N90-27299

Attitude Determination and Control System (ADCS) Maintenance and Diagnostic System (MDS)

A Maintenance and Diagnostic System for Space Station Freedom

David Toms and George D. Hadden
Honeywell Systems and Research Center
3660 Technology Drive
Minneapolis, Minnesota 55418

Jim Harrington
Honeywell Space Systems Operation
13350 US Highway 19 South
Clearwater, Florida 34624

Abstract

This paper describes the Maintenance and Diagnostic System (MDS) that is being developed at Honeywell* to enhance the Fault Detection Isolation and Recovery system (FDIR) for the Attitude Determination and Control System on Space Station Freedom. The MDS demonstrates ways that AI-based techniques can be used to improve the maintainability and safety of the Station by helping to resolve fault anomalies that cannot be fully determined by built-in-test, by providing predictive maintenance capabilities, and by providing expert maintenance assistance.

The MDS will address the problems associated with reasoning about dynamic, continuous information versus only about static data, the concerns of porting software based on AI techniques to embedded targets, and the difficulties associated with real-time response.

We have built an initial prototype of this MDS and are continuing development on it. The prototype executes on Sun and IBM PS/2 hardware and is implemented in the Common Lisp; further work will evaluate its functionality and develop mechanisms to port the code to Ada.

Introduction

The systems on Space Station Freedom (SSF) will require high levels of reliability, fault tolerance, and ease of maintainability. Our ability to satisfy these requirements can be enhanced by using sophisticated software techniques to further automate tasks. Many of the software techniques appropriate in this area grew out of, or are being developed by, research in artificial intelligence.

This automation is particularly useful for *remote* systems on SSF. Remote systems are defined as systems physically separated from the crew quarters. The Attitude Determination and Control System (ADCS) is one such remote system. All of the ADCS hardware, except for the Standard Data Processor (SDP), resides on a section of truss assembly approximately 25m from the habitable modules. The only means of repairing the ADCS hardware is for a crew member to perform extravehicular activity (EVA) or to use one of the SSF robot systems, such as the Flight Telerobotic Servicers (FTS) (not as dangerous as going EVA, but time-consuming and tedious).

Since the ADCS is a remote system, causing maintenance to be expensive, it becomes important to be able to reliably determine which ADCS element is faulty and to be able to predict when a failure might occur (or that a component is degrading). Accurate fault information is required to plan a maintenance action and optimize the time spent on the maintenance action.

* Work on the MDS is funded by NASA contract NAS 9-18200, through McDonnell Douglas Space Systems Company contract #87916007. The McDonnell Douglas Technical Monitor is Gerry Ridout.

The Maintenance and Diagnostic System (MDS) could provide additional information to the Fault Detection, Isolation, and Recovery (FDIR) system to aid in the fault isolation process. The MDS derives this additional information by reasoning about data in a global way; this is not possible with built in test (BIT) within an individual ADCS Orbital Replaceable Unit (ORU). The MDS software will collect, store, and analyze the health monitoring data from each active ADCS element to provide additional information for fault analysis. In addition to fault isolation, it is expected that the MDS will also be able to verify and flag ADCS BIT false alarms.

One of the goals of the MDS is to predict when a fault might occur. Fault prediction is a difficult problem to solve. However, most of the components within the ADCS are modifications of standard hardware, and a significant database of failure modes exists. The successful implementation of predictive capabilities in the MDS would provide significant benefits:

- Additional information is gained, leading to the identification of a faulty element (thus reducing the time and cost of fault isolation).
- Maintenance actions can be planned for a time when a crew member is "in the neighborhood" (thus reducing maintenance cost).
- Replacing or repairing the system before it malfunctions could prevent further damage to other systems as well as providing more reliable systems operation.
- Safety could be enhanced by predicting and preventing problems that would have occurred during a critical maneuver such as berthing.

Once a fault has been isolated, the MDS can be used as a maintenance aid, to instruct crew members on test and replacement procedures.

This paper discusses the overall architecture of the MDS, the general approach being used to develop each of its capabilities, and specific plans for the prototype currently under development.

