Persistence-Based Production Rules for On-Board Satellite Automation

Michael A. Swartwout, Christopher A. Kitts, Rajesh K. Batra
Space Systems Development Laboratory
Stanford University
Durand 250 / 496 Lomita Mall
Stanford, CA 94305-4035
650/725-6794 (rcktboy@leland.stanford.edu)

Abstract— As the cost of operations comes under greater scrutiny, the use of automated systems is receiving greater attention. A vital aspect of on-board automation is the manner in which system knowledge and operational procedures are represented. Production rules are a commonly-implemented method. They possess the advantage of coherent, logical formulation of responses, represented in a manner consistent with operational procedures. However, they can be inflexible to changing parameters, insensitive to issues of time duration, and require many rules to cover all cases. To be of greater use to automated space missions, production-rule systems need to be augmented with the ability to handle uncertainty and incorporate temporal characteristics.

As part of its research program in space system operations and autonomy, Stanford University's Space Systems Development Laboratory (SSDL) has developed an advanced production rule approach to automation. These production rules track the temporal persistence of conditionals as a primitive aspect of their implementation. The persistence-based production rules have been integrated into the Chatterbox flight software on Sapphire, SSDL's first satellite. Chatterbox's programmable production rules control a beacon-based health management system as well as other on-board activities.

This paper motivates the use of persistence as an integral aspect of automated health monitoring, introduces the concept of an "integral of threat" to replace static alarm thresholds, describes how persistence tracking has been integrated into Chatterbox, and reviews the applications of the resulting system for SSDL's spacecraft.

TABLE OF CONTENTS

1. INTRODUCTION

As the cost of spacecraft mission operations comes under greater scrutiny, the use of automated systems is receiving greater attention. Automation allows for cost savings during nominal and even off-nominal operations. Additionally, on-board automation can enable advanced mission concepts that cannot be effectively performed with ground-based control.

A vital aspect of on-board automation is the manner in which system knowledge and operational procedures are represented. If the spacecraft is to be highly autonomous, it must possess the ability to clearly determine parameters such as the vehicle state and the status of mission goals, and to formulate responses that are flexible to these time-varying parameters. It is, however, equally important that the information be conveyed to ground operators in a manner consistent with their capabilities to understand and reason. This trade-off must also take into account the capabilities of the computational hardware available to the spacecraft.

Production rules are a commonly-implemented solution to such automation problems. They possess the advantage of coherent, logical formulation of responses, represented in a manner consistent with standard operational procedures. However, they can be inflexible to changing parameters, insensitive to issues of time duration, and can become quite complex. To be of greater use to automated space missions, production-rule systems need to be augmented with the ability to handle uncertainty and perform temporal reasoning.

As part of its research program in space system operations and autonomy, Stanford University's Space Systems Development Laboratory (SSDL) has developed an advanced production rule system to support on-board spacecraft automation. This production rule system tracks the temporal persistence of conditional states as a primitive aspect of its implementation.

The persistence-based production rule system has been integrated into the Chatterbox flight software that has been developed for Sapphire, SSDL's first spacecraft. Chatterbox's programmable production rules are used to control a beacon-based health management system as well as other on-board anomaly protection and management activities.

Operational experience with Sapphire has led to several innovations and lessons learned. These lessons will be applied to creating a more advanced version of this software for SSDL's future multi-satellite Emerald mission. The production rules will cover similar responsibilities as on Sapphire and assume other on-board management activities.

Automation vs. Autonomy

Before further discussion, it will be helpful to clarify the definitions of "automation" and "autonomy". Automation as defined for space systems is the capability to perform a specific task without outside assistance. Typically, the attitude control subsystems of many spacecraft are automated such that the vehicle can point in a specific direction in the presence of unknown disturbances. Autonomy is the capability to translate ambiguous goals into specific action, usually while meeting a set of time-varying constraints. The Deep Space 1 spacecraft, for example, has an autonomous navigation system: a specific comet is identified and the spacecraft is capable of targeting, tracking, and adjusting its orbit to intercept that comet without further input from the ground [1].

There has been much study in both fields of autonomy and automation. The focus of this paper is the automation of spacecraft health monitoring procedures, which is intended to reduce the cost and improve timely response of spacecraft anomaly detection. The bulk of this study is therefore devoted to automating this particular task. However, such automation innovations can also be applied to autonomous operations; these possibilities are also discussed.

This paper motivates the use of persistence as an integral aspect of automated health monitoring, introduces the “integral of threat” concept as an alternative to static alarm limits, describes how persistence tracking has been integrated into the Chatterbox production rule system, and reviews the applications of the resulting system for SSDL's spacecraft.

2. Production Rules for Automation

Automation can improve both the competitive performance of space missions, often considered in terms of cost, timeliness, and quality. Mission quality is improved because highly capable electronics allow information to processed more rapidly with fewer mistakes, compared with conventional, human-intensive approaches. It is also expected that automated systems require fewer humans to manage the same tasks compared with a conventional approach, thereby reducing cost. Another significant advantage of automation is it enables the migration of responsibilities from ground control to spacecraft control, which allows for faster response and less resources devoted to communication.

