PXI: A Space Shuttle Mission Operations Knowledge-Based System Project

Matthew R. Barry

Rockwell Shuttle Operations Company
NASA/Johnson Space Center, DF63
Houston, TX 77058

Abstract
This paper presents an examination of a knowledge-based system (KBS) prototyping project. The work entails reasoning about the impact of component failures in Space Shuttle Orbiter subsystems. The effort is directed toward the development of a system that will recognize passive component failures as potential safety hazards, rather than toward an active failure identification or diagnostic tool. Flight controllers in the Mission Control Center (MCC) at the Lyndon B. Johnson Space Center (JSC) will use this prototype to help develop requirements for a Space Shuttle mission operations tool.

1 Introduction
The most critical assignment for a flight controller is maintenance of the astronaut crew’s safety. He or she is always responsible for being cognizant of the condition of the spacecraft. Fulfillment of this obligation is particularly arduous when Orbiter subsystems sustain failures. This process entails the rapid recognition and evaluation of a component failure and the assessment of the functional impact to the crew, spacecraft, and mission. Flight controllers train to manage most single component failures from memory. The failure management process is more complex, however, when it involves combinations of two or more components.

The PXI project concerns two of the Orbiter subsystems: the orbital maneuvering system (OMS) and reaction control system (RCS). These propulsion systems perform orbit adjustment maneuvers and rotation and translation maneuvers, respectively. The OMS and RCS present interesting development areas, as they operate with electrical and mechanical redundancy and they interface with several Orbiter subsystems. This domain, nonetheless, is bounded, well-defined, and involves only a small number of operations.

2 Considerations

2.1 Scope
PXI provides mechanisms for (1) annunciating mission rule and hardware constraint violations, and (2) evaluating proposed subsystem reconfigurations. When a failure occurs, the flight controller immediately takes action based on the observed failure. Diagnosis of the problem is made only when time permits after attaining an approved configuration. PXI is not intended as a diagnostic tool.

2.2 Preprocessing
Most component failures are readily discerned. Undetected failures of active components are rare. The spacecraft systems announce most of the high-impact single failures. PXI uses this onboard processing and the output of ground-based fault-detection processes as its input interface [5,6]. Input consists of the assertion of an explicit component failure.

2.3 Postprocessing
The output of the system specifies mission rule and operational constraint violations. The program will provide recommendations for or against subsystem reconfigurations if available.

An assumption-based truth maintenance system (TMS) similar to de Kleer’s generalized diagnostic engine [7] could eventually be used to perform diagnostics. Such systems, which handle competing diagnoses and unexplained failures, will be required for programs that deal with multiple failures. This type of TMS assumes, however, that sequential testing of components can be performed directly. This, unfortunately, is not the case in spacecraft.

Most of the troubleshooting procedures are already defined [8]. The work of Georgeff [9] has demonstrated the feasibility of a diagnostic system that reasons about these procedures. The system is reactive and capable of prioritizing activities. Such a program can be valuable in bridging the gap between the immediate actions and diagnostic analysis.*

2.4 Knowledge acquisition
Accurate evaluation of failure scenarios requires detailed knowledge of the hardware and software components. The knowledge acquisition process has been trivial, since the experts are producing this project. Several published sources supply the requisite knowledge. The vehicle schematic drawings [1] define the hardware components and furnish the intercomponent relationships. Hardware performance characteristics [2] and software design specifications [3] provide information about how these components may be operated. The “STS Flight Rules” [4] provide failure definitions and dictate approved modes of operation (reference figure 1). Heuristic knowledge, perfected through testing, training, and experience, is manifested as recommended responses to failure scenarios.

3 Design

3.1 Overview
The PXI system is similar to a justification-based TMS. Observed failures are asserted and propagated through a reasoning process.

*Flight controllers have been participating in the development of an RCS prototype using Georgeff’s Procedural Reasoning System in order to demonstrate its applicability.
lattice to uncover unobserved failures. An element may be either failed or operational (analogous to in and out in other TMS's).