ADCS/Space Station Freedom Overview and Maintenance Philosophy

The software for the ADCS will reside in the Guidance Navigation and Control (GN&C) SDP. Besides the software in the GN&C SDP, the ADCS also consists of six Control Moment Gyros (CMGs), three Star Trackers (STs), and three Inertial Sensor Assemblies (ISAs). The ISAs and STs are the sensors supporting the Attitude Determination Function (ADF); the CMGs are part of the

Attitude Control Function (ACF), along with the Reaction Control System (RCS). Functional representation of the ADCS within the GN&C SDP is portrayed in Figure 1.

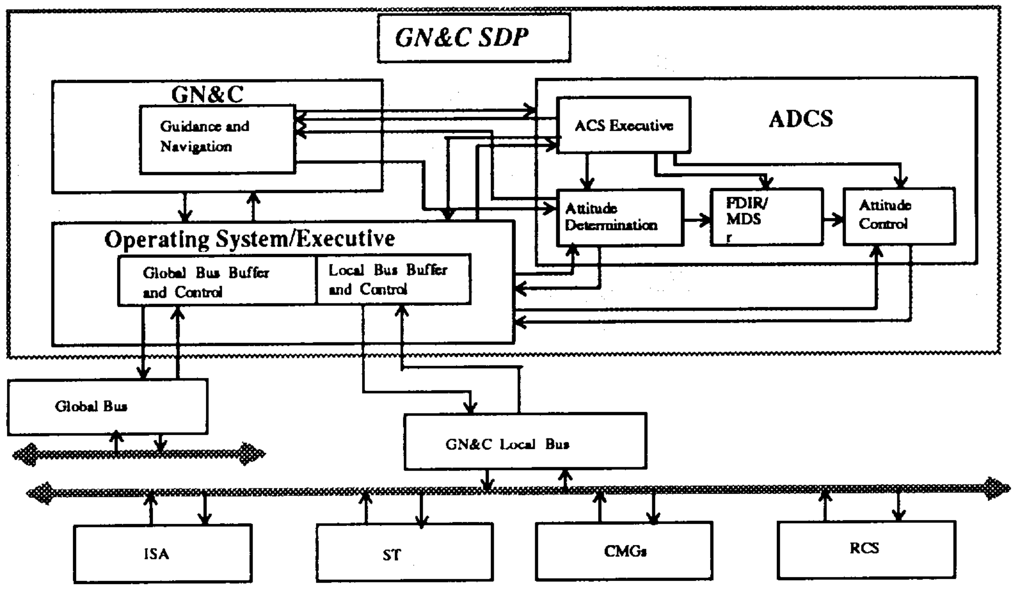


Figure 1. Functional Overview of MDS in ACDS

The minimum configuration of the ADF is the operation of one ISA and one ST; during docking, berthing, and other critical operations, additional ISAs and STs would be brought on line. The primary purpose for running in the minimum configuration is to conserve power.

The primary error checking parameter currently designed for the ADF, besides the BIT, is the chi-squared product (χ^2) formed from the primary Kalman filter used for attitude determination in the ADF. The BIT could indicate that both the ISA and ST are working within performance bounds; however, the χ^2 could indicate significant error in the calculations based on the data returned from the ISA and ST. Without additional information, it is not possible, however, to determine which of the two elements is causing the problem.

Another problem that occurs in a Kalman filter is that one of the units (either the ST or the ISA) is degrading slowly. The Kalman filter can continue to add a bias to the ORU's data that will keep the system (and the χ^2) within established limits masking the degraded performance of the ORU. The coefficients of the χ^2 can be monitored along with additional data generated by the Kalman filter to detect when this condition is arising.

ORUs are defined at a very high level: STs, ISAs, and CMGs are each considered an ORU. The CMG is a special case in that it has two additional ORUs mounted on it (the CMG electrical assemblies). By definition, no diagnosis or repair will be done below the ORU level. We do, however, expect to use the internal structure of the ORUs in our reasoning and not treat them as black boxes.