However, before automation can be made useful, the responsibilities of the human operators must be translated into a coherent set of procedures. This seemingly simple axiom is actually quite difficult to apply to spacecraft operations, because much of conventional operations is based on experiential reasoning, using procedures and equipment incrementally developed over a series of missions.

One of the commonly used methods for expressing operational procedures in both automated and conventional systems is to formulate conditional, "if-then" statements. Also called production rules, these conditionals capture both primitive and high-level directions about the system. Typical production rules are: "If the battery temperature exceeds 30ºC, then put the vehicle into safe mode," and "If the vehicle is in safe mode, then schedule additional contacts and notify vehicle experts."

Advantages of Production Rules

One of the primary advantages to production rules is that they express information in the same manner used by many human operators, while at the same time this format is expressible in many computing languages. Therefore the procedures can be easily translated into code, and the automated system is easy for humans to understand, troubleshoot, and reason about. This logical transparency is very important when human operators must interact with an automated system [2, 3].

Production rules are simple to implement; if-then constructs exist in many programming languages. Because of their primitive nature, production rules can be incorporated into a variety of complex or sophisticated architectures for execution. For example, production rules can be incorporated into a "production system," where the order of execution does not matter. This enables very efficient code structures and powerful algorithms to implement them.

Whatever the method, the essential point is that the process of executing production rules can be developed independently of the formulation of the rules themselves. This scheme allows for great flexibility on the part of the designer. As long as the production rules reflect the operational tasks to be automated, a variety of properly constructed implementations are possible.

Disadvantages of Production Rules

The disadvantages of production rules can be summarized in one word: fragility. Production rules tend to be narrowly defined to effectively manage specific situations, but they do not adapt well to uncertain conditions. Unlike human operators, automated methods using production rules do not possess the ability to intuitively reason about new situations. Furthermore, although the rules reflect the pre-defined logical decisions used by operators, it is not at all clear that the operators' established methods are the best approach for many tasks in spacecraft operations.

For example, in the task of health monitoring, it is typical to define high and low alarm thresholds for each sensor; when a sensor goes beyond the thresholds, the component is defined to be in a dangerous condition and the operator takes action. If the sensor is noisy, however, the threshold may be crossed even though the component is indeed healthy. Moreover, for a noisy sensor or even a physical condition where the output is hovering near the threshold, there could be repeated excursions into and out of alarm conditions. In these common situations, the manner in which anomalies are defined – threshold crossing – impairs the operator’s ability to discern true anomalies from false alarms.

In addition, production rules can only be written for sensors and conditions that are known. Given the extreme complexity of modern spacecraft, not all health conditions can be determined at design time. It may be that a certain sensor

reflects a previously unknown phenomenon vital to the understanding of the vehicle state. Until this condition is identified, it goes unreported. Incrementally modifying an automated system based on experience is a cumbersome means of defining operational procedures.

Case Study: Alarm Thresholds

Both the advantages and disadvantages of production rules can be demonstrated in a brief study of alarm thresholds. As mentioned above, it is common in mission operations to define functional and survival high and low limits for each vehicle sensor. Much of the operator's time in health monitoring is spent verifying that the sensors remain within acceptable limits.

The process of defining alarm thresholds is at best an inexact science. Typically, the thresholds reflect functional or survival limits for components as specified by the manufacturer, often padded with a safety margin. However, a component may be malfunctioning even though its outputs are well within the functional limits. A sensor's expected output usually falls in a narrower band, inside the functional limits, that depends on the functional mode of the spacecraft and environmental conditions such as sunlight or eclipse. Simple high/low thresholds cannot accommodate more sophisticated models of the system.

To work around this limitation, operators often define a series of alarm thresholds for a given component, based on the type of response needed or, in some cases, the mode of the vehicle. The limits are checked on a case-by-case basis. While such accommodations enable the simple rule-based approach to be used, effective use of the rules requires a large and cumbersome set of conditionals to cover every variation of every state. Production rules are therefore limited by computational complexity, programming error, and the ability to rationally cover every possible alternative.

Philosophically speaking, the health monitoring task is not fully expressed by the concept of alarm thresholds. For example, if the upper limit of the battery temperature is defined as 30ºC, there is a fundamental difference between sensor readings of 30.01ºC and 45ºC. A human operator would understand the difference between a slight or extreme threshold violation, but a production rule defines both as the same limit crossing. Most missions work around this issue by implementing different alarm thresholds for each sensor, but as was mentioned above, this leads to a cumbersome set of rules. Even with this exhaustive set of rules, this concept of overshoot is not truly expressed.

