SRI International

Real-Time Reasoning: The Monitoring and Control of Spacecraft Systems

Technical Note No. 478

January 1990

By: Michael P. Georgeff, Director
Australian Artificial Intelligence Institute
and
Francois Félix Ingrand, Computer Scientist
Artificial Intelligence Center

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

Some of the material appearing in this paper was presented at the 1989 Space Operations Automation and Robotics Workshop, Houston, Texas

This research is supported by the National Aeronautics and Space Administration, Ames Research Center, under Contract No. NA2-12521.

333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

Real-Time Reasoning: The Monitoring and Control of Spacecraft Systems

Michael P. Georgeff[1]
Australian AI Institute
1 Grattan Street
Carlton, Victoria 3053
Australia

François Félix Ingrand[1]
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025

Abstract

This paper describes research [1] concerned with automating the monitoring and control of spacecraft systems. In particular, the paper examines the application of SRI's Procedural Reasoning System (PRS) to the handling of malfunctions in the Reaction Control System (RCS) of NASA's space shuttle. Unlike traditional monitoring and control systems, PRS is able to reason about and perform complex tasks in a very flexible and robust manner, somewhat in the manner of a human assistant. Using various RCS malfunctions as examples (including sensor faults, leaking components, multiple alarms, and regulator and jet failures), it is shown how PRS manages to combine both goal-directed reasoning and the ability to react rapidly to unanticipated changes in its environment. In conclusion, some important issues in the design of PRS are reviewed and future enhancements are indicated.

1 Introduction

As space missions increase in complexity and frequency, the automation of mission operations grows more and more critical. Such operations include subsystem monitoring, preventive maintenance, malfunction handling, fault isolation and diagnosis, communications management, maintenance of life support systems, power management, monitoring of experiments, satellite servicing, payload deployment, orbital-vehicle operations, orbital construction and assembly, and control of extraterrestrial rovers. Automation of these tasks can be expected to improve mission productivity and safety, increase versatility, lessen dependence on ground systems, and reduce demands for crew involvement in system control.

It is very important that any system designed to perform these tasks be as flexible, robust, and interactive as possible. At the minimum, it should be capable of responding to and diagnosing abnormalities in a variety of configurations and operational modes. It should be able to integrate information from various parts of the space vehicle systems and recognize potential problems prior to alarm limits being exceeded.

[1] This research is supported by the National Aeronautics and Space Administration, Ames Research Center, under Contract No. NAS2-12521.
Some of the material appearing in this paper was presented at the 1989 Space Operations Automation and Robotics Workshop, Houston, Texas.

The system should suggest and execute strategies for containing damage and for making the system secure, without losing critical diagnostic information. It should be able to utilize standard malfunction handling procedures and take account of all the relevant factors that, in crisis situations, are easily overlooked. False alarms and invalid parameter readings should be detected, and alternative means for deducing parameter values should be utilized where possible.

In parallel with efforts to contain damage and temporarily reconfigure vehicle subsystems, the system should be able to begin diagnosis of the problem and incrementally adjust reconfiguration strategies as diagnostic information is obtained. The system should also be capable of communicating with other systems to seek information, advise of critical conditions, and avoid harmful interactions. Throughout this process, the system should be continually reevaluating the state of the space vehicle and should be capable of changing focus to attend to more serious problems should they occur.

Finally, the system should be able to explain the reasons for any proposed course of action in terms that are familiar to astronauts and mission controllers. It should be able to graphically display the system schematics, the procedures it is intending to execute, and the critical parameter values upon which its judgment is based.

Achieving this kind of behavior is well beyond the capabilities of conventional real-time systems. It requires, in contrast, mechanisms that can reason in a "rational" way about the state of the space vehicle and the actions that need be taken in any given situation. Moreover, the system should be both *goal directed* and *reactive*. That is, while seeking to attain specific goals, the system should also be able to react appropriately to new situations in real time. In particular, it should be able to completely alter focus and goal priorities as circumstances change. In addition, the system should be able to *reflect* on its own reasoning processes. It should be able to choose when to change goals, when to plan and when to act, and how to use effectively its deductive capabilities.

A number of system architectures for handling some of these aspects of real-time behavior have been recently proposed e.g., [Firby, 1987; Kaelbling, 1987; Hayes-Roth, 1985]. Some of these approaches are evaluated elsewhere [Georgeff and Ingrand, 1989; Georgeff and Lansky, 1987; Laffey *et al.*, 1988].

The system to be discussed in the paper is called a *Procedural Reasoning System* (PRS). It has been developed over a number of years at SRI International and has been reported, in part, in previous publications [Georgeff and Ingrand, 1989; Georgeff and Ingrand, 1988; Georgeff, 1988; Georgeff and Lansky, 1986a; Georgeff and Lansky, 1986b; Georgeff and Lansky, 1987].

