N92-14217

GETTING EXPERT SYSTEMS OFF THE GROUND:
Lessons Learned from Integrating Model-based Diagnostics with Prototype Flight Hardware

Amy Stephan
Carol A. Erikson

TRW Space & Technology Group
One Space Park
Redondo Beach, CA 90278

ABSTRACT

As an initial attempt to introduce expert system technology into an onboard environment, a model-based diagnostic system developed using the TRW MARPLE software tool was integrated with prototype flight hardware and its corresponding control software. Because this experiment was designed primarily to test the effectiveness of the model-based reasoning technique used, the expert system ran on a separate hardware platform, and interactions between the control software and the model-based diagnostics were limited. While this project met its objective of demonstrating that model-based reasoning can effectively isolate failures in flight hardware, it also identified the need for an integrated development path for expert system and control software for onboard applications. In developing expert systems that are ready for flight, we must evaluate artificial intelligence techniques to determine whether they offer a real advantage onboard, identify which diagnostic functions should be performed by the expert systems and which are better left to the procedural software, and work closely with both the hardware and the software developers from the beginning of a project to produce a well-designed and thoroughly integrated application.

INTRODUCTION

This paper discusses research at TRW aimed at integrating artificial intelligence (AI) technology into an on-board environment. The work described was a joint effort of the Expert Systems on Spacecraft internal research and development project, spacecraft power system engineers, and flight software developers. The goal of the project was to demonstrate an expert system working in concert with on-board flight software and a hardware testbed. This paper first discusses the nature of AI and flight software, and issues involved in integrating these two technologies. We then describe the work performed at TRW and the results of the expert system tests on the prototype power subsystem. We conclude with a discussion of the problems encountered and lessons learned from this work, and describe a methodology for future integration of AI and onboard data systems.

FLIGHT SOFTWARE AND EXPERT SYSTEMS

The design and development of onboard flight software is driven by the limited capabilities of onboard hardware, the high reliability required to ensure successful operations in space for extended periods, and the deterministic response times inherent in spacecraft control algorithms [Filarey 90]. Required to operate in an extreme environment, onboard data processing systems are constructed of specialized parts designed to survive radiation effects while minimizing size, weight and power consumption. These systems do not provide the throughput and memory capabilities found in ground-based computer systems.

The software developed for onboard systems must also maintain a very high level of reliability, and this places a number of constraints on flight software design and development. Flight software must be designed to be testable; its reliability must be proven on the ground well before launch. The tests performed on flight software range from low-level unit testing, in which every machine instruction and every path of each software module is executed and tested, to spacecraft integration and testing, where the entire spacecraft is assembled and tested. To ensure that the software can be properly tested, its design must be deterministic: given a set of inputs, one must be able to pre-determine both the required output and the path the software will take to produce that output. Most spacecraft control algorithms involve sampling spacecraft sensor data, analyzing the data and sending commands to spacecraft units to maintain stabilized control. The performance of these algorithms depends on accurate scheduling based on known delays between sensor samples and commanded responses. Given these constraints, onboard flight software is limited to those spacecraft functions which are deterministic, efficient in memory use, executable within a guaranteed time interval, and testable.

Typical candidates for onboard flight software include basic spacecraft support functions such as attitude control, thermal control and power management, and spacecraft fault detection and management. Onboard fault detection is limited to a set of predetermined faults which can be diagnosed by a simple analysis of available spacecraft sensor data. While in some cases the flight software may be able to isolate the source of a fault and switch to a redundant unit or alternative control scheme, often the flight software reacts to an anomaly by placing the spacecraft in a non-operational "safe-hold" mode, relying on the ground to isolate and correct the fault. Ground operators also control all mission planning and operations, often through detailed, low-level command sequences. As spacecraft missions require greater

PRECEDING PAGE BLANK NOT FILMED

survivability, autonomy and complexity, this heavy reliance on ground support must be alleviated by software that can perform higher-level decision making [Fesq 89].