Furthermore, a human operator would not make a fundamental distinction between a component at 29.99ºC and the same one at 30.01ºC, yet the former is considered nominal and the latter is an alarm. Defining alarms based on precise limits – a definition forced by the use of production rules – obscures the definition of anomalous behavior because it creates a fixed and arbitrary line between nominal and anomalous behavior.

Conceptually, it would make sense that an output that hovers just below the alarm threshold for several hours is more of a concern than one that climbs from nominal, dips just into the alert range, and then returns to well within in the nominal range. Yet in a conventional system, only the second type triggers the alarm, although it may not truly be a threat to the component's health.

Each of these concepts is illustrated in Figure 1. A conceptual difference exists between a lingering, high-value sensor output (Type 1) and a short-duration spike (Type 2), but alarm-threshold rules only consider the spike to be anomalous. Moreover, a spike with a slightly lower threshold (Type 3) would escape detection altogether, although there is little true difference between these types.

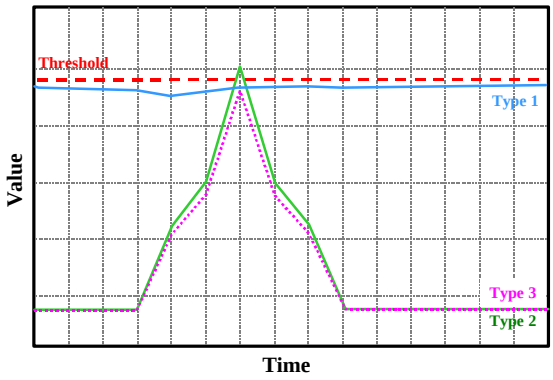

Figure 1 Illustration of the difference between lingering near the threshold (Type 1), spiking above it (Type 2) or spiking near it (Type3)

In summary, production rules provide a powerful and easy means for implementing standard operating procedures. But for certain tasks, especially anomaly detection, production rules have significant shortcomings. The rules are highly dependent on how parameters such as alarm limits are specified, and their inability to capture higher-level information such as temporal constraints constrains their effective implementation.

3. Persistence & Temporal Reasoning

Because of the limitations described above, advanced anomaly-detection techniques typically employ other methods than alarm thresholds to distinguish abnormal behavior from normal behavior. These methods also require more advanced computing capabilities. However, given the limited processing available to current and planned SSDL spacecraft, the authors were reluctant to abandon the simplicity and operator-readability of production rules.

It was believed that a simple modification – incorporating an explicit sense of time – could enable enhanced operational performance without requiring significantly enhanced computational performance. This modification is called persistence, which indicates that the condition for a rule must hold true for a specified amount of time before the action is executed.

One potential implementation is presented in Figure 2. In this approach persistence is measured as a counter that increments each sensing cycle where the sensor output exceeds the threshold and decrements when the output is measured within the nominal range. The maximum persistence in this example is four cycles. Once the maximum is reached, the conditional clause is defined to "latch" true; the condition is considered to be true until the persistence counter decrements back to the minimum value, at which point the conditional "unlatches" back to false. This particular implementation is used for the Chatterbox operating system and will be discussed in some detail, below.

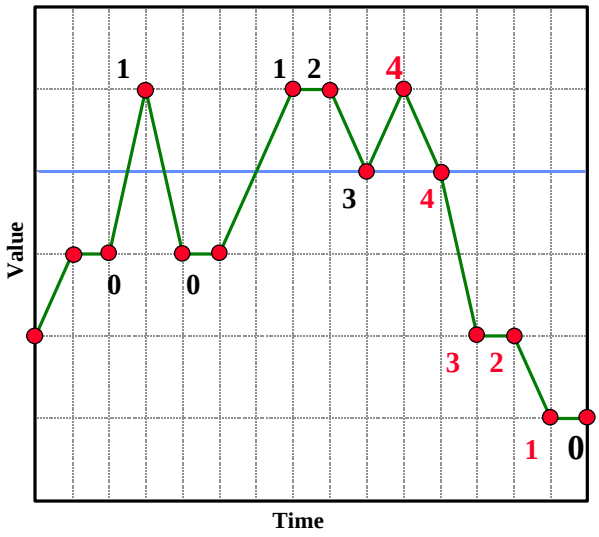

Figure 2 An example of persistence with the time threshold at four cycles; high latching is in red, low latching in black.

Implementing persistence brings immediate benefits. It directly addresses the issues of noisy sensors and limit cycling. There is an explicit lag between the time the sensor crosses the alarm threshold and the time it is considered a true reading; presumably, confidence that this is a true threshold crossing grows as persistence increases. While this lag can adversely affect the timely response to a true alarm, it is possible to redefine the alarm thresholds and persistence values to minimize the impact of lag.

Persistence-tracking production rules have already been applied to on-board fault protection in space missions, such as Magellan [4] and Cassini [5]. Persistence was used in these missions primarily to protect against false alarms. They also tended to favor one-directional persistence counters; the counter would count up but would reset to zero if the output returned to nominal.