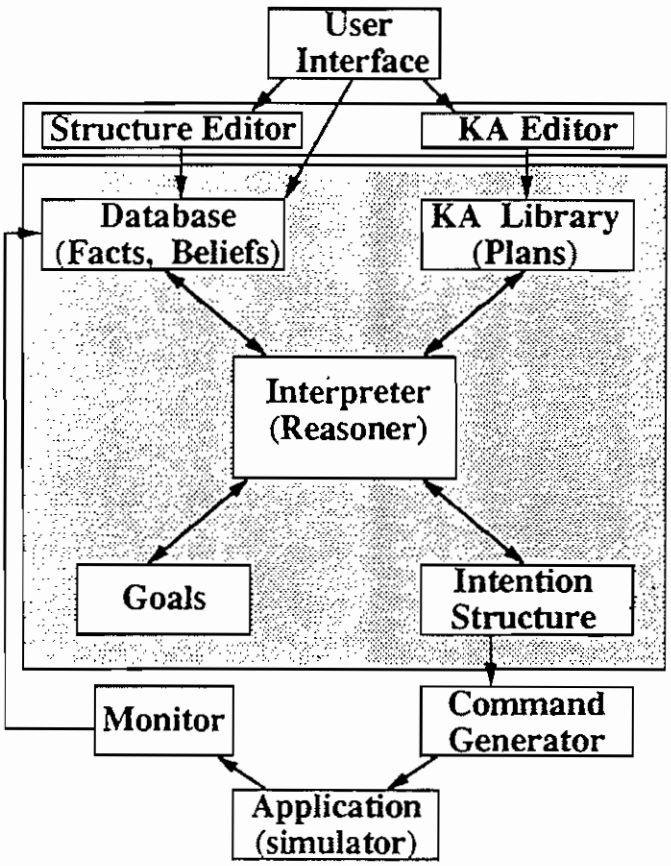

Figure 1: Structure of the Procedural Reasoning System

2 Procedural Reasoning System

PRS is designed to be used as an embedded, real-time reasoning system. As shown in Figure 1, PRS consists of (1) a *database* containing current *beliefs* or facts about the world; (2) a set of current *goals* to be realized; (3) a set of *plans*, called knowledge areas (KAs), describing how certain sequences of actions and tests may be performed to achieve given goals or to react to particular situations; and (4) an *intention structure* containing all KAs that have been chosen for execution. An *interpreter* (or *inference mechanism*) manipulates these components, selecting appropriate plans based on the system's beliefs and goals, placing those selected on the intention structure, and executing them.

The system interacts with its environment, including other systems, through its database (which acquires new beliefs in response to changes in the environment) and through the actions that it performs as it carries out its intentions.

Goals and Beliefs

The beliefs of PRS provide information on the state of the space vehicle systems and are represented in a first-order logic. For example, the fact that a particular valve, `v1` say, is closed could be represented by the statement `(position v1 c1)`.

The goals of PRS are descriptions of desired tasks or behaviors. In the logic used by PRS, the goal to achieve a certain condition C is written (! C); to test for the condition is written (? C); to wait until the condition is true is written (^ C); and to conclude that the condition is true is written ($\Rightarrow$ C). For example, the goal to close valve `v1` could be represented as `(! (position v1 c1)`, and to test for it being closed as `(? (position v1 c1))`.

Knowledge Areas

Knowledge about how to accomplish given goals or react to certain situations is represented in PRS by declarative procedure specifications called *Knowledge Areas* (KAs) (see. for example, Figure 10). Each KA consists of a *body*, which describes the steps of the procedure, and an *invocation condition*, which specifies under what situations the KA is useful and applicable. Together, the invocation condition and body of a KA express a declarative fact about the results and utility of performing certain sequences of actions under certain conditions [Georgeff and Lansky, 1986a].

The body of a KA can be viewed as a plan or plan schema. It is represented as a graph with one distinguished start node and possibly multiple end nodes. The arcs in the graph are labeled with the subgoals to be achieved in carrying out the plan. Successful execution of a KA consists of achieving each of the subgoals labeling a path from the start node to an end node. This formalism provides a natural and efficient representation of plans involving any of the usual control constructs, including conditional selection, iteration, and recursion.

The invocation condition contains a *triggering* part describing the *events* that must occur for the KA to be executed. Usually, these events consist of the acquisition of some new goals (in which case, the KA is invoked in a goal-directed fashion) or some change in system beliefs (resulting in data-directed or reactive invocation) and may involve both.

The set of KAs in a PRS application system not only consists of procedural knowledge about a specific domain, but also includes *metalevel* KAs; that is, information about the manipulation of the beliefs, goals, and intentions of PRS itself. For example, typical metalevel KAs encode various methods for choosing among multiple applicable KAs, modifying and manipulating intentions, and computing the amount of reasoning that can be undertaken, given the real-time constraints of the problem domain.

The Intention Structure

The intention structure contains all those tasks that the system has chosen for execution, either immediately or at some later time. These adopted tasks are called *intentions*. A single intention consists of some initial KA together with all the sub-KAs that are being used in attempting to successfully execute that KA. It is directly analogous to a *process* in a conventional programming system.

At any given moment, the intention structure may contain a number of such intentions, some of which may be suspended or deferred, some of which may be waiting for certain conditions to hold prior to activation, and some of which may be metalevel intentions for deciding among various alternative courses of action.

For example, in handling a malfunction in a propulsion system, PRS might have, at some instant, three tasks (intentions) in the intention structure: one suspended while waiting for, say, the fuel-tank pressure to decrease below some designated threshold; another suspended after having just posted some goal that is to be accomplished (such as interconnecting one shuttle subsystem with another); and the third, a metalevel procedure, being executed to decide which way to accomplish that goal.