The reasoning process is therefore nonmonotonic. A declarative semantic construction permits incremental development and facilitates verification.

<table>
<thead>
<tr>
<th>PROX OPS GMC SYSTEMS MANAGEMENT - Continued</th>
</tr>
</thead>
<tbody>
<tr>
<td>The loss of rotation, translation in any axis, and the loss of all contacts in any axis of the flight controllers would require the termination of the rendezvous operation until system reconfiguration can be performed to regain the lost capabilities.</td>
</tr>
</tbody>
</table>

B. THE FOLLOWING CAPABILITY IS REQUIRED TO CONTINUE PROX OPS

1. PREC ROTATION AND TRANSLATION CAPABILITY IN EACH AXIS. JET REQUIREMENTS ARE AS FOLLOWS:
   a. +X - 1 LEFT FIRING JET IN EACH AFT POD, GROUP (7 & 8)
   b. -X - 1 FORWARD FIRING JET (GROUP 1)
   c. +Y - 1 LEFT FORWARD AND 1 LEFT AFT JET (GROUP 2 & 9)
   d. -Y - 1 RIGHT FORWARD AND 1 RIGHT AFT JET (GROUP 3, 10)
   e. +Z - 1 DOWNFIRING JET EACH GROUP (5, 6, 13, 14)
   f. -Z - CONSISTING OF:
      - NORMAL Z: 1 UPFIRING JET EACH POD, GROUP (4, 11, 12)
      - LOW Z: 2 FORWARD FIRING JETS AND 1 AFT FIRING JET EACH POD, GROUP (1, 7, 8)

NOTE:
0. IF OBSTACLE IN ACTIVE ATTITUDE HOLD, ONLY 1 JET TOTAL IS REQUIRED, BUT PROPELLANT CONSUMPTION IS GREATER.

PROX OPS, similar to rendezvous, also requires rotation and translation control in all axes. The number of jets required for each group is specified. Any jet from each group will meet the minimum jets required for control.

3.2 Knowledge representation

PXI admits two varieties of knowledge bases. The independent knowledge base contains those frames that are invariant with mission phase, such as the hardware configuration. These instantiations are applicable to all possible requirements specifications, and contain only the most generic requirement frames. The dependent knowledge base, on the other hand, contains phase-dependent or mission-unique knowledge. The preprocessor incorporates the dependent knowledge bases into the independent knowledge base at run time. There may be several dependent knowledge bases available, any combination of which may be used in the reasoning process.

A frame-based formulation consisting of class and instance frames structures the knowledge. Each frame contains slots that identify its characteristic properties. These slots contain data or procedures. Specification of these two types of frames renders a convenient slot-inheritance mechanism. This mechanism establishes unknown characteristics of a component by borrowing this information from ancestor classes.

Knowledge ascribed to the class frames consists of generic characteristics applicable to each member of a particular category. Slots within the class frames contain method parts for configuration appraisal and interfaces with the inference engine. The configuration appraisal methods provide techniques for evaluating a component configuration. These methods identify redundant component paths and verify satisfaction of operational requirements to assure that functional capability is available and that the system configuration is satisfactory. Each class is in turn a member of a metaclass. A metaclass characterizes a class as a component, a function, or a requirement. A component describes a particular entity, such as a thruster or an effector or a perceptor. A function describes a software process that controls the components, such as a digital autopilot that determines when to fire the thrusters or deflect the aerosurfaces.

A requirement is an operational constraint and mission rule. Instances of this class specify minimum redundancy levels and approved hardware configurations for particular mission phases. These instances represent the "goal states." Satisfaction of these requirements is of fundamental importance to the flight controller. The program alerts the user upon violation of any requirement.

