prematurely
- A critical sequence command has been rejected

The CDS responds to a Spacecraft Safing Alert Message by activating the Spacecraft Safing Response. After quickly terminating all stored sequences and shedding all non-essential spacecraft loads, the Spacecraft Safing Response issues a special-purpose safing command back to the ACS. The ACS response to the safing command, termed the ACS Safing Response, is described in the following section.

ACS Safing Response Overview

The ACS Safing Response has three primary objectives:

1) Put the propulsion system in a known, benign state
2) Put the attitude control system in a known, low power, low activity state
3) Turn the spacecraft to a thermally safe attitude that will support Earth communications, given the system-directed safe state of the RFS.

Although there is a final state that completely satisfies these objectives, there are some conflicts among these objectives along the path to that final state. Where conflicts arise, the ACS Safing Response prioritizes the objectives as shown above.

ACS safes the propulsion system by:

- closing a latch valve to isolate the bi-propellant tanks from the high-pressure helium supply.

- closing four latch valves to isolate both of the main engines from both of the bi-propellant tanks.
- closing a latch valve to isolate the backup thruster branch from the hydrazine tank.

All of the above-mentioned latch valves are enforced closed, regardless of their positions at the time the ACS Safing Response is executed. These latch valves are enforced closed one by one, by temporarily applying power to the normally unpowered latch valve drive electronics. These valve closures require momentary increases in the ACS power consumption.

The ACS Safing Response enforces the following low power ACS configuration:

- prime celestial sensors (SSA, SRU) on
- backup celestial sensors off
- prime IRU on
- backup IRU unchanged
- prime thruster drive electronics on
- backup thruster drive electronics off
- all latch valve drive electronics off
- all engine valve drive electronics off
- all engine gimbal drive electronics off
- all reaction wheels off
- accelerometer off

Although the ACS Safing Response immediately begins the transition to this ACS configuration, it won't power off the latch valve drive electronics until it has finished the above-mentioned safing of the propulsion system. The ACS Safing Response will also keep the reaction wheels powered until the prime thrusters are ready to use (note: the prime thruster catalyst beds may need to be "warmed up" if the reaction wheels were being used for attitude control at the time that the ACS Safing Response was activated).

After the ACS Safing Response has safed the propulsion system and transitioned the ACS to its low power configuration, it turns the spacecraft to a ground-prescribed safing attitude. The operations team is responsible for assuring that the ACS safing attitude is appropriate for the current mission phase, and is compatible with the RFS configuration that will be asserted by the Spacecraft Safing Response. The safing attitude is described the same way that the users normally specify a desired attitude (see Reference 5), which is a command to point a body-fixed axis at a target body whose predicted motion is propagated onboard. The expected safing attitude for the entire mission will be the attitude that points the -Z axis (the HGA and LGA-1) at the Sun, while also minimizing the angle between LGA-2 and the Earth. At this attitude, at least one of the LGAs should always be able to provide at least 10 bps downlink.

ACS RESPONSES TO ACS FAILURES

Figure 8 is a high-level "roadmap", indicating how the ACS activation rules and response scripts are organized to respond to ACS failures. In this view, there are five groups of algorithms. Four of these groups address failures that can be directly attributed to ACS assemblies. The fifth group addresses failures of ACS functions that cannot be directly attributed to any particular assembly, at least not immediately.

The following subsections describe the behavior of each algorithm group.

ACS Flight Computer Failures

If the ACS flight software is able to effectively respond to a fatal error interrupt, it will immediately initiate a full reset of that AFC. If an AFC failure brings the flight software to a halt, then the AFC will be reset by a hardware-asserted response to the expiration of a 2.2 second watchdog timer. After an AFC has been reset, its Start-Up ROM (SUROM) program will perform a variety of self-tests and attempt to re-establish communications between that AFC and the CDS.

The CDS expects to receive fresh, valid heartbeat messages from the prime AFC, as described in a previous section of this paper. If the prime AFC stops communicating with the CDS or reports that it has reset and is now operating its SUROM program, the CDS will issue commands to re-establish that AFC as the prime AFC. If these attempts are repeatedly unsuccessful, the CDS will power on the backup AFC and establish it as the new prime AFC. For more information on the ACS Start-Up ROM program and its interactions with the CDS, see Reference 8.

If the ACS flight software has not detected any fatal error interrupts, but has reason to believe that the host AFC is no longer healthy, it can invite the CDS to reset its host AFC by sabotaging its heartbeat messages. A one-word "healthbeat" field of the heartbeat message is reserved for this purpose; if the ACS flight software sets the "healthbeat" field to anything other than its expected value, the CDS will treat this as an invalid heartbeat message, and will activate its Heartbeat Loss response.