Execution

Unless some new belief or goal activates some new KA, PRS will try to fulfill any intentions it has previously decided upon. This results in focussed, goal-directed reasoning in which KAs are expanded in a manner analogous to the execution of subroutines in procedural programming systems. But if some important new fact or goal does become known, PRS will re-

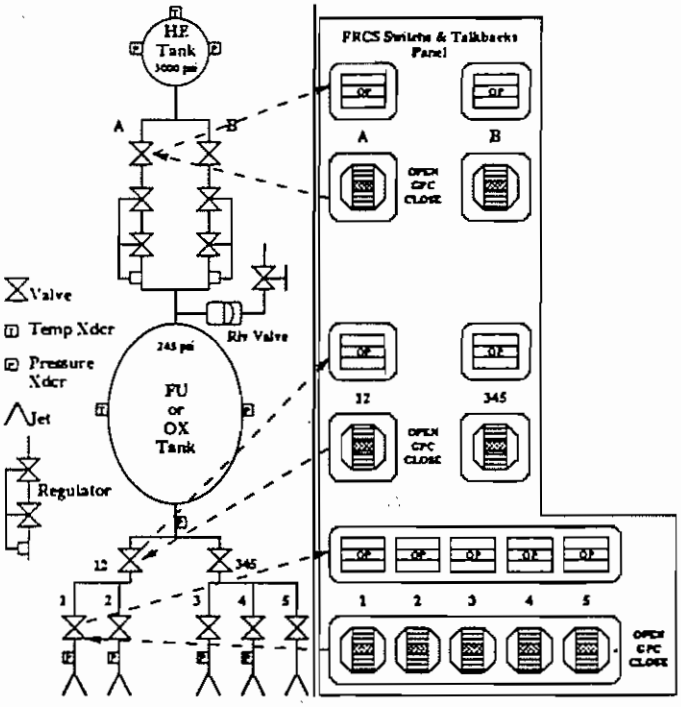

Figure 2: System Schematic for the RCS

assess its current intentions and perhaps choose to work on something else. Thus, not all options that are considered by PRS arise as a result of means-end reasoning. Changes in the environment may lead to changes in the system goals or beliefs, which in turn may result in the consideration of new plans that are not means to any already intended end. PRS is therefore able to change its focus completely and pursue new goals when the situation warrants it. In many space operations, this may happen quite frequently as emergencies of various degrees of severity occur in the process of handling other, less critical tasks.

Multiple Systems

In some applications, it is necessary to monitor and process many sources of information at the same time. Because of this, PRS was designed to allow several instantiations of the basic system to run in parallel. Each PRS instantiation has its own data base, goals, and KAs, and operates asynchronously relative to other PRS instantiations, communicating with them by sending messages.

The system described above has been implemented on Symbolics 3600 Series LISP, Sun Series 3, and Mac Ivory machines. A more complete description of PRS can be found elsewhere [Georgeff and Ingrand, 1989; Georgeff and Ingrand, 1988].

3 The RCS Application

The system chosen for experimentation with PRS is the Reaction Control System (RCS) of the space shuttle. The system structure is depicted in the schematic of Figure 2 (left part). One of the aims of our research is to automate the malfunction procedures for this subsystem. A sample malfunction procedure is presented in Figure 3.

The RCS provides propulsive forces from a collection of jet thrusters to control the attitude of the space shuttle. There are three RCS modules, two aft and one forward. Each module contains a collection of primary and vernier jets, a fuel tank, an oxidizer tank, and two helium tanks, along with associated feedlines, manifolds, and other supporting equipment. Propellant flow, both fuel and oxidizer, is normally maintained by pressurizing the propellant tanks with helium.