Research in the field of AI has sought to increase the capabilities of on-board software in the areas of diagnosis, planning and scheduling. The potential benefits of such research are many. Onboard diagnostic and planning would allow spacecraft to achieve a high level of autonomy, operating for months without ground contact. More complex in-flight navigation could be achieved with little ground control. Large systems such as the Space Station Freedom could make better use of available resources. In addition to increasing the satellite's on-orbit capabilities, enhanced onboard software could significantly reduce the cost of ground operations.

While AI has made much progress in the years since rule-based expert systems first became popular, these advances have not generated a great deal of interest among flight software developers. This results partially from the fact that AI researchers are seldom the same people who develop flight software, and the two groups have markedly different approaches to software development. AI systems are almost always built on specialized workstations where memory and throughput limits seldom impact performance. Even more importantly, AI systems typically are designed to exhibit novel behavior and respond to uncertain conditions. Rule-based systems, for example, are designed to be able to chain through rules in unexpected ways, and often several paths may exist between a set of input and output data. By their nature such systems are non-deterministic and difficult to test. It is no wonder, then, that when AI researchers emerge from their labs with their latest prototype, the onboard software community issues a collective yawn.

The fields of AI and flight software are both changing, however, and the goal of our research is to understand how these two software methodologies can be combined to more effectively carry out the spacecraft mission. More deterministic expert system techniques, such as model-based reasoning, are now in use operationally. A growing sub-field of AI is actively researching methods for developing testable applications. In areas outside of onboard processing, including commercial applications and ground software, AI has been successfully integrated with conventional software. In fact, in the ground-based domain, AI is fast becoming just another tool in a programmer's repertoire. As more powerful processors and larger memories migrate onboard, the flight software environment will no longer be so highly constrained and onboard software will be able to take advantage of applicable AI techniques.

SYSTEM DEVELOPMENT

As an initial attempt to introduce expert system technology into an onboard environment, a model-based diagnostic system was integrated with prototype flight hardware and software. This work demonstrated the ability of a model-based expert system to isolate hardware faults. It further showed that an expert system could be effectively integrated with conventional flight software to produce a significant improvement in diagnostic capabilities. For purposes of this demonstration, the testbed hardware and software were also required to operate independently of the expert system. We therefore chose a loosely-coupled software architecture, which limited the amount of communication and cooperation possible between the flight code and the expert system. While this system met all of our initial goals, its limitations have taught us that tighter integration between conventional and AI-based code is needed for a realistic onboard system.

The prototype flight hardware and software illustrated in Figure 1 were developed as part of an advanced concept power subsystem project. The hardware testbed included power control electronics, a solar array simulator, spacecraft batteries, current and voltage sensors, and a 1750A instruction set architecture (ISA) onboard processor. While the testbed was being assembled, the associated power control software was developed in 1750A assembly language on an HP9000 workstation. The power control software included sensor processing functions, a control algorithm to ensure constant battery charge while preventing overcharge, command output functions, and limited fault management capabilities. Once the testbed was assembled and the flight software was completely developed, the code was downloaded to the onboard processor and the hardware and software were integrated and tested.

Although the expert system was planned as part of the demonstration system from its inception, the flight software and the expert system were considered two separate efforts, each with its own development team. The expert system was a model-based diagnostic system developed on a Texas Instruments microExplorer using MARPLE, an in-house model-based expert system shell [Cowles 90; Fesq 91]. The MARPLE shell implements a model-based reasoning technique known as constraint suspension. This technique, developed in the MIT AI Lab, does not attempt to model the behavior of failed components. Instead, it constrains the values placed at system components based on a series of transfer functions and systematically suspends these assumptions to isolate a failed component [Davis 88].

MARPLE is a LISP-based tool containing all the functions necessary to run a model-based diagnostic system: the user need only supply models of the system to be monitored. For this experiment, we developed hierarchical models of the power testbed, including a top-level model of the entire testbed and more detailed models of the power distribution electronics and the solar array simulators. Graphical representations of these models were developed for the expert system user interface shown in Figure 2. After the models were coded using MARPLE's high-level model definition language, the expert system was tested using a VAX-based power system simulator. This simulator allowed us to test the operation of the expert system before integration with the hardware testbed, and also allowed us to exercise the system against a wider range of faults than was possible on the actual hardware.