Conceptual Power of Persistence

However, these and other methods to track persistence have not explored the conceptual underpinnings. Such an investigation allows for more effective use of persistence for production rules, and points to future powerful applications.

Figure 1 presented the dilemma of two outputs: one, which hovers just below a threshold – setting off no alarm – and another that briefly spikes above it. The underlying reason behind this dilemma is that it is not just the sensor value that matters, but also how long it persists above the threshold. This fact is demonstrated by the nature of Figure 1 itself; the problem becomes apparent when a time-history of the sensor output is plotted against the alarm thresholds.

In a very real sense, a component alert threshold is better formulated as an integral. True violation of an alert threshold is when the time-integrated sum of a sensor output over a specified time interval gets too high. Such a conceptualization enables very simple rules to cover a broad spectrum of thresholds. The batteries remaining at an elevated temperature for a long period of time are considered just as threatened as a short duration high-temperature event. Figure 3 demonstrates the concept of the “integral of threat.”

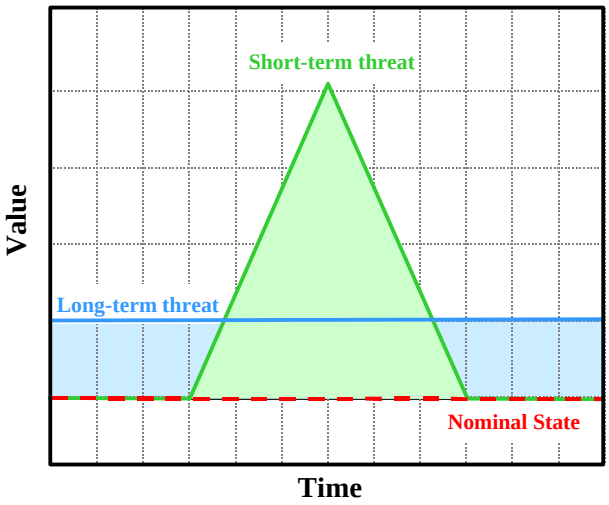

Figure 3 Comparing the integral of threats between a low-value long-term threat and a short-term high-value threat.

For systems with sufficient computing power, persistence counters provide an effective way to represent this "integral of the threat." Each time the value persists above a nominal level of interest, the overshoot can be added to the total threat sum until the sum exceeds its allowed maximum. Even for systems not capable of such sums (such as Sapphire's Chatterbox, described below), the integral can be approximated by setting a low alarm threshold with a high persistence to catch the lingering threats, and having a second higher alarm threshold with a low persistence to catch the short-duration spikes. Such an approach does not solve the problem of having many rules for the same sensor output, but the addition of persistence tracking still enables better performance. It also serves to demonstrate the usefulness of the method for future implementation with more capable computers.

Structure of Persistence

The discussions of persistence have thus far largely ignored the specific implementation details. There are several options, such as strictly incrementing versus incrementing and

decrementing, and latching versus resetting to default. The specific choice will depend on the nature of the production rules being implemented. The Sapphire study of Section 4 uses persistence counters which both increment and decrement, and which latch true. The design choices involved will be explained, below.

4. Implementation: Sapphire

Sapphire is the first spacecraft designed, built and managed by students at SSDL [6]. Shown in Figure 4, the Sapphire vehicle is approximately 44 cm across, 31 cm tall, with a mass of 18 kg. The project was started in January 1994 and the spacecraft was fully tested and completed in July 1998. It is now awaiting a secondary launch opportunity. This delay in securing a launch has been a mixed blessing, as it has allowed the vehicle to be part of several controlled ground-based experiments such as this production rule study.

Introduction to Sapphire

Sapphire's major components are modified and extensively tested commercial products, such as a Motorola 68332 microcontroller, Hamtronics 70 cm band transmitter and 2 m band receiver, and Kantronics terminal node controller. Several student-built boards are responsible for telemetry sensing and analog to digital conversion, and students fabricated the aluminum honeycomb structure as well. Space-rated Sanyo batteries, GaAs solar cells, and radiation-hardened memory are strategically included in the spacecraft. The spacecraft is passively stabilized by a combination of permanent, body-mounted permanent magnets and a radiometer-induced spin caused by different paints on the turnstile transmit antennae.

Sapphire's primary mission is to perform first-flight testing of a set of micromachined infrared sensors. Developed by Professor Tom Kenny of Stanford's Design Division, these devices are room-temperature edge detectors that fit within a microchip [7]. Sapphire's other instruments involve a student-interest commercial camera and voice synthesizer.

In addition, Sapphire has become a research testbed for the authors' PhD projects within SSDL. It will be the first vehicle fully operated within the ASSET operations architecture [8] and serves as an experimental platform for research in beacon-based health monitoring [9, 10]. In this latter function, the flight software for Sapphire was modified to enable on-board health assessment.