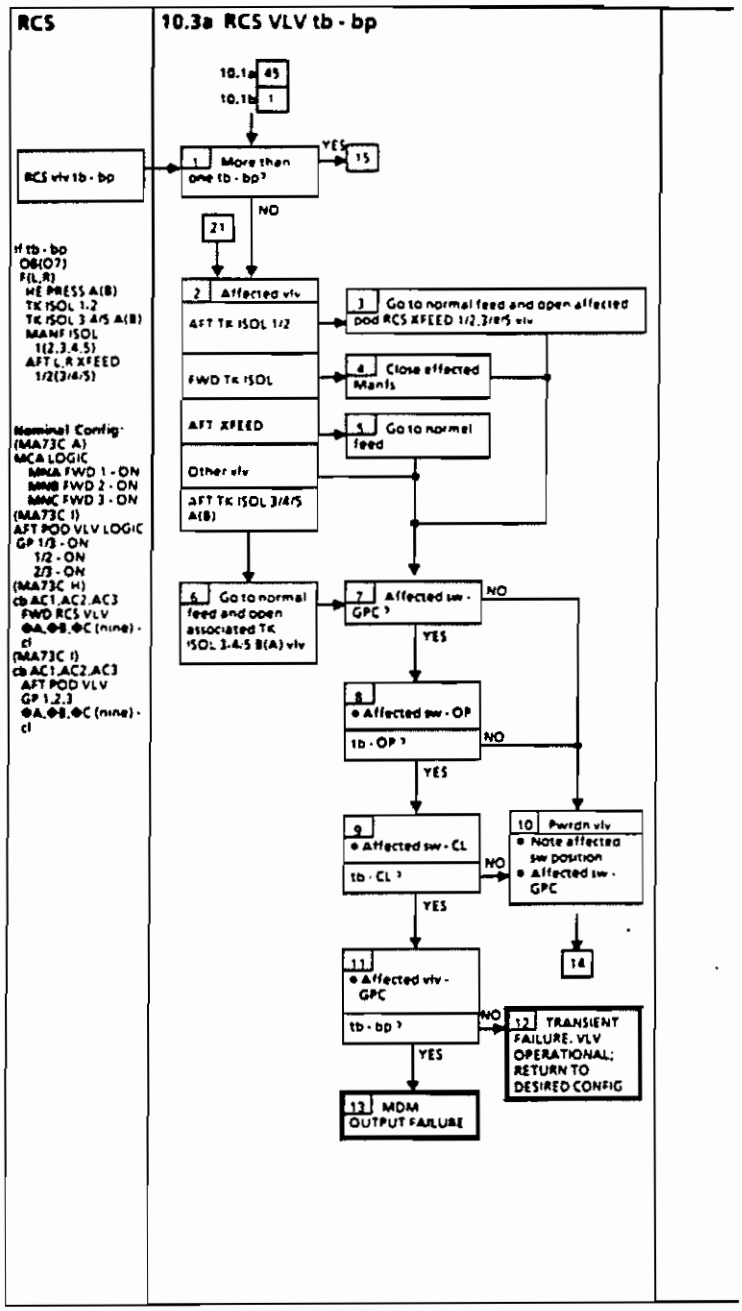

Figure 3: A RCS Malfunction Procedure

The helium supply is fed to its associated propellant tank through two redundant lines, designated A and B. The pressure in the helium tanks is normally about 3000 psi; this is reduced to about 245 psi by regulators that are situated between each helium tank and its corresponding propellant tank. A number of pressure and temperature transducers are attached at various parts of the system to allow monitoring.

Each RCS module receives all commands (both manual and automatic) via the space shuttle flight computer software. This software resides on five general purpose computers (GPCs). Up to four of these computers contain redundant sets of the Primary Avionics Software System (PASS) and the fifth contains the software for the Backup Flight System (BFS). All of the GPCs can provide information to the crew by means of CRT displays.

The various valves in an RCS module are controlled from a panel of switches and talkbacks (Figure 2, right part). Each switch moves associated valves in *both* the fuel subsystem

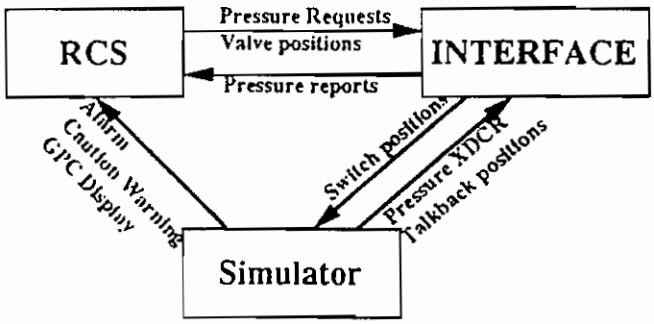

Figure 4: System Configuration

and the oxidizer subsystem.[2] Switches can be set to OPEN, CLOSE, or GPC, the last providing the GPCs with control of the valve position. The talkback provides feedback on the associated valve position. The talkback reading normally corresponds with the associated switch position, except when the switch is in GPC; in this case, the talkback shows whichever position the GPC puts the valve in. The talkbacks may not correspond if a valve has jammed or if the control or feedback circuit is faulty. If the valves in both the fuel and oxidizer subsystems do not move in unison, because of some fault, the talkback displays a barberpole.

As with most dynamic systems, transient faults are common. For example, in the process of changing switch position, there will be a short time (about 2 seconds) when the positions of the talkback and the switch will differ from one another. This is because it takes this amount of time for the actual valve to change its position. Furthermore, during this transition, the talkback will also pass through the barberpole position. Thus, a mismatched talkback and switch position or a barberpole reading does not always indicate a system fault.

4 System Configuration

Two instances of PRS were set up to handle the RCS application. One, called INTERFACE, handles most of the low level transducer readings, effector control and feedback, and checks for faulty transducers and effectors. The other, called somewhat misleadingly RCS, contains most of the high-level malfunction procedures, much as they appear in the malfunction handling manuals for the shuttle. To test the system, a simulator for the actual RCS was constructed.

The complete system configuration is shown in Figure 4. Each of these parts is described in the following sections.

4.1 The Simulator

During operation, the simulator sends transducer readings and feedback from various effectors (primarily valves) to INTERFACE and communicates alarm messages as they appear on the shuttle system displays to RCS. The simulator, in turn, responds appropriately to changes in valve switch positions as requested by INTERFACE. The simulator can be set to model a variety of fault conditions, including misread transducers, stuck valves, system leaks, and regulator failures.

A future implementation of the system will be connected to the more sophisticated shuttle simulator used at Johnson Space Center.