As one would expect, there are stringent requirements for mean time between failure (MTBF), error detection, and false alarms. BIT is required to detect 70% of the errors in the continuous mode and 95% in the commanded mode. It is required that each system must generate no more than one false alarm every 3000hr. (This, of course, raises questions such as what exactly is considered a system, and whether or not an error in the BIT firmware counts as a failure or a false alarm.)

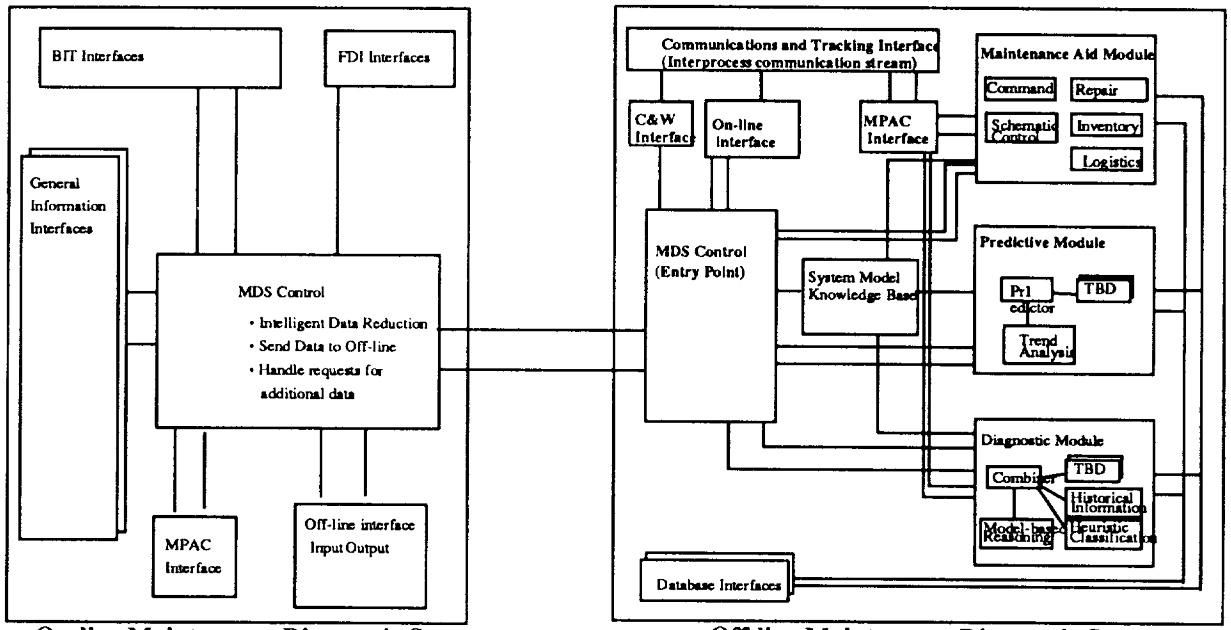
MDS Approach and Top Level Architecture

Our approach to both diagnostics and prognostics uses the concept of a *global view* of the ADCS. By this we mean the ability to view the state of all ORUs in the ADCS, monitor communications, correlate health status from multiple systems, and reason about this information. This global view of the system will give us the capability to address problems that BIT might not handle adequately, much as in the previously mentioned chi-square test.

The MDS is being designed to operate on and with a real-time control system, and will receive a regular "flow" of information from continuous BIT and discrete signals. It has to choose what information to reason about, handle changing information, and know when to look for and request additional information.

This real-time aspect of the MDS implies the need for it to handle real-time concerns such as speed, responsiveness, and timeliness. Further, this data has associated time-values, which complicates the data with temporal concerns such as degraded validity over time, etc.