Written in C, the Chatterbox operating system was developed by SSDL students for the 68332 microprocessor, which serves as Sapphire's CPU [11]. Chatterbox is a bulletin-board system with a UNIX-like interface, capable of handling up to 26 separate users with varying degrees of access permissions. Functions are divided into high-level controllers and device drivers, allowing for future upgrades to individual components without major code overhaul. Specific built-in features include a file system, time-based command scheduling, programmable variables, and the production rule system, described below. The Chatterbox code consists of 15000 lines that compile into a 148 kb application, plus another 4000 lines and 134 kb for the software drivers specific to the camera payload.

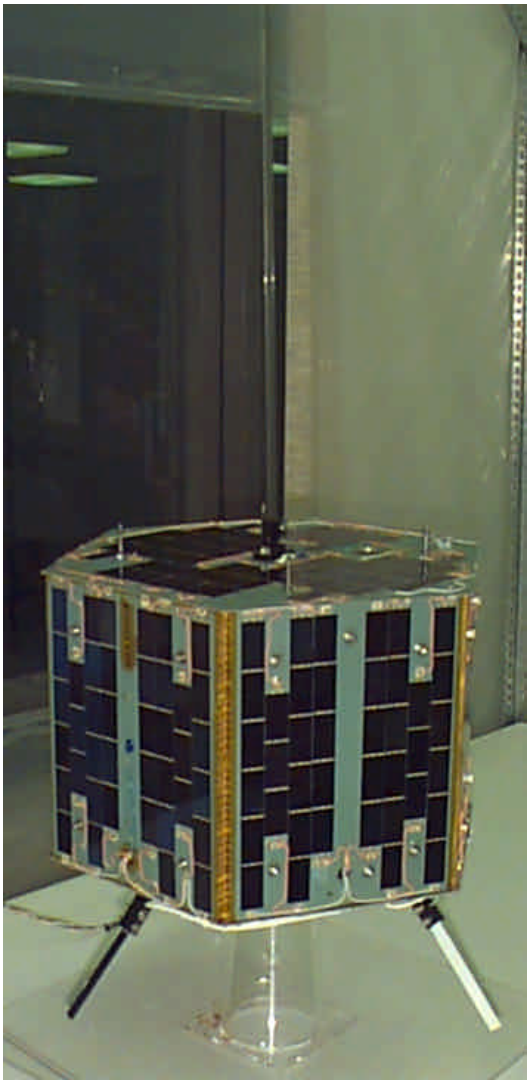

Figure 4 Sapphire in launch configuration; the umbilical cord in the lower left is an external power source which also trickle-charges the batteries.

Overview of Beacon Monitoring

Beacon-based health monitoring, also called beacon monitoring, is a proposed cost-saving alternative to conventional anomaly detection and notification for spacecraft [12]. The ground control no longer performs regular state of health contacts, resulting in significant cost savings in communication resources and manpower. The on-board software becomes responsible for observing the vehicle state and determining whether or not the system is healthy. Instead of downlinking the full telemetry to the ground, the vehicle broadcasts a simple low-bandwidth signal that maps the vehicle health to a few states corresponding to the urgency of operator response. The signals are received by automated stations and relayed to mission control.

Implementation of Production Rules

The production rule system within Chatterbox is based on comparing a sensor against high and low thresholds. An output that persists latches the rule and its action is executed. Any system command can be used as the action clause of the production rule. This ability to utilize any command – or even sequence of commands – in the response gives Chatterbox great flexibility in implementation. Furthermore, any of the production rules can be uploaded or altered through user commands.

Persistence is an essential element of Chatterbox production rules. Sapphire's sensors are particularly noisy, and thus persistence is needed to prevent cycling around a threshold boundary. Moreover, alarm thresholds that incorporate the time element are necessary in order to make the best use of the beacon monitoring.

Chatterbox implements persistence as a counter. The production rules are checked on a regular interval of ten seconds. The persistence counter is incremented each time the sensor of interest is determined to be outside the threshold limits, and is decremented each time it is within limits. When the rule latches, and every cycle it remains latched, the response is executed. The persistence counter must step back to zero before the rule unlatches.

This definition of persistence is used because it reflects the logic behind the anomaly detection algorithms. Anomalies are defined as persistent excursions beyond some threat threshold. As described in Section 3, some anomalies are defined as a low-level excursion that persists for a long period of time, while others indicate a high-level excursion that may only last a short time. Since the Sapphire system has different responses to each of these threat types, it is helpful to monitor them with different production rules and persistence counters.

Note that this method for implementing persistence can be ambiguous, since the same counter is used for high and low limits. A sensor that jumped from an excessively high value to an excessively low value between cycles would increment the counter without any indication of this radical jump. For Sapphire it was assumed that sensors would not exhibit such behavior, and alert thresholds are selected accordingly.

The production rules are implemented as instances of a specialized table structure. The structure consists of the following elements: sensor to check, low threshold, high threshold, persistence threshold, persistence counter, latch indicator, and response. Typical table entries are listed in Table 1. Note that latch indicator and persistence counters are not defined by the user; the values change based on sensor outputs. Therefore they are not listed in Table 1.