Communication between the flight software and the expert system was designed to be minimal and to be compatible with the flight software's existing command and telemetry capabilities. A one-way communication scheme was designed in which the flight software periodically sent packets of information containing spacecraft sensor data to the expert system. The expert system converted this raw data stream into voltage and current readings before processing it through the MARPLE models. A full-duplex mode of operation was planned but never executed, in which the expert system would send messages back to the flight software, indicating which unit had failed and allowing the flight software to bypass the fault. The diagnostic capabilities of the expert system were not designed to be integrated with the flight software, but were intended to augment or replace the flight software's fault management capabilities.

values and data were collected. This information was analyzed and the expert system models were modified appropriately. Most modifications involved changing scale factors in the model's equations, although in the case of the power control electronics, a new set of models had to be developed to reflect the component's actual behavior. Data analysis and off-line calibration lasted about two months. This period could have been shortened if contention for use of the testbed had not delayed the data collection process.

After the off-line calibration was complete, we linked the expert system to the flight software and began on-line calibration. An RS-232 link was established between the spacecraft processor and the microExplorer. Every 25 ms the power control software sent testbed sensor data to the expert system. The model-based system was first tested with nominal testbed data (no faults) to assure that it was correctly monitoring the testbed performance.

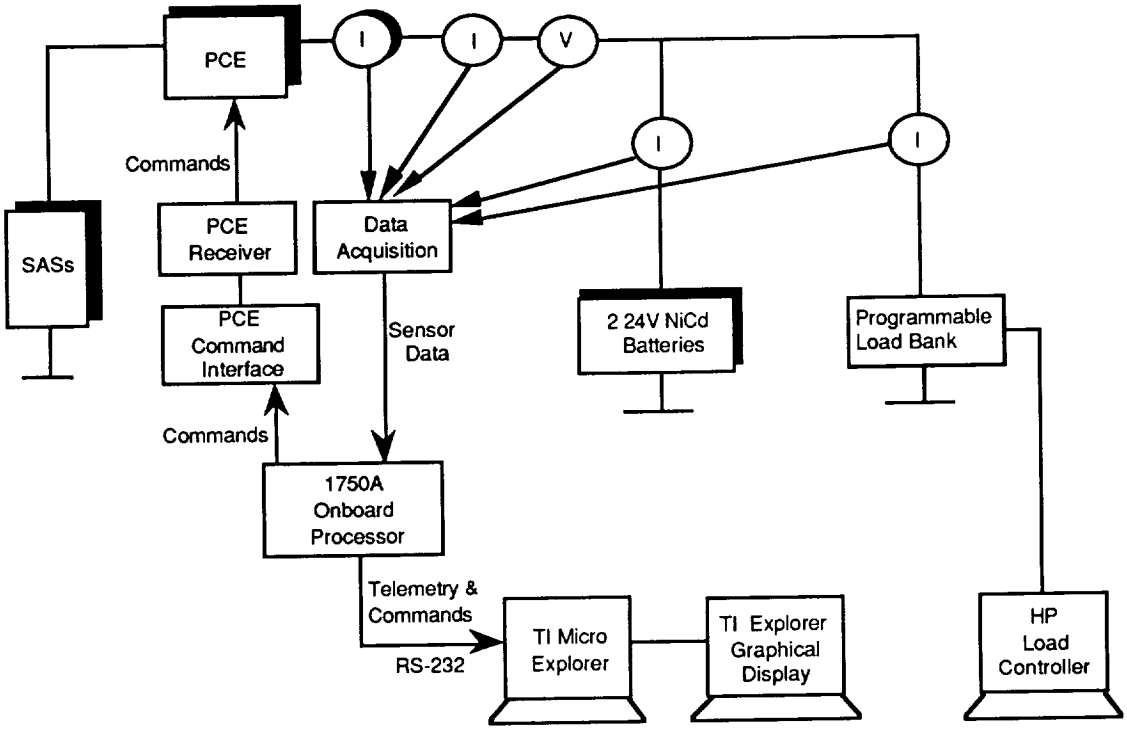

Figure 1. The Advanced Concept Power Subsystem Testbed

Only after the flight hardware, software and expert system were fully developed and tested did we begin integrating the expert system with the prototype power subsystem. The first step in this integration was the calibration of the expert system models to the hardware testbed. Because the simulator used to perform the initial tests on the expert system was developed before the test hardware was available, it was based on idealized models of the power system behavior. Needless to say, these theoretical simulations did not always adequately reflect the actual performance of testbed components. Once the testbed was operational, each component was exercised through the range of its

Although the expert system models were not extremely accurate (most components were modeled within one to three percent of their actual values), the expert system was able to monitor the testbed's performance without producing any false alarms. The expert system was now ready to diagnose failures.