ACS Bus Failures

If two or more ACS peripheral assemblies are unable to provide error-free communications on the ACS Bus, the flight software will step through the following progression of responses until error-free communications are restored:

1) Reset the AFC's Bus Controller subassembly
2) Switch to the redundant ACS Bus for all ACS Bus transactions to all the peripheral assemblies
3) Remove power from all the peripheral assemblies that are not performing essential functions.
4) One by one, replace an essential peripheral assembly with its backup, then remove power from the old prime assembly.

The first two responses address possible failures of the common hardware that is used in all ACS Bus transactions. The last two responses address possible failures that would cause a peripheral assembly to respond to one or more ACS Bus addresses that are not its own.

As shown in Figure 8, if all of these responses are repeatedly unsuccessful, the flight software will sabotage its heartbeats, as previously described in the section on AFC failure responses. At this point, the hypothesis is that persistent ACS bus transaction failures are being caused by a subtle, undiagnosed failure of the AFC.

Vital Assembly Failures

A vital assembly is a peripheral assembly whose function must be autonomously restored, given the ACS goals and the ACS usage of that assembly at the time of the failure.

An assembly must be a prime assembly in order to be considered vital. Some prime assemblies (e.g. the prime VDE control unit) are vital in all the ACS modes. Other prime assemblies (e.g. the prime SRU) are only vital in a subset of the ACS modes. Still other prime assemblies (e.g. the prime RWAs) are never vital.

If a vital assembly fails to perform within the acceptable limits of its error monitors, the flight software will step through the following progression of responses until acceptable performance is restored:

1) Reset that assembly
2) Switch to the redundant ACS Bus for all ACS Bus transactions to that assembly
3) Replace that assembly with its backup, then remove power from that assembly

As shown in Figure 8, if all of these responses are repeatedly unsuccessful, the flight software will activate the previously-described ACS Bus failure response. At this point, the hypothesis is that persistent failures of both a prime assembly and its backup are being caused by a subtle, undiagnosed failure of the communications path.

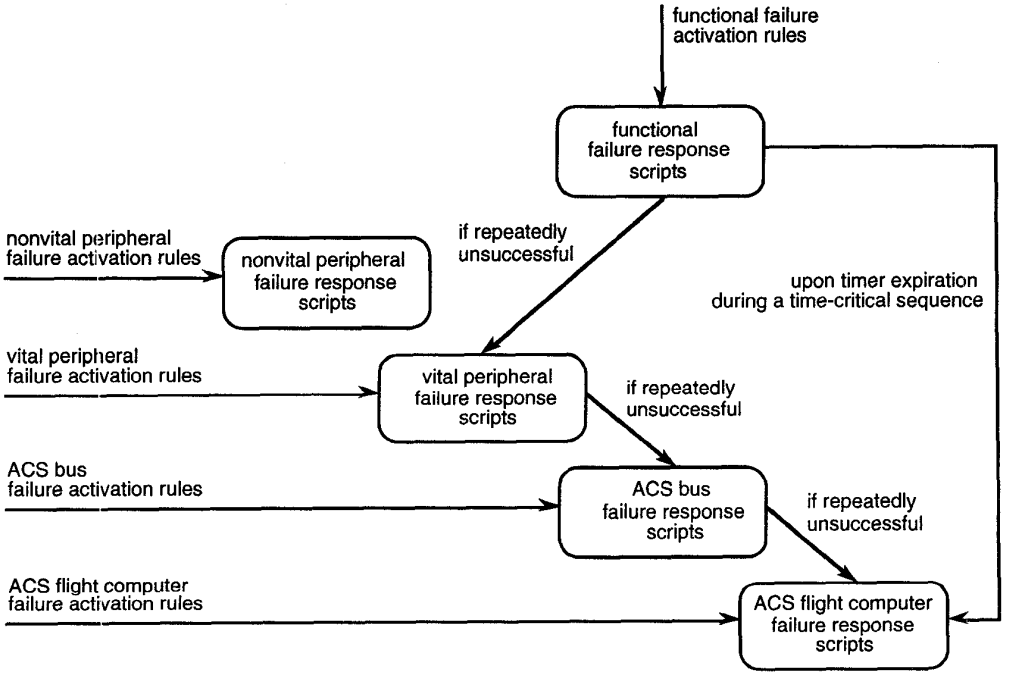

Figure 8: Roadmap of the ACS Responses to ACS Failures

Nonvital Assembly Failures

A nonvital assembly is an assembly whose function does not need to be autonomously restored, given the ACS goals and the ACS usage of that assembly at the time of the failure.

Backup assemblies are always considered nonvital. If a nonvital backup assembly fails to perform within the acceptable limits of its error monitors, the flight software will step through the following progression of responses:

1) Reset that assembly
2) Switch to the redundant ACS Bus for all ACS Bus transactions to that assembly
3) Remove power from that assembly

If any of the autonomously monitored symptoms indicate that the failure is imminently hazardous, the flight software will proceed immediately to step 3 in order to quickly remove power from the failed assembly.