4.2 The RCS

The top-level PRS instantiation, RCS, contains most of the malfunction handling procedures as they appear in the operational manuals for the space shuttle. RCS takes an abstract view of the domain: it deals in pressures and valve positions, and does not know about transducers, switches, or talkbacks. For example, whenever RCS needs to know the pressure in a particular part of the system, it requests this information from INTERFACE, which is expected to deduce the pressure from its knowledge of transducer readings and transducer status. Similarly, RCS will simply request that INTERFACE moves a valve to a certain position, and is not concerned how this is achieved. In this way, RCS can represent the malfunction handling procedures in a clean and easily understandable way, without encumbering the procedures with various cross-checks and other details.

4.3 The INTERFACE

The PRS instantiation INTERFACE handles all information concerning transducer readings, valve switches, and valve talkbacks. It handles requests from RCS for information on the pressures in various parts of the system and for rates of change of these values. Determination of this information can require examination of a variety of transducers, as readings depend on the status of individual transducers, their location relative to the region whose pressure is to be measured, and the connectivity of the system via open valves.

INTERFACE also handles requests from RCS to change the position of the valves in the RCS. This involves asking the astronaut to change switch positions, and waiting for confirmation from the talkback.

While doing these tasks, INTERFACE is continually checking for failures in any of the transducers or valve assemblies. When it notices such failures, it will notify the astronaut or mission controller and appropriately modify its procedures for determining pressures or closing valves. It will also consider the consequences of any failures, such as are prescribed in various flight rules for the shuttle.

5 Sample Interactions

In this section, we examine different scenarios illustrating the capabilities of PRS.

5.1 Changing Valve Position

The following example illustrates the capacity of the system to reason about more than one task at a time. Consider the situation where INTERFACE gets a request from RCS to close some valve, say `frcs-ox-tk-isol-12-valve` (Forward RCS, OXidizer TanK, one-two ISOLation VALVE). RCS achieves this by sending INTERFACE the message (`request RCS (!(position frcs-ox-tk-isol-12-valve cl))`). Responding to this request, INTERFACE calls a KA that, in turn, asks the astronaut to place the switch corresponding to this valve in the closed position (see Figure 5). Once the astronaut has done this, INTERFACE will wait until the talkback shows the requested position and will then advise RCS that the valve has indeed been closed (Figure 5).

However, while this is taking place, INTERFACE will also notice that, just after the switch is moved to the closed position, there is a mismatch with the talkback indicator (which will still be showing open, because of the normal delay in the valve starting to move). Furthermore, a fraction of a second later, the talkback will move into the barberpole position, another indication that things could be wrong with the valve.

Each of these events will trigger a KA and thus initiate execution of a task (intention) that seeks to confirm that the talkback moves to its correct position within a reasonable time; Figure 6 shows the KA which monitors the barberpole position. At this point, the system is dealing with three different tasks, one responsible for answering the request, one checking the miscomparison between the switch and the talkback, and

[2] Because the two propellant subsystems are identical, only one system is represented in the left part of the figure.

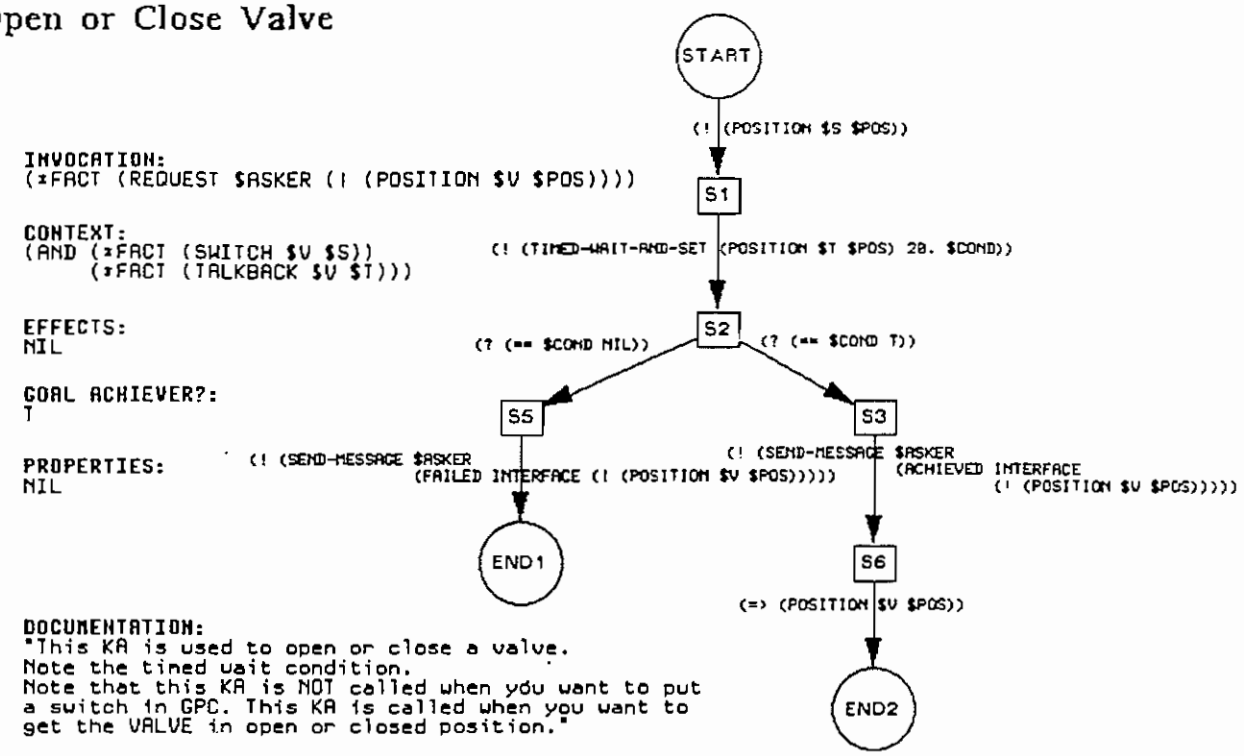

Figure 5: KA for Closing a Valve

one checking for the barberpole position. Each of these last two tasks immediately suspend themselves (using the "wait-until" (^) operator) while awaiting the specified condition to become true.