Prior to integration, we identified several typical on-orbit faults which could be safely simulated on the hardware. The faults tested included open and short circuit failures in the power control electronics, open circuit battery cells, shadowed solar arrays and failed sensors. These faults were injected into the testbed during a normal

system run and the expert system's performance was monitored. In most cases, the expert system was able to diagnose the fault immediately, displaying its conclusions on the graphical user interface.

RESULTS

Through a series of extensive tests, we proved that the model-based technique is capable of diagnosing hardware faults. Table 1 summarizes the faults that we injected in the testbed and the ability of the expert system to isolate the source of each failure. A majority of the test cases involved open or short circuit failures in the power control electronics since these failures may be easily injected in the hardware by removing a fuse, or by connecting the output of a solar array simulator directly to the output of the power control electronics. Over the range of operation for which the expert system was calibrated, it diagnosed the correct unit in two-thirds of the open circuit test cases, and isolated the fault to correct string of the power control electronics in the remaining cases. Additional open circuit tests were performed in the battery overcharge region, for which no calibration data was available. As might be expected, the diagnostic system fared poorly in this region, correctly identifying less than 50 percent of the faults. In the short circuit tests, however, the expert system performed well in both the overcharge and normal operation modes. In both of these regions of operation, the model-based technique was able to correctly identify the faulty string each time a short circuit failure was induced. In 66 percent of the short circuit test cases, the expert system was able to isolate the fault to the component level.

Tests were performed with failures injected in the solar array simulators to verify the ability of the expert system to distinguish between solar array and power control electronics failures. Open, short and shadowed solar array cells were induced in the testbed by programming the solar array simulators with the skewed characteristics of these failures. Since simulation was the the only means of causing these failures, test cases were not as extensive as with hardware induced failures. The expert system was able to isolate a failed string of the solar array for each type of fault induced. Although successful, test cases of battery failures were necessarily limited because of the inherent danger of introducing short or open circuits in battery cells. (An explosion of a 48V battery would have been detected by the expert system, but the test conductors may not have survived to document the results.)

Although the flight software and the expert system development efforts were segregated, communication