Some of the prime assemblies are also considered nonvital. For instance, the prime RWAs are nonvital, since the flight software can respond to a prime RWA failure by transitioning to a thruster-based control mode. The flight software's response to a failed nonvital prime assembly is similar to the above-described progression for a nonvital backup assembly; however, the flight software cannot remove power from a prime assembly until it has transitioned ACS to a mode in which that assembly is no longer required for control or estimation.

As shown in Figure 8, unsuccessful responses to nonvital assembly failures will not lead to the activation of other, more drastic responses. If a nonvital assembly has failed and cannot be turned off, the flight software will simply leave it on and cease responding to its failure symptoms. From a subsystem perspective, this is fine as long as the failed assembly does not corrupt ACS Bus transactions to any of the other peripherals.

Functional Failures

A functional failure is a failure of the subsystem to perform a required function within a required time to a required level of accuracy. Examples of ACS functional failures include:

- Failure to acquire the Sun within the allotted time
- Failure to acquire stellar reference within the allotted time
- Failure to keep the attitude control error below a prescribed threshold
- Failure to keep the attitude knowledge uncertainty below a prescribed threshold

Functional failures can be caused by assembly failures, but can also be caused by operator errors, unexpected environmental conditions, and software design errors. Therefore the autonomous responses to functional failures have been carefully constructed to address all of these possibilities in a safe and logical manner.

Although each of the functional failure responses has some unique features, they all share one of two common approaches:

1) If the failing function is a nonessential function, the software will simply suspend that function, and retreat to a mode in which that function is not needed. One example of this approach is the response to an unacceptably large attitude control error in reaction wheel control mode. If this occurs, the software will simply discontinue reaction wheel control and transition to its thruster control mode. The exact cause of the reaction wheel control failure, and any desired corrective actions, are left for the ground to decide.
2) If the failing function is an essential function, the software will temporarily terminate that function, temporarily retreat to an intermediate state that does not require that function, and then reinitialize and retry the entire function. One example of this approach is the response to an unacceptably large attitude control error in the thruster control mode. If this occurs, the software will briefly disable the thruster-based attitude controller, and then restart the controller from scratch. The thruster-based attitude controller will begin by simply trying to regain control of the spacecraft rates, and then will proceed back to three-axis attitude control.

As shown in Figure 8, unsuccessful responses to functional failures eventually cause the flight software to activate one or more of the previously-described responses to vital assembly failures. At this point, the hypothesis is that the function is repeatedly failing because of a subtle, undiagnosed failure of one of the assemblies that supports that function.

If a functional failure occurs during a time-critical sequence such as SOI, there may not be enough time for the normal progression of responses to address all the possible assembly failures. Figure 8 shows that if an essential function remains failed for some prescribed time during a time-critical sequence, the flight software will sabotage its heartbeat messages, activating the system-level Heartbeat Loss response (described in the previous section on AFC failure responses). At this point, the Heartbeat Loss response, though drastic, probably provides the best chance for mission success.

ACS RESPONSES TO FAILURES IN OTHER SUBSYSTEMS

As outlined previously, the Cassini ACS is dependent on several other avionics subsystems for power, status information, and directions. Failures of other subsystems could cause an interruption or loss of one or more interface services. The ACS design helps contain the effects of failures in other subsystems by tolerating:

Interruption of Power. All of Cassini's engineering subsystems, including the ACS, are required to tolerate infrequent power outages of up to 37 msec in duration. There are no credible faults in the power supply and distribution system that cause longer-duration interruptions. The Cassini ACS meets this requirement via an integrated hardware and software solution:

1) Capacitive power storage in each AFC's power supply keeps the RAM flight software program intact across a brief power outage.

2) The ACS SUROM program recognizes the difference between a "warmboot" (e.g. from a brief power interruption or a flight-software-initiated reset) and a "coldboot" (e.g. from a previously unpowered state). The SUROM responds to a "coldboot" by performing a CDS-assisted memory load, but can autonomously respond to a "warmboot" by quickly validating and transferring execution back to its unaffected RAM program. For more information on the ACS SUROM program, see Reference 8.

Loss of Status Information. Other subsystems provide ACS with power switch status, bi-propellant engine temperatures and chamber pressures via the CDS-controlled Command and Data Bus. The CDS also provides ACS with spacecraft time and synchronization signals. ACS is required to tolerate loss or interruption of all this information, which could occur during autonomous CDS responses to failures affecting the Command and Data Bus. ACS employs either of two primary strategies, depending on the criticality of the information to the subsystem:

1) Some information, such as the spacecraft time, is essential to ACS flight software processing; in order to tolerate loss of this data, ACS must provide a backup source. In the absence of fresh valid time transmissions from the CDS, ACS propagates its own internal time, and uses that time instead. When and if fresh valid time transmissions resume, the ACS calculates the difference between its locally-propagated time and the new CDS-transmitted time. If a significant discontinuity exists, ACS will briefly suspend all time-sensitive activities, and then synchronize to the new time.

