N90-22314

AUTOMATED PROCEDURE EXECUTION for SPACE VEHICLE AUTONOMOUS CONTROL

by Thomas A. Broten and David A. Brown

TRW
One Space Park R2-2062
Redondo Beach, CA 90278

Abstract

Increased operational autonomy and reduced operating costs have become critical design objectives in next-generation NASA and DoD space programs. Our objective is to develop a semi-automated system for intelligent spacecraft operations support. This paper presents the Spacecraft Operations and Anomaly Resolution System (SOARS) as a standardized, model-based architecture for performing High-Level Tasking, Status Monitoring and automated Procedure Execution Control for a variety of spacecraft. The particular focus here is on the Procedure Execution Control module. A hierarchical procedure network is proposed as the fundamental means for specifying and representing arbitrary operational procedures. A separate procedure interpreter controls automatic execution of the procedure, taking into account the current status of the spacecraft as maintained in an object-oriented spacecraft model.

1. Introduction

A new generation of NASA and DoD space vehicles is emerging, for which a high degree of operational autonomy is a fundamental requirement. The requirement is typically expressed as a need for on-board task management and contingency handling for extended periods of time, without direct human involvement [Sobieski, 1989; GSFC, 1988]. The sources of the requirement stem from a) life-cycle cost control through reduction of ground support crew size (eg, for Space Station platform and mission support activities), b) improved spacecraft survivability through reduced dependence on vulnerable fixed-base ground stations (eg, for early warning and communications satellites), and c) operations at extreme distances from the Earth where signal propagation time precludes tightly-coupled human control (eg, for interplanetary probes, rovers). The increased levels of space vehicle autonomy now being proposed will substantially exceed the capabilities of traditional space/ground command link operations. The next level of autonomy will be termed the "task level" here, to distinguish it from the "command level" and to signify independent execution and control of a complete operational procedure upon receipt of an external tasking order or an internal alarm message.

Raising space vehicle autonomy to the task level involves advances in a number of areas. First, independent procedure execution means more than simply executing a prestored sequence of commands. The structure of the procedure must contain interim tests of the current situation to assure that the previous step was performed correctly before the next step is begun; otherwise, equipment damage or unsafe conditions could result. The procedure must also incorporate basic contingency handling routines, to avoid leaving the space vehicle in an undesirable state in the event of premature procedure termination.

Second, there must be facilities for on-board monitoring of the current situation that are tied-in to the space vehicle's baseline telemetry and command system. Automated real-time interpretation of data is necessary when direct human supervision is absent or delayed. Recent experiments with expert systems for automatic detection/diagnosis of anomalies, such as SCARES [Hamilton, 1986], SHARP [Lawson, 1989], SFMS [Parks, 1990], and StarPlan [Siemens, 1986] are gradually expanding the capability for real-time situation assessment.

Third, true high-level tasking implies the existence of a suitable tasking language in which a task (or operational goal) can be precisely and unambiguously expressed. Additional AI planning facilities for breaking down the high-level task into a coordinated procedure containing primitive spacecraft commands and programmed status checks will be required in the long-term. Space vehicle autonomy can benefit directly by borrowing some of the new robot task planning languages and architectures [Fu, 1987; Albus, 1987].

2. SOARS Architecture

The Spacecraft Operations and Anomaly Resolution System (SOARS) is a model-based prototype for supporting autonomous spacecraft operations. As shown in Figure 1, SOARS represents an add-on capability for incorporation into any space