For example, the task concerned with monitoring a talkback barberpole reading will suspend itself until either the positions of both the switch and the talkback agree, or 10 seconds elapses. When either of these conditions become true, the task (intention) will awaken and proceed with the next step. If the talkback is still in the barberpole position, the astronaut or mission controller will be notified of the problem. Otherwise, the KA *fails*, and simply disappears from the intention structure.

Notice that the KAs that respond to the request from RCS to change the valve position, that monitor for possible switch dilemmas, and that check the barberpole reading are all established as different intentions at some stage during this process. Various metalevel KAs must therefore be called, not only to establish these intentions, but to decide which of the active ones to work on next.

A typical state of the intention structure is shown in Figure 7. It shows a number of intentions in the system INTERFACE, ordered for execution as indicated by the arrows. The intention labeled **Meta Selector** is a metalevel KA (Figure 8). The other intentions include two that are checking potential switch problems (**Switch Dilemma (Barberpole)** and **Switch Dilemma (Closed)**) and one that is responding to the request to close the valve (**Open or Close Valve**). The metalevel intention, in this case, is the one currently executing. Although not clear from the figure, it has just created and ordered the new intentions resulting from the talkback and the barberpole problems.

5.2 Handling Faulty Transducers

In this scenario, we show how two PRS agents cooperate and control the execution of their intentions so as to handle faulty transducers and the resulting false warning alarms.

We will assume that transducer **frcs-ox-tk-out-p-xdcr** fails and remains jammed at a reading of 170 psi. This causes a number of things to happen. First, it causes a low-pressure alarm to be activated. This will will be noticed by the PRS instantiation RCS, which will immediately respond to the alarm by initiating execution of the KA (**Pressurization Alarm (Propellant Tank)**). This KA will, in turn, request a pressure reading from INTERFACE to ensure that the alarm is valid.

While this is happening, INTERFACE by itself has noticed that the two transducers on the oxidizer tank disagree with one another (in this case, the other transducer is reading the nominal value of 245 psi). This invokes a KA that attempts to determine which of the two transducers is faulty. It does this by first waiting a few seconds to ensure that the mismatch is not simply a transient, and then testing to see if one of the readings is outside normal limits. If so, it assumes this is the faulty transducer; this is indeed the procedure used by astronauts and mission controllers. Other KAs, capable of more sophisticated acts such as checking the values of downstream or upstream transducers, are used if there is no corresponding transducer with which to do the cross-check.

Notice what could happen here if one is not careful. Having more than one thing to do, INTERFACE could decide to service the request for a pressure reading for the suspect tank. If it does so, it will simply average the values of the two transducer readings (yielding 207 psi) and advise RCS accordingly. Clearly, this is not what we want to happen: any suspect parameter readings should be attended to before servicing requests that depend on them.

In the examples we have considered, it has been sufficient to handle such problems with a relatively simple priority scheme. We first ascribe the property of being a so-called *"safety handler"* to all those KAs that should be executed at the earliest possible time. Then we design the metalevel KA that chooses between potentially applicable KAs to order all safety handlers for execution prior to other intentions. In the example given above, the KA that detects the faulty transducer is a safety handler, and thus is executed prior to servicing the request from RCS. When INTERFACE eventually gets around to servicing the request from RCS, it disregards the faulty transducer reading and thus advises RCS that the pressure is 245