The MDS needs access to this continuous information flow within the ADCS in order to maintain this global view. However, there are hardware constraints in SSF; memory and processing cycles available on the SDPs will be limited. This combination of requirements and constraints led us to partition the MDS architecture into two basic functional areas. The first is an *on-line* system, embedded within the SDP. The functions of the on-line MDS are intelligent data gathering, data compression, data transmission, and some reasoning. The other is an *off-line* reasoning system. The functions of the off-line MDS are diagnostics, prognostics, and maintenance aiding. In order to obtain the required computing power and mass storage capability it is likely that the off-line MDS would be on the ground. Figure 2 shows a general architectural view of the MDS system.



On-line Maintenance Diagnostic System Off-line Maintenance Diagnostic System

Figure 2. Basic On-line and Off-line Architecture

This partitioning provides the MDS with a natural mechanism to both handle data from a continuously operating real-time process and to reason in depth about the state of the system at any given time. Basically, the on-line MDS handles the real-time monitoring, screening and intelligently screening data generated by the ORUs. The current approach to this uses a dynamic thresholding approach, much as used in [Washington and Hayes-Roth, 1989]. The off-line MDS handles diagnostics, prognostics, and maintenance aiding, described in following sections.

Diagnostics

A diagnostic technique can be evaluated on its breadth and depth of coverage, its robustness and ability to degrade gracefully with ambiguous knowledge, its generality and adaptability to changing system configurations, its efficiency and ease of modification with the target system, and its ability to handle uncertainty. Possible diagnostic techniques include fault trees, explicitly coded fault models, heuristic classification, simulation, functional/causal modeling, and structural modeling.

All these techniques can weigh decisions based on various information sources, including historical information, FMEA (Failure Modes and Effects Analysis), and individual component reliability (predicted MTBF). Discussions of diagnostic approaches can be found in many places, including [Chu, 1988] and [Pau, 1986].

In general, heuristic classification, or "classical" expert systems, have many benefits. A key benefit is that they can capture "experiential" knowledge that is hard to come by and may be hard to characterize in other solutions. They provide an explicit separation of control logic and diagnostic knowledge. This makes them relatively maintainable, and often the diagnostic logic can be re-used. They can be easily extended, which might make them attractive given long-term extensibility goals. And, they provide viable solutions in situations where no model of the system exists.

There are, however, problems associated with the classical, shallow knowledge-based expert system diagnostic approach. First, they require a large amount of knowledge engineering up front to make them robust, because they require large amounts of explicit domain-specific information to solve diagnostic problems. Also, the requirement for explicit knowledge about fault situations makes these systems relatively "fragile" at the edges; if a situation isn't at least partially foreseen, the system could abruptly fail rather than use existing information to degrade gracefully. This reliance on prior knowledge of a system's explicit faults in given situations can cause problems, especially in newly designed systems, or any other system that might not have sufficient explicit fault knowledge immediately available.

Reasoning methods based on connectivity models overcome many of the problems with knowledge engineering and robustness, given that design information will be available with which the model can be built. The major drawback can be that connectivity models fail to characterize all types of fault models completely. This can result in overly ambiguous or incorrect answers in specific types of cases.

Functional qualitative models can alleviate these problems but have higher initial development costs and higher maintenance costs. They are also susceptible to the possibility of computational intractability due to the combinatorial explosive search. Another problem is ensuring the "correctness" of the model, particularly in the cases where good models of the process do not exist (e.g. in some of the ISA failure modes).

We believe that hybrid approaches based on multiple reasoning techniques are the most effective. A particular approach may rely on one dominant, controlling technique and draw on and weight the outputs of other techniques as needed. Hybrid approaches have been used with excellent success in diverse areas. This hybrid diagnostic approach, using heuristic and design-derived fault mode knowledge as it is available, can provide optimal diagnostic capability.

The MDS is being designed with a hybrid architecture, able to combine a variety of reasoning techniques. Using various kinds of knowledge is especially effective in a system that has been designed with both established, proven technologies and new design, such as ADCS. The ISAs, for example, are based on previous designs for which extensive performance information is available, the CMGs are based on a prototype design developed for Marshall Space Center, and the Star Trackers are based on an existing design.

Also, since this architecture can combine an arbitrary number of reasoning methods, it provides an excellent framework to handle extensions.