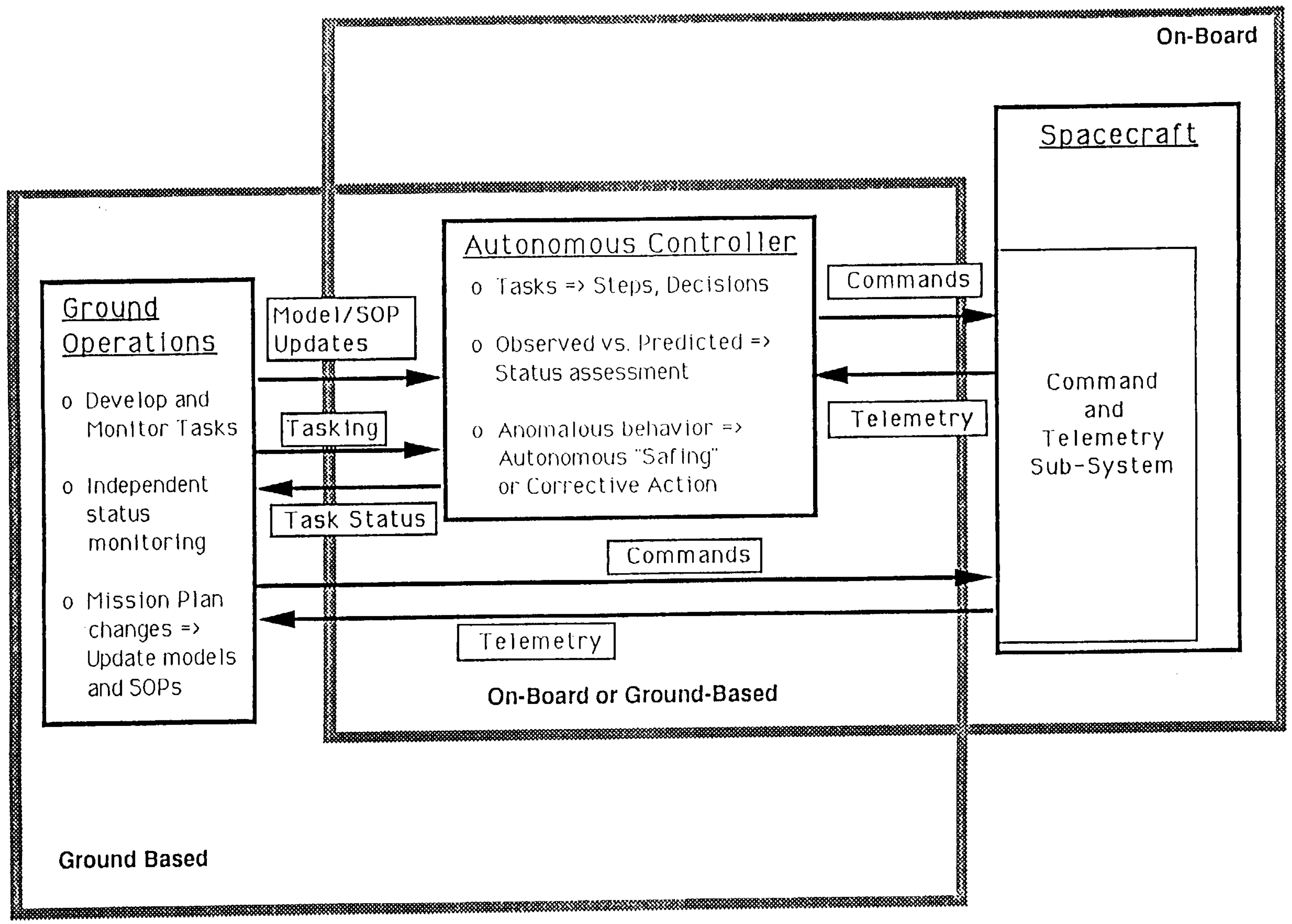

On-Board
Spacecraft
Command and Telemetry Sub-System
Autonomous Controller
o Tasks => Steps, Decisions
o Observed vs. Predicted => Status assessment
o Anomalous behavior => Autonomous "Safing" or Corrective Action
Commands
Telemetry
Ground Operations
o Develop and Monitor Tasks
o Independent status monitoring
o Mission Plan changes => Update models and SOPs
Model/SOP Updates
Tasking
Task Status
Commands
Telemetry
On-Board or Ground-Based
Ground Based