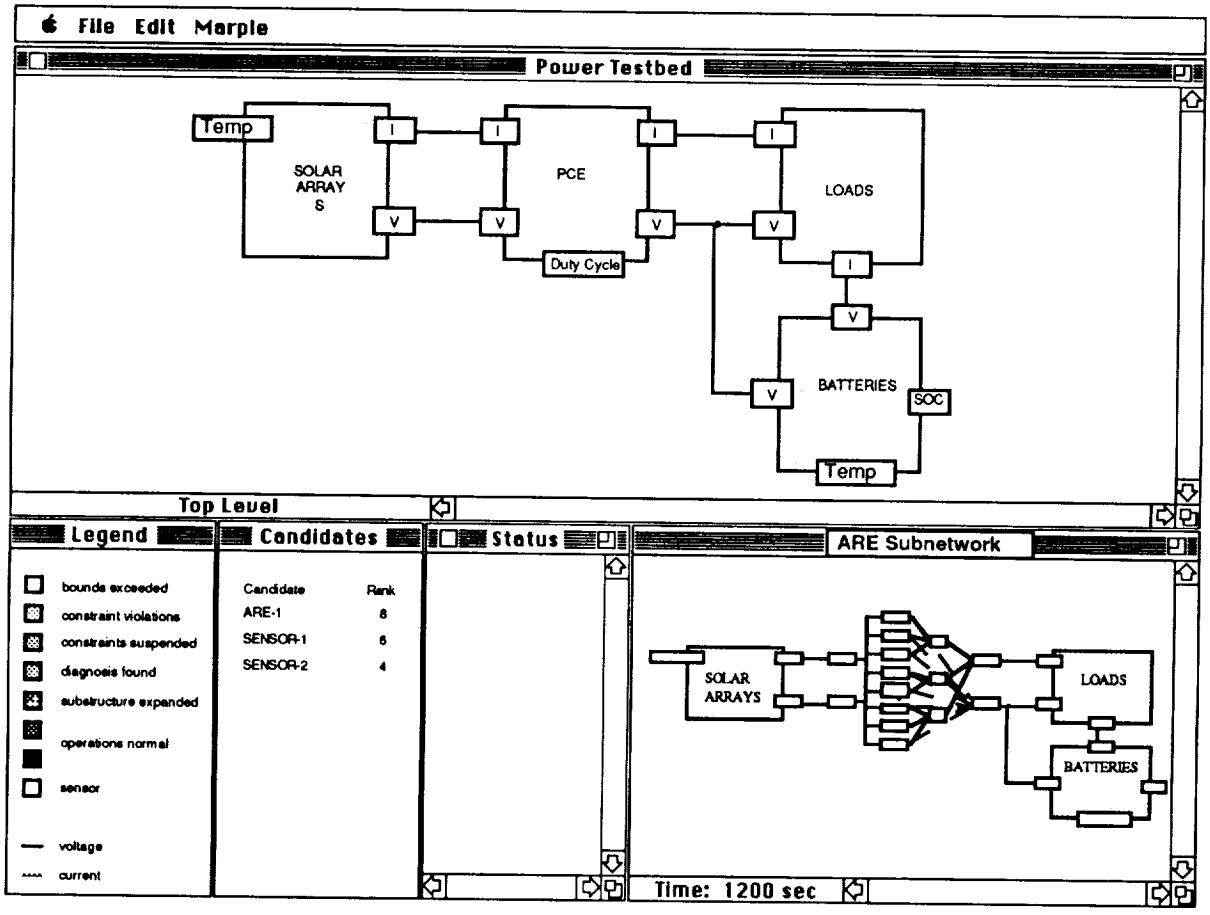

Figure 2. MARPLE User Interface developed for the advanced power testbed.

between the hardware, software and expert system personnel was strong throughout the life of the project. The expert system developers participated in weekly project meetings and were co-located in the same lab as the hardware and flight software developers. The testbed engineers, although busy, were interested in the expert system project and willing to answer questions. This close communication contributed to the relative ease with which the expert system was integrated with the hardware and flight software.

Another factor which helped smooth the integration effort was the extensive off-line calibration performed on the expert system using testbed data. Long hours were spent acquiring performance data, analyzing the results, and adjusting the expert system models as necessary. Having models which accurately represented the hardware proved invaluable in reducing the schedule required to test the expert system. From the start of on-line testing, the expert system was able to model the behavior of the testbed, allowing us to proceed with the test cases rather than refining the models for each fault injected. The importance of off-line calibration is apparent in our test results. The expert system was much less accurate in regions for which inadequate calibration data was available, such as at very high battery voltages.

The competition for time on the testbed hardware made off-line calibration even more critical. The hardware designers needed the testbed to validate their design and acquire performance data; flight software developers were required to verify the execution of their control algorithms on the actual hardware. Expert system developers were allocated any remaining time for calibration and testing. Conducting the data analysis and model adjustment off-line limited system contention and allowed the hardware and software developers to complete their tasks with little impact from the introduction of the expert system.

The fault diagnosis capabilities of the expert system were a significant improvement over those designed for the flight software. The flight software fault management capabilities were limited to reacting to faults by switching on additional circuitry: while it could identify broad classes of faults and initiate recovery strategies, the flight software could not identify exactly which component had failed. The expert system was able to identify not only what component failed, but in some cases it could characterize the nature of that failure (for example, a short-circuit versus an open-circuit failure in the power control electronics). The expert system also proved more adept at diagnosing sensor failures, immediately identifying the failed sensor, as opposed to the lengthy (and inconclusive) isolation algorithm implemented in the control software.