Prognostics

Earlier in this paper we discussed the reasons for using prognostic reasoning (also referred to as predictive maintenance). Briefly, these are: 1) faster fault isolation, 2) ability to plan maintenance, 3) reduction of the possibility of cascading faults, and 4) enhanced probability of operation (providing greater safety) during critical maneuvers.

Traditionally, predicting device failure has been notoriously difficult, for both mechanical and electronic devices (although mechanical devices are usually easier). The primary reason for this is that it is difficult to correlate the long-term behavior (fault signature) with a particular failure mode.

There are however a few exceptions which our system looks for explicitly. One example we have found is the Ring Laser Gyroscope (RLG) in the ISA. This device has a predictable relationship between input current and output power. There are situations in which monitoring the characteristics of this relationship will allow us to predict future problems with the lasers. Another example is the fault signatures of the CMG wheel bearings. We believe that particular types of vibration patterns might indicate upcoming problems.

The interesting thing from an artificial intelligence point of view is that there is no model for these effects. That is, we cannot evaluate these systems from first principles. Naturally, this makes a model-based approach to this problem relatively difficult. We are in the process of acquiring other predictive maintenance scenarios from experts in the ADCS, but our expectation is that these will be rare and dissimilar.

Our initial approach to prognostics, therefore, is somewhat "brute force." The on-line system has two major functions. The first function is to look specifically for BIT status signatures which might indicate future problems and make appropriate recommendations to the off-line system. The second thing is to send a minimal (but sufficient) set of device status data to the off-line system so that it can do more extensive trending analysis.

Maintenance aiding

Once a failure has been found, or if a problem is predicted, a correction may or may not be in order. The MDS has the context to support access to intelligent, on-line technical information for maintenance aiding, and the initial prototype demonstrates this capability.

One of the ways the MDS provides this access is to present a checklist to the user for subtasks that need to be done to perform the maintenance action. As these are accomplished, the user uses the mouse to check off a box. If there are ordering constraints among the subtasks, the MDS forces them to be met. In other words, the user can not move on to a new subtask unless the ones that must be done before it have been finished. In the initial prototype, the user can also ask for help, at which point a bitmap is displayed.

Another capability that we might add is preventive maintenance (PM) aiding. A PM procedure is usually complex enough that it is necessary to provide the user with more than a simple checklist. Normally, one finds that the procedure can vary quite a bit depending on what is found during the PM [Hadden]. At this point, as far as we know, no PM procedures have been specified for subsystems in SSF, the philosophy being that they will not be necessary. If this continues to be the case, then obviously there will not be a need to provide PM aiding.

In the future, we will extend the initial prototype in many ways. First, we will expand the help options so that the user will be able to control how much help is presented and at what level. We will also allow the user to request explanations for what is being done, explanations both in the sense of "Why is this necessary, given the current goals?" and "How do I perform the requested action?" This latter extension might take the form of a finer-grained action list.

MDS Prototype

We are currently developing a prototype MDS to:

- Test our approaches in an environment "similar" to the fielded environment
- Implement an approach to fit the MDS within the ADCS architecture
- Provide a mechanism to get feedback from engineers on the program, and test diagnostic and prognostic components

The prototype MDS provides the same "look and feel" that a crew member would have on SSF, given the current NASA user interface definitions. These user interface definitions encompass the display, interactions with the crew, and, to some degree, control. Figure 4 shows the basic look of the current prototype.

We expect the final prototype to reside on two machines, one Sun workstation and one IBM PS/2, reflecting off-line and on-line MDS components, respectively, and to communicate directly with the real-time ADCS simulation systems (possibly on 386-class personal computers). Figure 5 shows this configuration.

We chose the IBM PS/2 to run the off-line software to remain consistent with the Software Support Environment (SSE) tools currently under development. The simulations that the prototype will communicate with are being developed to functionally represent the CMGs, STs, and ISAs.