Figure 1. SOARS Architecture

vehicle/ground station control loop with standardized bus and processor interfaces to command & telemetry data. Under traditional nonautonomous operations, commands are transmitted from the ground on the uplink, and routed to the appropriate space vehicle subsystems for execution. The space vehicle subsystems continuously report status, which is encoded in telemetry for transmission to the ground on the downlink. Under upgraded autonomous operations, additional commands dealing with high-level tasking and operational procedure control can be sent to the space vehicle. Correspondingly, task status and autonomous processing performance are reported back to the ground. (Note that some additional command and telemetry slots must be allocated above the space vehicle baseline to cover autonomous operations.) For the initial demonstration flights, all of the SOARS functions would reside on the ground to permit thorough open-loop validation of the control processes. Subsequently, many of those functions would migrate to the space vehicle for true autonomous operations in the closed-loop mode.

The target SOARS flight segment, called the Autonomous Controller Unit or ACU, is composed of three basic elements, which are shown in Figure 2. The Task Analyzer receives high-level tasking commands from the ground segment (via the command subsystem), and decomposes the task into a coordinated program of primitive commands and procedural checkpoints. The Status Monitor receives status data from the space vehicle subsystems, and detects the occurrence of specified vehicle states indicating expected (ie, planned) or unexpected (ie, anomalous) conditions. The Procedure Executor executes a tasking program (received from the Task Analyzer), or a corrective action procedure (based on the detection of an anomalous condition by the Status Monitor) under real-time control.

The remainder of this paper focuses on the design and operation of the Procedure Executor component. Details of the preliminary design of the other components are contained in the ACU specifications [Brown, 1989]. The Procedure Executor interfaces with a simulated ground segment workstation called the Ground Tasking Interface, through which the operator can directly program and supervise autonomous operations. The prototype implementation runs on a Symbolics 3645 with the human/computer interface and spacecraft model (the DSP satellite) implemented in the Knowledge Engineering Environment (KEE) from Intellicorp and the Procedure Executor implemented in Common Lisp.