The instance frames refine the characteristics attributed to them through their class membership. The interesting traits of the instance frames are their status evaluation functions and their dependency lists. Each instance frame contains a Boolean function that may be called to determine the operational status of that frame. This function usually investigates the operational status of those elements upon which it depends. If the function returns false, the frame may be considered failed. The Requires slot enumerates these upstream dependencies. In the current implementation of PXI, the status evaluation function produces the dependency list directly.

A lattice of the domain components is constructed from the names of the instance frames and their dependency lists (reference figure 2). The terminals on this lattice are members of the requirement class. The Preprocessor Utility, (described below) fills a Supplies slot, which enumerates the downstream connections of an element, while building this lattice.

![Figure 1.- Dictation of a typical requirement.](image1)

![Figure 2.- A typical reasoning lattice.](image2)
3.3 Inference engine

Application of object-oriented programming techniques separates the problem domain knowledge from the inference engine. It consists of four utilities used to reason within the entire spectrum of failure domains, and a message passing mechanism. These features are described below.

The Preprocessor Utility prepares a knowledge base for incorporation into the reasoning environment. This operation involves the creation of forward (causal) links from the backward (dependent) links already specified in each element. Recall that these links specify those frames required by the status evaluation function. To accomplish this, the preprocessor creates a queue of requirement nodes while loading a dependent knowledge base. For each element of this queue, the preprocessor collects the names of the frames in each Requires slot. It places the name of the current element into the Supplies slot of each of the elements in its Requires slot, then appends these "touched" elements to the end of the queue. Upon deprecation of the queue, the reasoning environment consists of a doubly-linked lattice of elements. This process merges the dependent and independent knowledge bases into one reasoning environment.

The requirement frames in a given dependent knowledge base may not generate links to all the elements specified in the independent knowledge base. Therefore, some of the frames, possibly entire sublattices, may not contain forward links in their Supplies slots. These unused nodes do not, however, represent errors in the knowledge base. Removing them from the lattice conserves memory and reduces searching time.

The Failure Assertion Utility governs the dominant mode of operation. It manages component failure assertions and retractions (explicit assertion of a function or requirement is not permitted). A failure assertion propagates through the lattice by visiting recursively each element in the failed element's Supplies slot. Each element considers the explicit failure through its status evaluation function. Any element downstream of the explicit failure which determines that it is also failed (by failing its status evaluation function), becomes an implicit failure. The failure propagation process will repeat until there are no new implicit failures. Implicit failures in requirement metaclass elements are collected and displayed after each change of state. Instances of other classes report their status upon inquiry.

The inference engine maintains explicit failures until retracted. The retraction process is the same as the assertion process. Herein lies the nonmonotonicity of PXI. Retraction of an explicit failure removes that element from the failure list, thus reducing by one the number of explicit failures from the previous propagation run.

The Failure Assertion Utility recognizes two types of failure states: actual and hypothetical. The actual mode represents the real component failures and any implicit function failures or requirement violations. The hypothetical mode represents temporary component failures asserted in the current knowledge base state. These assertions generate requirement violations based on the postulation that a failure may occur in the future. This mode of reasoning is particularly useful when investigating proposals for subsystem reconfigurations. Both modes use the same failure propagation scheme. Data-dependency notes label each assertion with the reigning mode of operation.

The Search Utility investigates next-worst component failures and least-impact subsystem reconfigurations. The next-worst evaluation scheme at present considers all requirements to have the same value. This will be altered in the future to include weightings of frame importance, failure frequency, and so forth. These weights will have to adapt to mission phase, yet can be "trained" by the experts. The least-impact evaluation scheme considers redundant paths available for recovering an element, judging them by requirement violations that may occur if reconfigured. The Failure Assertion Utility will automatically trigger a least-impact search if a failed frame contains a reconfiguration procedure. This reconfiguration will be investigated and evaluation of this alternative presented to the user.

The Explanation Utility handles output processing. Method parts assembled from instance, class, and metaclass membership provide a mechanism for reporting failures and recovery recommendations. The lattice links leading to requirement violations and function failures may also be displayed, providing insight into the complete failure path.