The Chatterbox tables are grouped into one main and up to ten sub-tables. The rules of the main table are checked, sequentially, every cycle. If a rule is triggered, then the response is entered into the spacecraft command queue, and table checking resumes. Commands in the queue are executed sequentially, though not necessarily immediately after they are placed in the queue.

Instead of calling for a command, a production rule's response may be to jump to the start of another table. This allows mode-dependent tables to be checked only when certain conditions apply. For example, Sapphire has different thresholds for some components depending on whether the vehicle is in sunlight or eclipse. The mode-based tables are called based on the measured input of the solar panels. Once a sub-table has been checked from start to finish, the next entry on the original table is examined. Sub-tables can jump to other sub-tables as well.

During development of the production rule system, it became evident that a set of programmable counters would be extremely helpful. For example, it is necessary to keep track of the number of panels in sunlight in order to determine whether the spacecraft is in eclipse. Chatterbox was modified to support these "virtual telemetry channels." These channels can be thought of as "scratch paper" that Chatterbox can use to keep sums and store values. They are observable to both Chatterbox and operators, and they are treated like telemetry sensors in all respects – save that the values of these sensors can be incremented, decremented, or set by command. The addition of these channels enables Chatterbox to meet or exceed all the requirements of the beacon monitoring experiment.

Use of Production Rules – Beacon Monitoring Experiment

The primary use of the production rule system of Chatterbox is to support beacon-based health monitoring of the Sapphire spacecraft. These rules are used to encode a simple but powerful on-board health monitoring system.

Table 1 Sample table entries for Chatterbox health monitoring system

Channel	Low	High	Persist	Action	Description
30	2217	2490	4	sensor set 60 step 1	*Increment channel 60 (critical_event_counter) if channel 30 (BAT2 temperature) persists 4 cycles above or below the specified bit readings*
58	0	250	1	sensor acquire 0-63 2 1	*Take a full-telemetry snapshot if channel 58 (alert_episode_timer) is not between 0 and 250*
60	-	1	1	os beacon message 10; os users delete 65-68; os pins set 0-4 0	*Set beacon message to '10' (critical), log off all users, and turn off all payloads if channel 60 (critical_event_counter) is 1 or more*
0	-	0	1	9	*Jump to table 9 if channel 0 is 0 or higher (i.e., always jump to table 9)*

The main tasks of Sapphire's health monitoring system are listed in Table 2, along with the types of sensors used. Each of these tasks correspond to a sub-table used on Sapphire. Although the table language is somewhat cryptic, the ability to correlate sub-tables with specific functions greatly enhances operator/programmer understanding of flight system behavior.

The Sapphire spacecraft has been involved in a number of beacon monitoring experiments since August 1998 [13]. Since these experiments are designed to test the notification aspect of beacon monitoring, they do not compare the performance of the ground and on-board anomaly detection methods. In fact, the operators rely on the conclusions of Chatterbox to determine whether or not the vehicle is healthy and perform checks to confirm that Chatterbox is functioning properly. Still, these experiments have demonstrated the usefulness of the Chatterbox tables.

Most notably, the first anomaly of the controlled experiment was unplanned: a Northern California heat wave caused a shutdown of the air conditioning in the building housing Sapphire, causing the battery temperature to edge above the safe limits! The production rules included conditions to check on the battery health, so the anomaly was caught and proper action was taken to safe the spacecraft.

Operational Experience & Lessons Learned

The process of running the beacon monitoring experiments and other operator experience with the Chatterbox system has resulted in several lessons for future implementations. This process has been immeasurably aided by the fact that the primary spacecraft operators were involved in defining and coding the flight version of Chatterbox.

Beyond the programming errors and bugs that are normal for an experimental system, Sapphire operators have identified several changes to Chatterbox that would result in enhanced performance. Most importantly, the production rules should have access to more state information. Only the actual telemetry sensors and the user-programmable "virtual channels" can be examined within the production rule framework. Operators suggest altering Chatterbox such that available memory, the on-board clock, and the on/off state of components could be the basis for production rules. These added states would be extremely helpful in formulating very flexible and capable rules.

Experience with these tables also highlights another difficulty of automated systems. As expected, the table-based production rules are easy to read by human operators and easy to troubleshoot. However, this ease of use can be deceptive because the tables are also quite susceptible to poor programming. Several of the authors' initial health monitoring tables involved conceptual oversights, resulting in false alarms and so many summary files generated that the memory was overloaded and shut down! It is imperative that complex, intricate tables such as those used for Sapphire's anomaly detection schemes be carefully proofread and tested before implementing.

Table 2 Major beacon monitoring tables for Sapphire listed in order of execution; parentheses indicate conditional action.