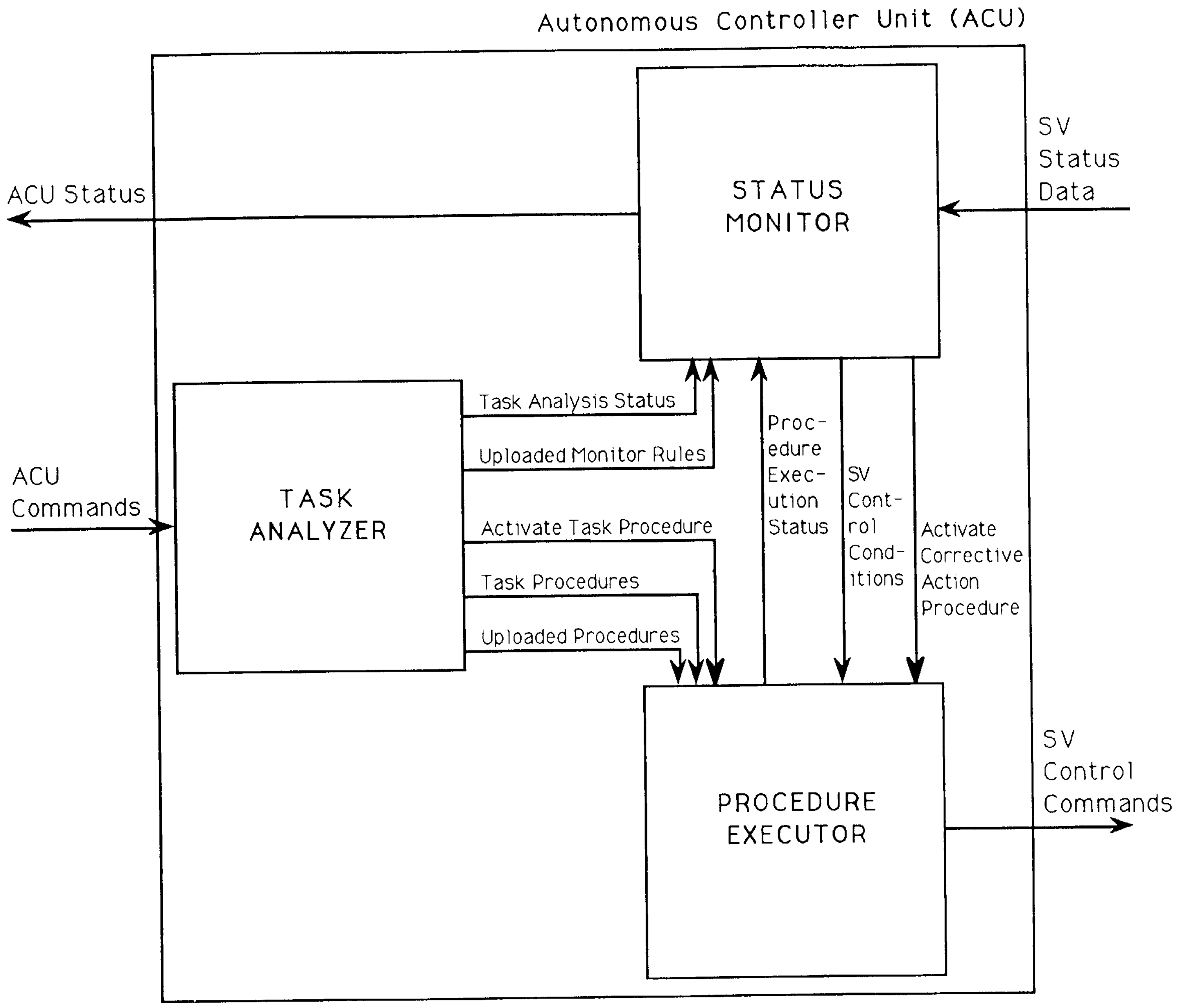


Figure 2. SOARS Flight Segment

3. Procedure Development

Figure 3 shows a sample operational procedure from the Defense Support Program (DSP). The procedure specifies how to attempt recovery from the loss of the second communications link. (For the DSP satellite there are three links total, each with a different recovery procedure. Manual procedures such as this are normally developed during detailed design and delivered as standard operating procedures [Stager, 1987].)

The procedure is entered at the top and execution follows a path through the nodes until an exit point is reached. The procedural steps specify tests to be performed (diamonds), actions to be taken (boxes), or exit points (circles). The arrows specify possible test results or simple continuations following an action. The tests and actions in some cases are primitive (eg, 'Is the L2 carrier present?' or 'Command lo bit rate switch to mode 50'); and in other cases complex (eg, 'Verify L3' -- which may involve running the complete procedure for the third link).

In the demonstration scenario a corresponding procedure graph is created on the ground using the interactive facilities in the Ground Tasking Interface (GTI). A portion of the Link-2 procedure graphic is shown in the large window of the GTI display in Figure 4. Once created by the operator, the procedure graphic is automatically processed and translated into an efficient internal representation suitable for automatic execution.

4. Procedure Representation

There are various possible ways to internally represent a space vehicle operational procedure for computational purposes. One straightforward representation is as a directly executable computer program. In this approach the program will consist of a conditional branch statement (**if** <condition> **then** <step i> **else** <step j>) or a case statement (**case** <condition> A: <step i> B: <step j> C: <step k> ....) for each procedural step. In other words, choose the next step depending on the outcome of the conditional test, until an exit point is reached.