The Message Passing protocol in PXI is polymorphic. This feature facilitates interclass communications. The inheritance scheme primarily obtains message-passing methods rather than component information. Default frames and daemons are provided for each class to render generic method parts for each message category.

4 Evaluation

4.1 Problem

The OMS and RCS present interesting development areas as they operate with mechanical, electrical, and functional redundancy. They also require a number of interfaces with other Orbiter subsystems. The reasoning domain, nevertheless, is bounded, well-defined, and involves only a few operations. As Prerano explains [10], such domains are often satisfactory for experimental purposes.

The OMS/RMS independent knowledge base is defined in some detail (over 500 frames). Factual knowledge represents information from several flight control disciplines, including the electrical power; data processing; instrumentation; and guidance, navigation, and control subsystems.

Several dependent knowledge bases have been developed. The first consists of the minimum RCS thruster requirements for continuation of rendezvous and proximity operations. The rules governing this flight phase [4] are clearly defined (see figure 1) and represent a critical rule set based on the lowest level components. Other dependent knowledge bases involve requirements for external tank separation maneuvers and requirements for vehicle rotation control during supersonic atmospheric flight. These requirements are similarly well documented.

4.2 Example

Figure 2 displays part of a typical reasoning lattice. This lattice incorporates rule 2-61-B-1a from the rendezvous and proximity operations Flight Rules (see figure 1). FR-2-61-B-1a states that PLUS-X translation capability is a requirement; the MANEUVER instance PLUS-X states that both autopilot GROUP-7 and GROUP-8 thrusters are required for this function; GROUP-7 functionality requires either of thrusters R1A or R3A; and so on.

The lattice shows that at least two component failures are necessary to violate FR-2-61-B-1a (for example multiplexers demultiplexers (MDM's) FA1 and FA2). Onboard systems detect MDM failures, so they are readily apparent. The
MDM's supply current to the drivers that fire the thrusters. The loss of translation capability therefore might not be detected until a maneuver is attempted.

PX1 detects these situations by propagating the failure of FA1 and FA2 through the lattice to all elements downstream, and provides recommendations to regain lost capability. In this example, the system will determine that +X translation capability has been lost and will recommend that a recoverable MUM be connected to an alternative data path to regain capability to command one thruster from each autopilot group.

4.3 Utility

The system is useful as it exists today. It is a standalone program in which the user provides all input information. The interface between the fault-detection processes and PXI will be established in the near future. Development of dependent knowledge bases for other flight phases is being performed. Moreover, enhancement of the independent knowledge base to reflect minute sub-system details is feasible.

The likelihood of acceptance for a system like PXI will depend upon its degree of assimilation into the existing operational environment. Validation of the knowledge bases has already been accomplished. Once the system is integrated with the real-time fault detection processes, evaluation of the complete prototype can begin. Mission simulation, procedure development, and operator training sessions provide valuable time for testing within typical environments. The philosophy governing this activity is that it is more important that PXI not supply incorrect information than that it supply correct information.

5 Summary

The PX1 project has been instrumental in the development of techniques for reasoning about failures in redundant path hardware systems. The system described was designed to be integrated as a knowledge-based processing system utilizing input from a procedure-based processing system. Implementation of the consolidated system will occur when real-time telemetry data is available at the workstations. The system may be used standalone in the meantime.

The heightened awareness of KBS capabilities has been the most significant accomplishment of recent MCC software prototypes. They have encouraged internal analyses of flight control operations and inspired creative ideas for better software products. The MCC Upgrade [11] and similar efforts at JSC are dedicated to the enhancement of tools available at flight control consoles. Prototyping projects aimed at real-time KBS environments are playing a very important role in the development of detailed requirements for these activities.

Acknowledgements

The author gratefully acknowledges N. W. Hale, L. J. Hautringer, and L. E. Toole for their support of this project, and Omniiplan Corporation for preparing this document.

References