2) Other information, such as the engine temperature and pressure measurements, are used only for fault detection purposes. This information is not essential to the ACS, and therefore a backup source is not required. In the absence of fresh valid engine temperature and pressure measurements, ACS immediately sets the output colors of the affected error monitors to black (i.e. "no opinion"), and continues to evaluate the engine performance using the remaining available information. For instance, loss of an engine's chamber pressure measurement would prevent that measurement from contributing to an engine failure diagnosis, but would not prevent ACS from arriving at the same diagnosis via accelerometer measurements.

Corruption of Status Information. All incoming status information is tested for freshness using either an explicitly provided time-tag, or a monotonically increasing message counter. All critical information is also checksummed.

Long-Term Failure of the CDS. Although the Cassini CDS is a critical engineering subsystem that must be fault tolerant (see Reference 2), the ACS has been asked to tolerate a long-term absence of CDS services, and to respond by establishing a thermally safe, commandable attitude that can support ground-directed CDS recovery activities. The ACS meets this requirement by monitoring the length of time that certain fundamental CDS Bus transactions (such as the spacecraft time broadcast) have been absent, and then activating the unsupported_safing Response Script. The unsupported_safing Response Script was described earlier in this paper, as an example of the Response Script architecture. It recognizes that the CDS is not available to coordinate any ACS-requested changes in the ACS hardware configuration, and so it simply initiates a turn to an internally-stored "default" safing attitude using the currently available sensors and actuators.

SUMMARY

The Cassini ACS fault protection algorithms, and the interfaces between these algorithms and Cassini's system-level fault protection algorithms, allows the Cassini ACS to gracefully tolerate ACS assembly failures, ACS functional failures, and even failures in other subsystems. The architecture of these algorithms provides significant performance and operability improvements over the fault protection algorithms that have flown on previous interplanetary spacecraft.

ACKNOWLEDGMENT

The Cassini ACS fault protection algorithms were developed over a five year period during which many individuals made significant contributions in the areas of requirements definition, conceptual design, detailed design, flight software development, and testing. These individuals included: E. Attanasio, L. Bagby, D. Bernard, M. Brown, K. Clark, J. Hackney, K. Hilbert, S. Johnson, J. Kim, D. Lam, M. Lam, A. Lee, G. Macala, R. Manning, R. Rasmussen, G. Singh, J. Walker, G. Watney, M. Wette, and K. Zarnegar.

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

1) J.C. Hackney, D. E. Bernard, and R. D. Rasmussen, "The Cassini Spacecraft: Object Oriented Flight Control Software", 16th AAS Guidance and Control Conference, February 1993.

2) T. K. Brown and J. A. Donaldson, "Fault Protection Design of the Command and Data Subsystem on the Cassini Spacecraft", 13th Digital Avionics Systems Conference, October 1994.

3) R. D. Rasmussen, "Spacecraft Electronics Design for Radiation Tolerance", Proceedings of the IEEE, Vol. 76, No. 11, November 1988.

4) R. M. Manning, "Low Cost Spacecraft Computers: Oxymoron or Future Trend?", 16th AAS Guidance and Control Conference, February 1993.

5) R. D. Rasmussen et al, "Behavioral Model Pointing on Cassini Using Target Vectors", 18th AAS Guidance and Control Conference, February 1995.

6) G. M. Brown, D.E. Bernard, and R. D. Rasmussen, "Attitude and Articulation Control for the Cassini Spacecraft: A Fault Tolerance Overview", 14th Digital Avionics Systems Conference, November 1995.

7) J. P. Slonski, "System Fault Protection Design for the Cassini Spacecraft", 1996 IEEE Aerospace Applications Conference.

8) G. M. Brown, J.C. Hackney, R. D. Rasmussen, and K. Zarnegar, "Storing and Loading the Flight Software for Cassini's Attitude and Articulation Control Subsystem: A Fault Tolerant Approach", 15th Digital Avionics Systems Conference, November 1996.

9) D. L Gray and G. M. Brown, "Fault Tolerant Guidance Algorithms for Cassini's Saturn Orbit Insertion Burn", 1998 American Control Conference, June 1998.

10) G. A. Macala, "A State-Space Fault Monitor Architecture and Its Application to the Cassini Spacecraft", 1998 American Control Conference, June 1998.

11) A. Y. Lee, " A Model-Based Thruster Leakage Monitor for the Cassini Spacecraft", 1998 American Control Conference, June 1998.