This simple approach, however, has significant drawbacks. First, highly nested **if ... then ... else** or **case** statements are messy and notoriously difficult to understand and debug. For the sample procedure above, the longest path would require nesting to 16 levels. Second, there is no straightforward way to handle repeated steps in the procedure. Common subpaths must be represented separately and redundantly, unless **goto** statements are included (thereby overriding the natural program

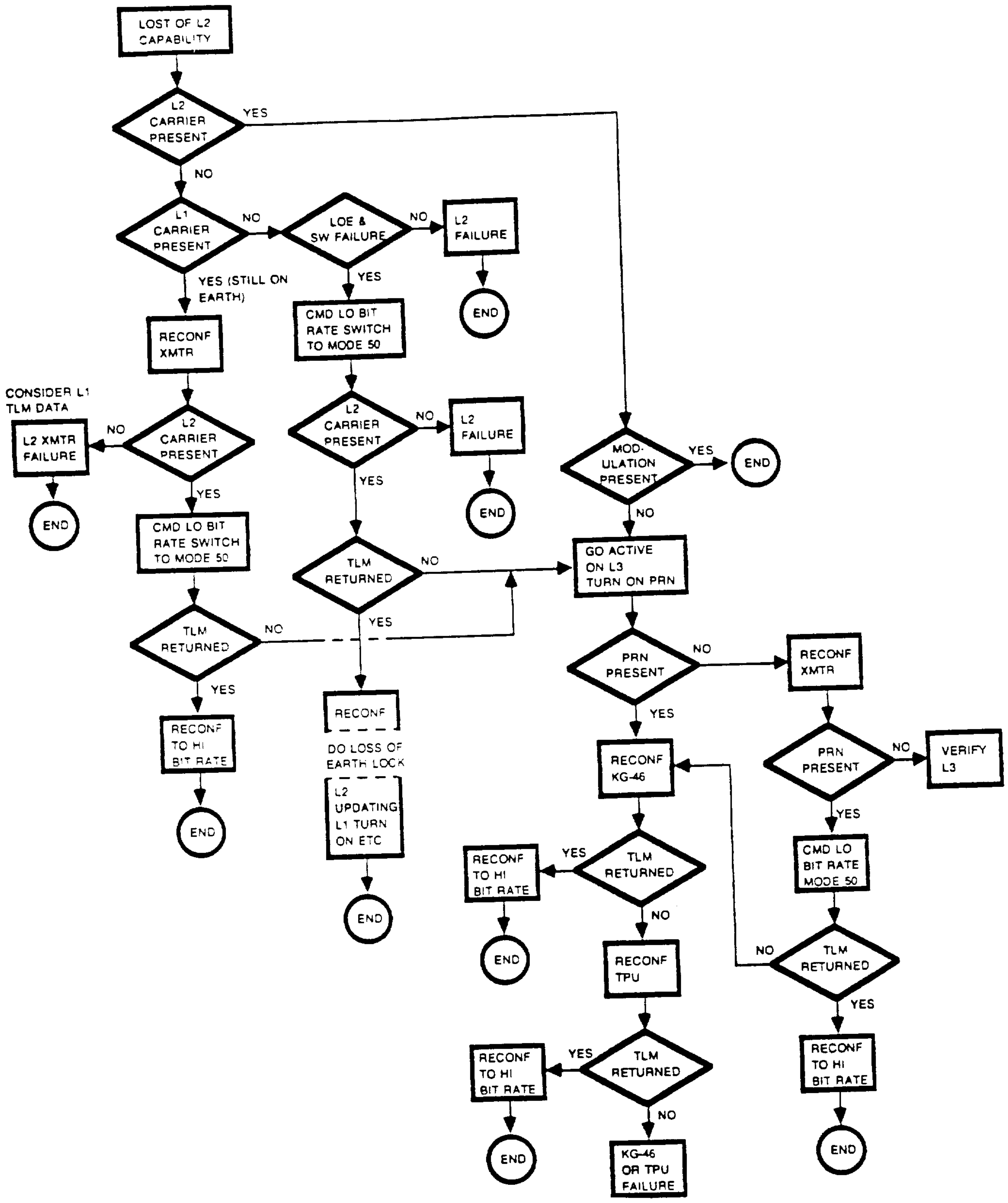
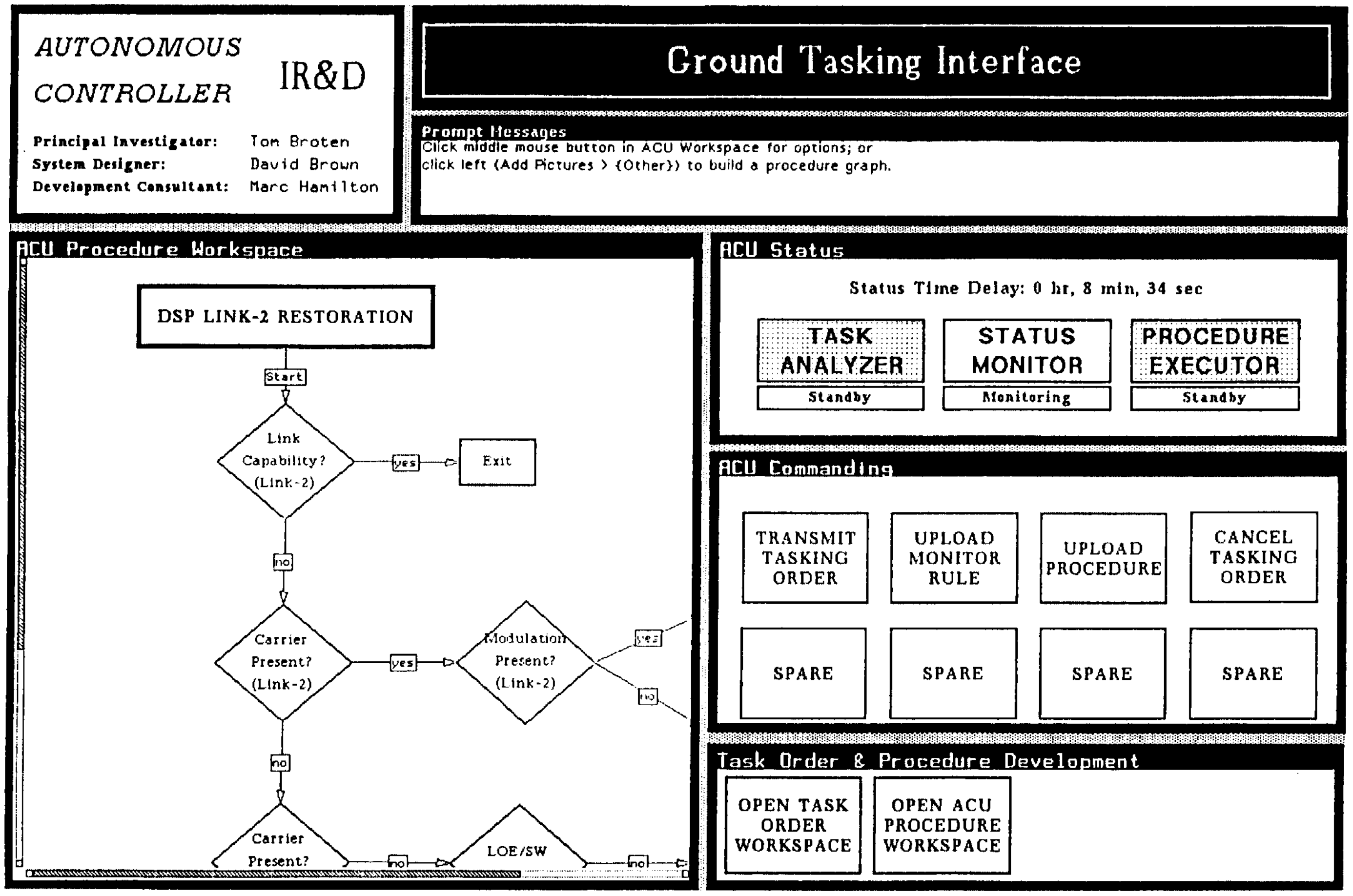