Although the expert system demonstrated more advanced fault detection and isolation capabilities than the flight software, much of the fault management processing performed by the two systems proved to be redundant. For demonstration and testing, this redundancy served a useful purpose; it allowed us to compare the performance of the expert system to the standard fault detection and isolation techniques used by the flight software. While some redundancy might be beneficial in an onboard system, the overlap between our expert system and flight software reflected the segregated development efforts and was too inefficient for use onboard. If the design of the two systems had been more coordinated, the fault diagnostic capabilities of the expert system could have become an integral component of the flight software's fault management scheme. Instead, the flight software developers designed a standard technique of analyzing spacecraft sensor data and determining faults while the expert system developers analyzed the same data using model-based reasoning with the same end goal -- diagnosing failures in the hardware.

By the completion of our research, we had made significant progress in proving the ability of model-based reasoning to diagnose hardware failures. This success was based on the development of two distinct working systems -- a prototype flight hardware and software system which successfully demonstrated a new concept in power control, and a model-based expert system which could isolate faults in the power system. Any attempt to integrate both systems on a single spacecraft processor, however, would require significant redesign of both systems. The control software would have to be modified to take full advantage of the expert system's fault diagnostics and ensure that data required by both the expert system and flight software were properly managed. The expert system would have to be streamlined to run on a target machine with limited throughput and memory and would have to be converted to a language supported by the spacecraft processor.

DISCUSSION

Although we tested our expert system extensively against both the software simulator and the actual testbed, these tests did not begin to meet the rigorous standards required for flight software. Because it was designed as a demonstration, not a flight system, low-level unit verification tests were not performed on the expert system. Given the relatively structured nature of the model-based reasoning approach used, unit testing could have been successfully performed on most of the expert system code. It is doubtful, however, that all possible execution paths could have been tested under realistic project budget constraints. Similarly, while the expert system performed well against both the simulator and the hardware, its reliability was not proven to be appropriate for onboard use. Again, it is important to realize that this system was designed as a demonstration. An onboard application would have been designed to achieve a higher level of reliability, and calibration and testing would have been more extensive. However, while more thorough testing and calibration undoubtedly would have increased the system's reliability, the size and potential number of execution paths through the expert system would have prevented standard testing methods from adequately measuring this reliability.

Current research at TRW seeks to develop new verification and reliability assessment methods for this type of model-based system based on a mathematical analysis of the expert system's properties. Additionally, the expert system used for this project has been redesigned and recoded in Ada, and hosted on a 1750A ISA processor. While such work eventually may prove fruitful for introducing large scale flight-based expert systems, our research suggests that such large, loosely coupled applications may not be the most effective use of AI onboard. For many applications, a tighter coupling of AI and procedural code tailored to the particular problem at hand would be more effective. In such a system, traditional procedural code would be enhanced by small, embedded applications of AI-based techniques. Because these small model- or rule-based modules would have a limited number of execution paths, they could be verified through standard methods. Such an approach could be used to avoid unnecessary redundancy and capitalize on the strengths of each programming technique. This embedded approach would also eliminate the need to redesign and rehost a ground-based prototype to run on a flight processor. If AI sections of code were designed and developed as part of an integrated flight software development effort, such problems as processor contention, communication, and integration could be simplified and a more powerful flight software package would result.

At the conclusion of our project, a team of expert system and flight software developers revisited the control software requirements and design. As part of this effort, the AI personnel conducted a detailed study of the flight software's goals and design, and the flight software engineers assessed the strengths and limitations of the expert system. The team then worked together to identify several sets of requirements which might be better served by AI-derived techniques than by procedural code. These included battery recharge ratio adjustment, optimizing peak power tracking, sensor consistency checking, performance prediction and monitoring the pattern of controller faults. In each of these cases, the AI-technique would have been more efficient, more accurate or more flexible than the procedural implementation. Also, each of these mini-applications of AI would be well-defined, so that its performance could be thoroughly characterized through test procedures. Were this sort of analysis to be