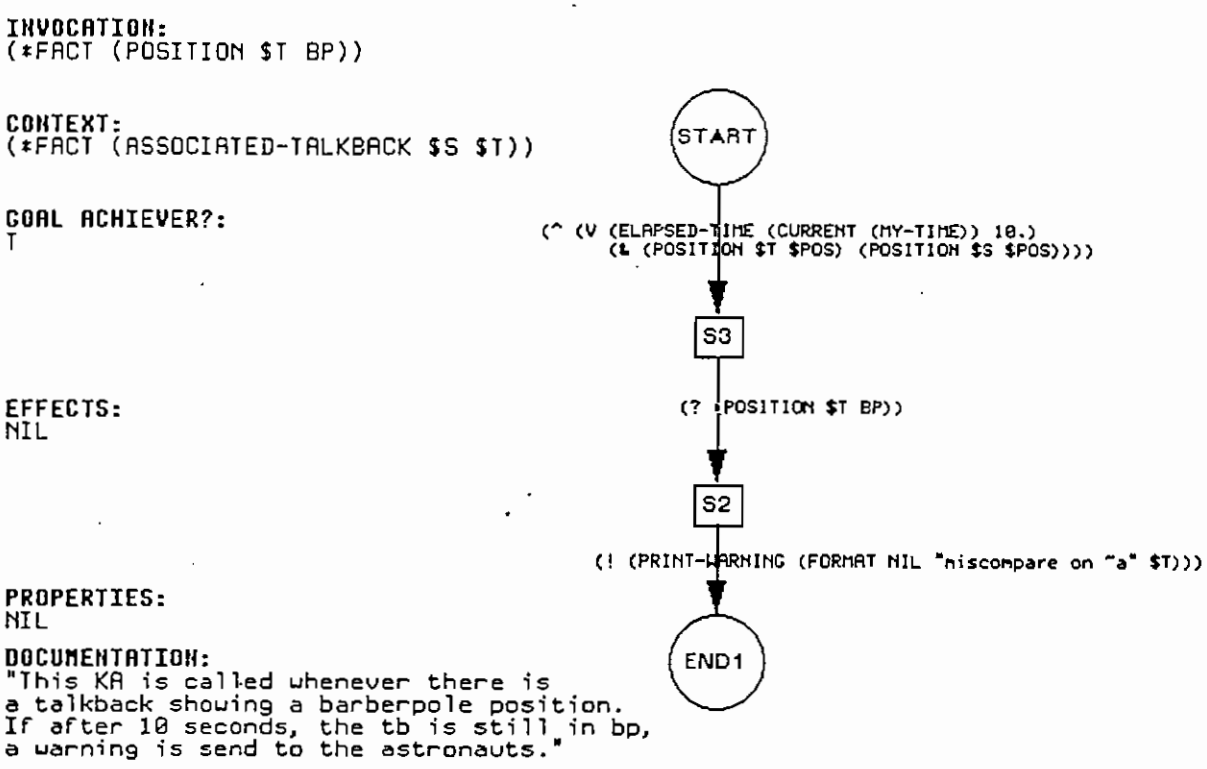

Figure 6: KA for Monitoring Talkback in Barberpole

psi. RCS then determines that the alarm was activated in error and that the pressure is within normal operating range.

Even with all this going on, other things are happening within the INTERFACE system. For example, the fact that the transducer is determined to be bad, together with the fact that it is the very transducer that informs the shuttle computers of overpressurization problems, causes the invocation of another KA. This KA reflects a flight rule that states that overpressurization protection is lost while the transducer is inoperative.

As before, metalevel KAs are invoked to determine which KAs to adopt as intentions and how to order them on the intention structure. The development of the intention structure during this process is shown in Figure 9.

5.3 Failed Regulator

Let's now consider the operation of the top-level PRS instantiation, RCS. The case we first examine occurs when the regulator on the feed line between the helium tank and its associated propellant tank fails. In this example, we will assume that the `frcs-fu-he-tk-A-reg` has failed. We will focus primarily on RCS (INTERFACE is, of course, working away during this process as discussed above).

The first thing that happens when the regulator fails is that pressures throughout the fuel subsystem begin to rise. When they exceed the upper limit of 300 psi, certain caution-warning (cw) alarms are activated. These events trigger the execution of a KA that attempts to confirm that the system is indeed overpressurized.

Note that this process is more complicated than it first appears. The high transducer readings that gave rise to the caution-warning alarm will also trigger KAs in the PRS system INTERFACE. These KAs will proceed to verify that the corresponding transducers are not faulty (as described in subsection 5.2); that is, that the reading of the transducers is indeed accurate. While doing this, or after doing this, INTERFACE will get a request from RCS to advise the latest pressure readings. If INTERFACE is in the process of checking the transducers, it will defer answering this request until it has completed its evaluation of transducer status. But eventually it will return to answering the request and, in the case we are considering, advise that the pressure is indeed above 300 psi.

On concluding that the system is overpressurized, another KA (`Overpressurized Propellant Tank`) is activated and this, eventually, concludes that the A regulator has failed (see Figure 10). Note that this KA establishes subgoals to close both the A valve and the B valve, as there are cases when both are open. For the A valve, this involves a request to INTERFACE as discussed above. However, for the B valve, the system notices that the B valve is already closed. Thus, its goal is directly achieved without the necessity to perform any action or request.

The final goal of this KA activates another KA that opens the valve of the alternate regulator (B). Having opened the valve, it is desirable to then place it under the control of the on-board computers. However, this cannot be done until the pressure in the system drops below 300 psi, as otherwise the GPC will automatically shut the valve again. Thus, the malfunction handling procedures specify that the astronaut should wait until this condition is achieved before proceeding to place the valve switch in the GPC position. RCS achieves this by asking INTERFACE to monitor the pressure and advise it when it drops below 300 psi. While waiting for an answer, the task is suspended, and RCS gets on with whatever else it considers important.