Figure 3. DSP Link-2 Diagnostic Procedure


AUTONOMOUS CONTROLLER IR&D

Principal Investigator: Tom Broten
System Designer: David Brown
Development Consultant: Marc Hamilton

Ground Tasking Interface

Prompt Messages
Click middle mouse button in ACU Workspace for options; or
click left (Add Pictures > (Other)) to build a procedure graph.

ACU Status

Status Time Delay: 0 hr, 8 min, 34 sec

TASK ANALYZER: Standby
STATUS MONITOR: Monitoring
PROCEDURE EXECUTOR: Standby

Figure 4. SOARS Operator Interface

flow). Repetitive procedural cycles are also a problem for the same reason, with the added drawback that the depth of recursive calls is not predictable before the procedure is run.

Another possible representation for a procedure is as a collection of rules. In this approach each procedural step would be encoded in a separate **if ... then ...** rule. The rule-based representation permits a high degree of modularization of the procedural contents that is easy to understand and maintain. Which rule fires next is decided by the current situation at any time, thus no explicit procedure actually exists.

The problem with this representation is that the rules would need to implicitly encode control knowledge as well as diagnostic knowledge in order to maintain the required flow of the procedure. Mixing the two kinds of knowledge always diminishes the clarity of a rule, and defeats the original objective of rule-based systems, which is to keep these knowledge types separate. Also, not having an explicit representation of a procedure makes it difficult to reason about the procedure itself.

In keeping with the DSP example, a procedure also can be formally represented as a directed (possibly cyclic) graph. This representation appears to overcome most of the problems with the other representations while retaining their advantages, at the expense of some additional software for an interpreter (see below).

The nodes in the graph correspond to tests of various status conditions derived from the continuously updated space vehicle database model, and the arcs correspond to procedural control decisions which are followed depending on the outcome of performing a node test. Space vehicle control actions (ie, commands) are handled as side effects associated with some nodes. The actual execution of a procedure is characterized by a sequence of procedural steps (ie, a path) through the graph that provides a diagnostic record of the specific tests performed, decisions made, and actions taken. The individual nodes in a procedure graph may themselves be procedures. A hierarchical procedural representation allows complex procedures to be built up from more primitive tests and actions. For example, a Mars lander suddenly switching to an alternate landing site during the descent phase may involve resetting a number of different control subsystems, all of which possess their own reset procedures.

Moreover, the formal graph representation permits a natural separation of control knowledge (ie, what to do next) from test and actuation knowledge (ie, how to do it). The control knowledge can be stored in the procedure representation where it belongs, while the test and actuation knowledge can be placed directly in the space vehicle model which the procedure is testing and actuating.

The above definition of a procedure as a directed graph generally allows for the representation of arbitrarily complex procedures; eg, it includes procedures with provisions for contingency handling, shared substeps, repetitive cycles, interim status reporting, and even delayed operator enabling of critical steps, if necessary. It also enables high-level (AI search) analysis of the declarative procedural form using graph traversal techniques. High-level procedural analysis methods are required by the SOARS Task Analyzer component, which is responsible for selecting, adapting and generating procedures to fulfill ground tasking orders.

5. Execution Control

A procedure graph in SOARS is implemented as a hash table. The hash table allows direct access to any node given that node's identifier name. The content of a node, in turn, stores the name of a primitive function which is to be executed when that node is activated (eg, a command or conditional test), and which node to branch to next given the real-time result of the command or test. A hash table is automatically created for a procedure after the operator develops the graphical representation of the procedure using KEE's interactive interface facilities .

The execution of a procedure graph (whether for verification purposes on the ground or for operational purposes on the spacecraft) is controlled by an interpreter module. The procedure interpreter takes as input the hash table representation of a procedure. It begins execution at the designated starting node and follows a path through the procedure graph. To execute a node, the interpreter looks up the name of the primitive function associated with the node in a telemetry/command function library. The library contains the primitive functions for performing all of the basic telemetry tests and command sequences for the spacecraft. The library, which will be different for each spacecraft, serves as the principal interface between SOARS and the spacecraft via the provided command & telemetry subsystem. The library functions are written in the Tell and Ask (tm) pseudo-English language provided with KEE to simplify the interface with domain experts. After executing the appropriate function for the node and receiving the returned results, the interpreter steps to the next node based on those results.