Table	Task	Sensors Used	Commands Issued
1	Determine eclipse state by counting lit panels	Solar panel currents, sunlit panel counter (virtual), eclipse indicator (virtual)	Increment sunlit panel counter, set eclipse indicator
2	Determine whether any alert events are occurring	Temperatures & voltages of many components, alert event counter (virtual)	Increment alert event counter
3	Determine whether any critical events are occurring	Temperatures & voltages of many components, critical event counter (virtual)	Increment critical event counter
(4a)	Sunlight mode, determine whether a summary event is occurring	Temperatures & voltages of many components, summary event counter (virtual), eclipse indicator (virtual)	Increment summary event counter
(4b)	Eclipse mode, determine whether any summary events are occurring	Temperatures & voltages of many components, summary event counter (virtual), eclipse indicator (virtual)	Increment summary event counter
5	Any mode: determine whether any summary events are occurring	Temperatures & voltages of many components, summary event counter (virtual)	Increment summary event counter
(6)	Log alert events	Alert episode counter (virtual), alert event counter (virtual), alert episode timer (virtual)	Increment alert episode counter, increment alert episode timer, store data snapshots, change beacon message
(7)	Log summary events	Summary episode counter (virtual), summary event counter (virtual), summary episode timer (virtual)	Increment summary episode counter, increment critical episode timer, store data snapshots
(8)	Log and respond to critical events	Critical episode counter (virtual), critical event counter (virtual), critical episode timer (virtual)	Increment critical episode counter, increment critical episode timer, store data snapshots, change beacon, shut down payloads, log off users, purge schedule

Many of the operators' comments further emphasized the need for sufficient observability of both the vehicle state and also the automated reasoning approach. The aforementioned programming errors were easier to debug once they had been detected because the table logic mimics the logic used by operators. Furthermore, some initial enhancements to Chatterbox's tables involved displaying the real-time persistence values and latch status for the table entries. This information was very helpful in understanding the inner workings of Chatterbox, and in setting proper persistence and threat limit values.

5. Conclusions & Future Work

The use of persistence in production rule systems already enjoys significant on-orbit experience. It is the authors' belief that further conceptual study and experimentation will enable simple production rules to represent sophisticated automated reasoning techniques. SSDL will continue such explorations through its student-developed vehicles and by research.

Chatterbox will face continued ground-based testing of the beacon monitoring experiment, involving refinements to its anomaly detection algorithm. Once Sapphire is launched, the tables will be involved in long-term beacon monitoring flight experiments. In addition, as time permits the Sapphire team will test non-health-related uses of the tables. For example, a useful feature would be to take a picture when the Earth fills the camera field of view or to capture a sunrise or sunset. The tables can be structured to respond to specific solar panel and Earth sensor outputs by snapping the photo.

In the near future, Chatterbox and its table-based production rule architecture will be used for the next vehicles in SSDL's spacecraft program: Emerald. This multi-satellite mission will demonstrate some basic formation-flying technologies in pursuit of lightning science research [14]. The tables will again participate in on-board anomaly management, incorporating the other observable elements described above.

The "integral of threat" concept has barely been explored, and yet the settings for alert thresholds drive many aspects of spacecraft health monitoring. Further examination of the conceptual basis for these thresholds would greatly assist designers and operators in determining how best to define an anomaly.

Production rules are simple yet powerful tools for translating operational procedures into automated systems. Augmenting these rules with explicit persistence tracking helps to overcome a number of the problems related to the fragility of this reasoning technique. In the area of health monitoring, the use of persistence has been demonstrated on the Sapphire vehicle with significant benefits. These benefits underscore the fact that further study of temporal behavior will improve the performance of space mission operations.

Acknowledgements

The authors would like to thank their colleagues in the Space Systems Development Laboratory, especially the past and current members of the Sapphire development team, the Chatterbox developers, and the beacon testing team. Special thanks go to Professor Robert Twiggs for his continued dedication to this laboratory in the role of lab director. The authors would additionally like to thank the conference reviewers whose comments have been very helpful in constructing this paper. This work was performed in partial satisfaction of graduate studies at Stanford University.

References

[1] Fesq, Lorraine, Abdullah Aljabri, Christine Anderson, Robert Connerton, Richard Doyle, Mark Hoffman, and Guy Man, "Spacecraft Autonomy in the New Millenium," *Guidance & Control 1996: Advances in the Astronautical Sciences*, volume 92, pp. 3-20.

[2] Thurman, David A., David M. Brann, and Christine M. Mitchell, "An Architecture to Support Incremental Automation of Complex Systems," *Proceedings of the 1997 IEEE Conference on Systems, Man, and Cybernetics*, Orlando, FL, 12-15 October 1997, pp. 1174-1179.

[3] Leveson, Nancy G., and Everett Palmer, "Designing Automation to Reduce Operator Errors," *Proceedings of the 1997 IEEE Conference on Systems, Man, and Cybernetics*, Orlando, FL, 12-15 October 1997, pp. 1144-1150.

[4] Johnson, Steven, "Fault Protection Design for Unmanned Interplanetary Spacecraft," *Guidance & Control 1987: Advances in the Astronautical Sciences*, volume 63, pp. 3-18.