Fault Induced	Number of Trials	Test Method	Expert System Diagnosis	Remarks
PCE FAULTS				
Open Circuit	8	H/W	6 PCE, 2 BATT	Two BATT from overcharge
Short Circuit	6	H/W	PCE	
Command I/F	1	H/W	PCE	
BATTERY FAULTS				
Low Discharge	10	VAX SIM	BATT	H/W Test may damage battery
Open Cell	1	H/W	BATT	
Short Cell	1	H/W	BATT	
Overtemp	10	VAX SIM	BATT	No temp sensor on testbed
S/A FAULTS				
Open Cell	1	H/W SIM	S/A	
Short Cell	1	H/W SIM	S/A	
Shadow	10	H/W SIM	S/A	
Temp Increase	10	VAX SIM	S/A	
SENSORS				
Stuck on	1	H/W	SENSOR	Current/voltage sensors
Stuck off	1	H/W	SENSOR	
LOAD FAULTS				
Disable/Enable	10	VAX SIM	LOAD	No i/f with loads on H/W
Change sizing	10	VAX SIM	LOAD	
S/A THREATS				
Pellets	10	VAX SIM	S/A	No temp sensor on testbed
Laser Attack	10	VAX SIM	S/A	

Note: All test cases performed on the hardware testbed were also successfully executed with the VAX-based simulator.

Table 1. A Summary of Model-based System Test Results

performed in the initial design phase, AI research could begin to benefit onboard software without violating its constraints.

The migration of AI technology onboard will require many changes from the methods typically used in developing spacecraft data systems. These changes will not occur without complete support of project management. Although flight software developers today require a knowledge of general hardware characteristics such as sampling delays and sensor accuracies, the calibrated models of an expert system require more in-depth knowledge of the hardware. Acquiring adequate data requires the commitment of the hardware designers and test engineers, as well as project managers who must assure that sufficient time is allocated on the hardware test system. As was evident during our research, many diverse tasks must be performed on the equipment, and priorities must be established to efficiently use this limited resource. The problem exists today and will only increase with the added competition for use of the hardware by expert system developers.

CONCLUSIONS

AI's migration onboard may not be accomplished through one giant leap, but rather in a series of small steps. As spacecraft systems and missions become more complex and demand greater autonomy, more high-level control functions will be carried out by the onboard software. AI research is seeking to automate many of the high-level applications such as diagnosis and planning that are now performed on the ground. In our research we have shown that a model-based expert system can interact with prototype flight software and correctly identify several hardware faults. However, this system and other large scale AI prototypes cannot yet meet the verification and validation requirements and other restrictions imposed by the highly constrained onboard environment.

Eventually researchers will be able to verify their AI systems and flight processors will grow sufficiently in size and throughput to accommodate large expert systems. However, even when feasible, porting these full-scale AI systems onboard may not be the most effective application of this technology. The loosely coupled architecture developed for our application did not allow the programs to take full advantage of each other's capabilities and led to redundant processing. A tighter integration of AI and conventional software in which each technique is applied to those system requirements to which it is best suited may be a more effective and easily managed solution. This approach would require flight software designers and AI software developers to work together to produce onboard software capable of meeting the needs of future spacecraft missions.

REFERENCES

[Cowles 90] S. D. Cowles, L. M. Fesq, A. Stephan, and E. J. Wild, "Expert Systems: A New Approach to Spacecraft Autonomy," TRW Quest Magazine, Volume 13, Number 1, Summer 1990.

[Davis 88] R. Davis and W. Hamscher, "Model-based Reasoning: Troubleshooting," Proceedings of AAAI-88, St. Paul, Minnesota, 1988.

[Fesq 89] L. M. Fesq and A. Stephan, "Advances in Spacecraft Autonomy using Artificial Intelligence Techniques," Proceedings of the Guidance and Control Conference, Keystone, Colorado, Volume 68, 1989.

[Fesq 91] L. M. Fesq, A. Stephan, E. R. Martin, and M. G. leRutte, "Model-based Reasoning for Space Station Freedom," Proceedings of the 26th Intersociety Energy Conversion Engineering Conference, Boston, Massachusetts, 1991.

[Filarey 90] A. R. Filarey, "Developing Onboard Spacecraft Software in Ada," Proceedings of the 1990 TRW Conference on Evolutionary Development Experience and Supporting Ada Technologies, Dominguez Hills, California, 1990