Procedure execution stops when a termination node is reached.

Throughout the execution of a procedure and at critical designated checkpoints, the Procedure Executor reports status back to the Ground Tasking Interface and to the on-board Status Monitor. The operator and Status Monitor together exercise external (high-level) supervisory control over the execution process.

6. Directions for Future Work

Our ultimate goal is to field an integrated system for intelligent spacecraft operations support, including a standardized SOARS shell and integrated Knowledge Capture Tools (KCT). The KCT developments of the prior year are targeted to support spacecraft design, integration & test, and operations through the coordinated collection and refinement of analysis and simulation models. The KCT comprises the off-line knowledge-base design component, while the SOARS shell represents the on-line operational component of the integrated system.

During the coming year we intend to produce a working prototype of KCT/SOARS and release it to one or more project teams for use in creating spacecraft models and evaluating simulated autonomous control. We plan to develop demonstrations that will focus on the Advanced Tracking and Data Relay Satellite (ATDRS) and a Polar Orbiting Platform (POP). The procedural control and high-level tasking elements of this year's development will be combined with the telemetry monitoring elements of the Spacecraft Fault Management System, or SFMS, (a currently operational real-time expert system) to complete the prototype SOARS architecture. The current SFMS, which is based on symptomatic analysis of fault conditions, is being upgraded to handle model-based fault diagnosis.

Once developed, KCT/SOARS will provide spacecraft designers and operators with an integrated prototyping environment for capturing and then using spacecraft data and models. The initial prototyping environment is specifically aimed at a) generating simple, well-structured models which can be later expanded; b) diagnosing a small number of faults to a level where a certified automatic procedure can reliably rectify or evaluate the condition; and c) providing a straightforward capability for high-level tasking (ie, simple on-board procedure activation) from the ground. Fully automated corrective action procedures, in general, will not be provided; instead, automated control will be limited to the execution of simple diagnostic procedures capable of providing backup information. We believe a careful, gradual approach to increased levels of satellite autonomy is the only viable approach, considering the potential consequences of precipitous actions.

Cited Works

Albus, J.S., McCain, H.G., Lumia, R.L., NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), National Bureau of Standards, 13 Mar 87.

Brown, D.A., Software Requirements Specification for a Space Vehicle Autonomous Controller, TRW Memo, 23 May 89.

Brown, D.A., Autonomous Controller Block Diagrams, TRW Memo, 31 May 89.

Fu, K.S., Gonzalez, R.C., Lee, C.S.G., Robotics: Control, Sensing, Vision, and Intelligence, McGraw-Hill, New York, 1987.

Goddard Space Flight Center (GSFC), Flight Telerobotics Servicer (FTS) Requirements Document, 1988.

Hamilton, M., Dignam, F., Murrin, J., "SCARES: A Spacecraft Control Anomaly Resolution Expert System", Proceedings of Expert Systems in Government Symposium, IEEE, Oct 86.

Lawson, D.L., James, M.L., "SHARP: A Multi-mission AI System for Spacecraft Telemetry Monitoring and Diagnosis", Goddard Conference on Space Applications of Artificial Intelligence, May 1989.

Parks, K.G., Broten, T.A., Rathe, R.A., "SFMS: An Expert System for Spacecraft Operations", Second Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-90), May 1990.

Siemens, R.W., Golden, M., Ferguson, J.C., "StarPlan II: Evolution of an Expert System", Proceedings Fifth National Conference on Artificial Intelligence (AAAI-86), Aug 86.

Sobieski, S., Customer Data and Operations System (CDOS) Operations Concept Document Version 1.0, Goddard Space Flight Center, Jun 1989.

Stager, D.C., Satellite Contingency Analyses and Schematics: Satellites 0014 - 0017, TRW Report # 45021-321-710A3-018, 2 Mar 87.

Special Topics