[5] Brown, Mark, Douglas E. Bernard, and Robert D. Rasmussen, "Attitude and Articulation Control for the Cassini Spacecraft: A Fault Tolerant Overview," *Proceedings of the 14th AIAA/IEEE Digital Avionics Systems Conference*, Cambridge, MA, 5-9 November 1995, pp. 158-163.

[6] Twiggs, Robert J., and Michael A., Swartwout, "SAPPHIRE - Stanford's First Amateur Satellite", *Proceedings of the 1998 AMSAT-NA Symposium*, Vicksburg, MS, 14-16 October 1998.

[7] Grade J., A. Barzilai, K. Reynolds, C.H. Liu, A. Partridge, H. Jerman, and T.W. Kenny, "Wafer-Scale Processing, Assembly, and Testing of Tunneling Infrared Detectors," *Proceedings of Transducers '97 - The 9th International Conference on Solid State Sensors and Actuators.*

[8] Kitts, Christopher A., and Michael A. Swartwout, "Experimental Initiatives in Space Systems Operations", *Proceedings of the Annual Satellite Command, Control and Network Management Conference,* Reston, VA, 3-5 September 1997.

[9] Swartwout, Michael A., Carlos G. Niederstrasser, Christopher A. Kitts, Raj K. Batra, and Ken P. Koller, , "Experiments in Automated Health Assessment and Notification for the SAPPHIRE Microsatellite", *Proceedings*

of SpaceOps '98: The 5th International Symposium on Space Mission Operations and Ground Data Systems, Tokyo, Japan, 1-5 June 1998.

[10] Niederstrasser, Carlos G., Christopher A. Kitts and Michael A. Swartwout, "A Beacon Receiving Station for Automated Health Operations," *Proceedings of the 1999 IEEE Aerospace Conference*, Snowmass, CO, 3-10 March 1999.

[11] Batra, Raj K., "The Design of a Highly Configurable, Reusable Operating System for Testbed Satellites", *Proceedings of the 1997 AIAA/USU Conference on Small Satellites*, Logan, UT, 15-18 September, 1997.

[12] Sue, Miles K., Robert Kahn, Gabor Lanyi, Victor Vilnrotter, E.J. Wyatt, and Ted Peng, "Spacecraft Beacon Monitoring for Efficient Use of the Deep Space Network," *Proceedings of the 48th Annual International Astronautical Federation Congress*, Turin, Italy, 6-10 October 1997.

[13] Niederstrasser, Carlos G., "Development of a Satellite Beacon Receiving Station", *Proceedings of the 12th Annual AIAA/USU Conference on Small Satellites*, Logan, Utah, 31 August – 3 September 1998.

[14] Kitts, Christopher A., "EMERALD: A Low-Cost Spacecraft Mission for Validating Formation Flying Technologies," *Proceedings of the 1999 IEEE Aerospace Conference*, Snowmass, CO, 3-10 March 1999.

Michael Swartwout *is a Doctoral Candidate in Aeronautics & Astronautics at Stanford University. His research is in spacecraft operations, specifically engineering data summarization. Mr. Swartwout has served as Project Leader for the Sapphire spacecraft since 1994 and is a co-investigator of the Beacon-Based Health Monitoring Experiment. He is involved in SSDL's Automated Space Systems Experimental Testbed (ASSET) operations research system, developing the system architecture and managing the blackboard tool that forms the basis of ASSET. Mr. Swartwout earned a BS in Aeronautical & Astronautical Engineering from the University of Illinois at Urbana-Champaign in 1991, and a MS in the same, from the same, in 1992.*

Christopher Kitts *is a Doctoral Candidate in Stanford University's Design Division where he specializes in spacecraft design and command and control systems. He holds a joint position as the Graduate Student Director of Stanford's Space Systems Development Laboratory and as the Co-Director of the Santa Clara Remote Extreme Environment Mechanisms Laboratory at Santa Clara University. Mr. Kitts is also a space systems engineer with Caelum Research Corporation at NASA's Ames Research Center where he develops spacecraft design and autonomy strategies for NASA's New Millennium Program. Mr. Kitts has served in the Air Force as a mission controller of and the Chief of Academics for the Defense Satellite Communications System III spacecraft constellation. He has held a research position at the Air Force Phillips Laboratory and has taught numerous graduate courses in space system design. Mr. Kitts received a BSE from Princeton University, an MPA from the University of Colorado, and an MS from Stanford University.*

Raj Batra *is a Doctoral Candidate in the Department of Aeronautics & Astronautics at Stanford University. His research is in visualization of fluid flow. Mr. Batra served on the Sapphire CPU team from 1994 to 1998, and has been manager of the team since 1996. He had primary responsibility for developing the Chatterbox table system. Mr. Batra earned a MS in Aeronautics & Astronautics from Stanford in 1994 and a BS in Aerospace Engineering from the University of Cincinnati in 1993.*