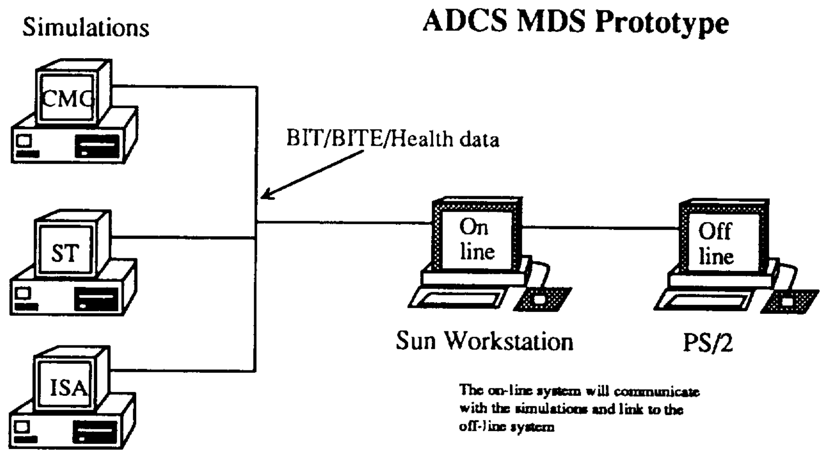


Figure 5. MDS Prototype

The initial demonstration prototype also resides on two machines, but does not yet communicate directly with simulations. Instead, we have simulated the ORUs (or rather the salient outputs of the ORUs) in software on the Sun workstation.

The prototype software for both our Sun and PS/2 environments is primarily implemented in Common Lisp and the Common Lisp Object System (CLOS), which provides us with many existing Lisp tools, the ability to use foreign functions (languages other than Lisp), and standard user interface calls.

This use of object-oriented software was partly driven by long-term goals for SSF for diverse and nearly unconstrained growth while in orbit, which imposes requirements for easy reconfiguration and transparency to continued technological upgrades.

Conclusions and Future Plans

Work on the prototype ADCS MDS has just reached the implementation stage. An infrastructure to simulate continuous ORU output data, an on-line implementation that intelligently screens data, a prognostics capability that acts on known predictive signatures in an individual way, and a maintenance aiding system are currently operational.

This MDS prototype is now being used to incrementally develop the diagnostic approaches and further prognostic capabilities.

Upcoming use and evaluations will determine the specific direction we take, but several areas will definitely be developed in the near future, including:

- Use of the MDS with the high-fidelity ORU simulators. This is a needed step to properly develop and evaluate the system.
- Development of a translation path from Common Lisp to Ada or the standard SSF expert system shell. This is needed to show compatibility with SSF embedded software standards.

In the future, the MDS replacement procedures may be useful in allowing an autonomous robot to perform the maintenance operations with only supervision (in contrast to tele-operation) by a crew member.

References

[1] Murugesan, S., "Considerations in Development of Expert Systems for Real-Time Space Applications" *Fourth Conference on Artificial Intelligence for Space Applications*, November, 1988, pp 487-494

[2] Hadden, George, "Mentor, An Expert System for Preventive Maintenance" Proceedings of the 1986 Annual Conference of the Instrument Society of America

[3] Washington, R. and Hayes-Roth, B., "Input Data Management in Real-Time AI Systems" P*roceedings of the Eleventh Joint International Conference on Artificial Intelligence,* August 1989, pp250-255

[4] Fink, P.K. and Lusth, J.C., "The Integrated Diagnostic Model - Towards a Second Generation Diagnostic Expert System" P*roceedings of the Air Force Workshop on Artificial Intelligence Applications for Integrated Diagnostics*, July 1986, pp188-197

[5] Pau, L.F., "Survey of Expert Systems for Fault Detection, Test Generation and Maintenance" E*xpert Systems*, April 1986, pp100-110

[6] Chu, Sai-Cheong, "Approaches to Automatic Fault Diagnosis: A Critical Evaluation" P*roceedings of the AI in Armaments Workshop*, March 1988

[7] McCown, P. and Lewy, D., "APU MAID - A Diagnostic Expert System Using Heuristic and Causal Reasoning" *AUTOTESTCON '87*, November 1987, pp371-375