When the pressure eventually drops below that threshold,

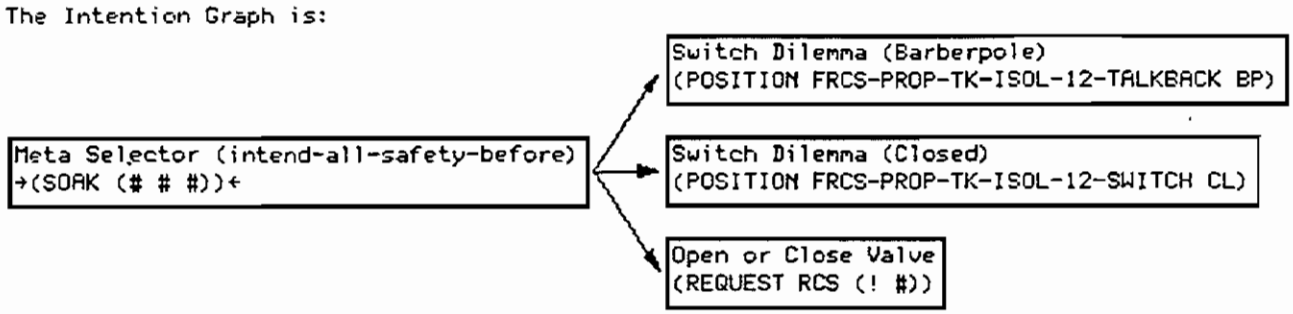

Figure 7: Intention Structure during Switch Operation

Meta Selector (intend-all-safety-before)

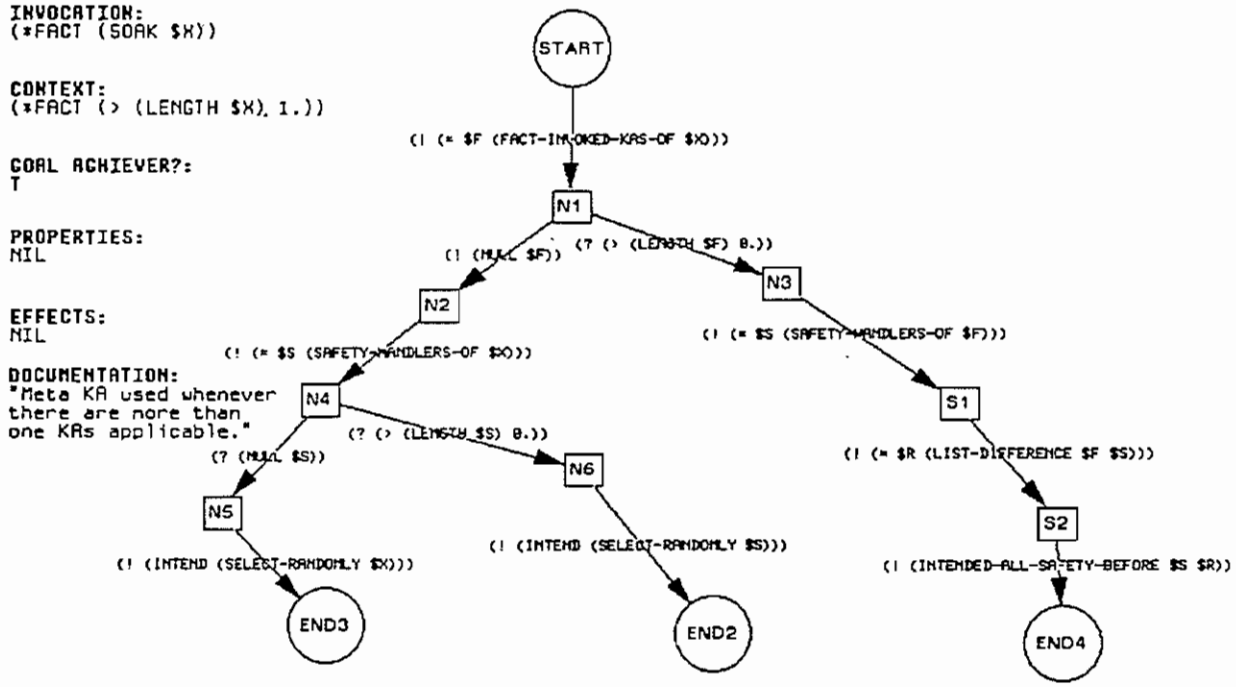

Figure 8: The Metalevel KA Meta Selector

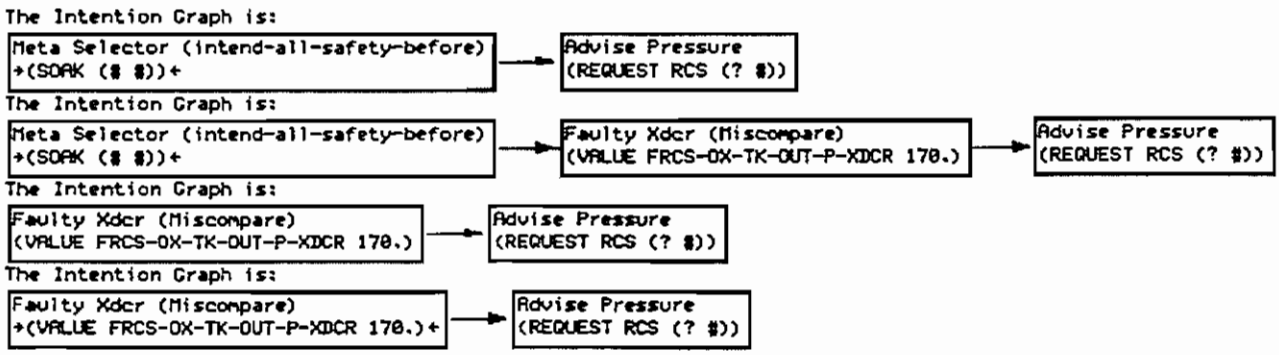

Figure 9: Intention Structure Development

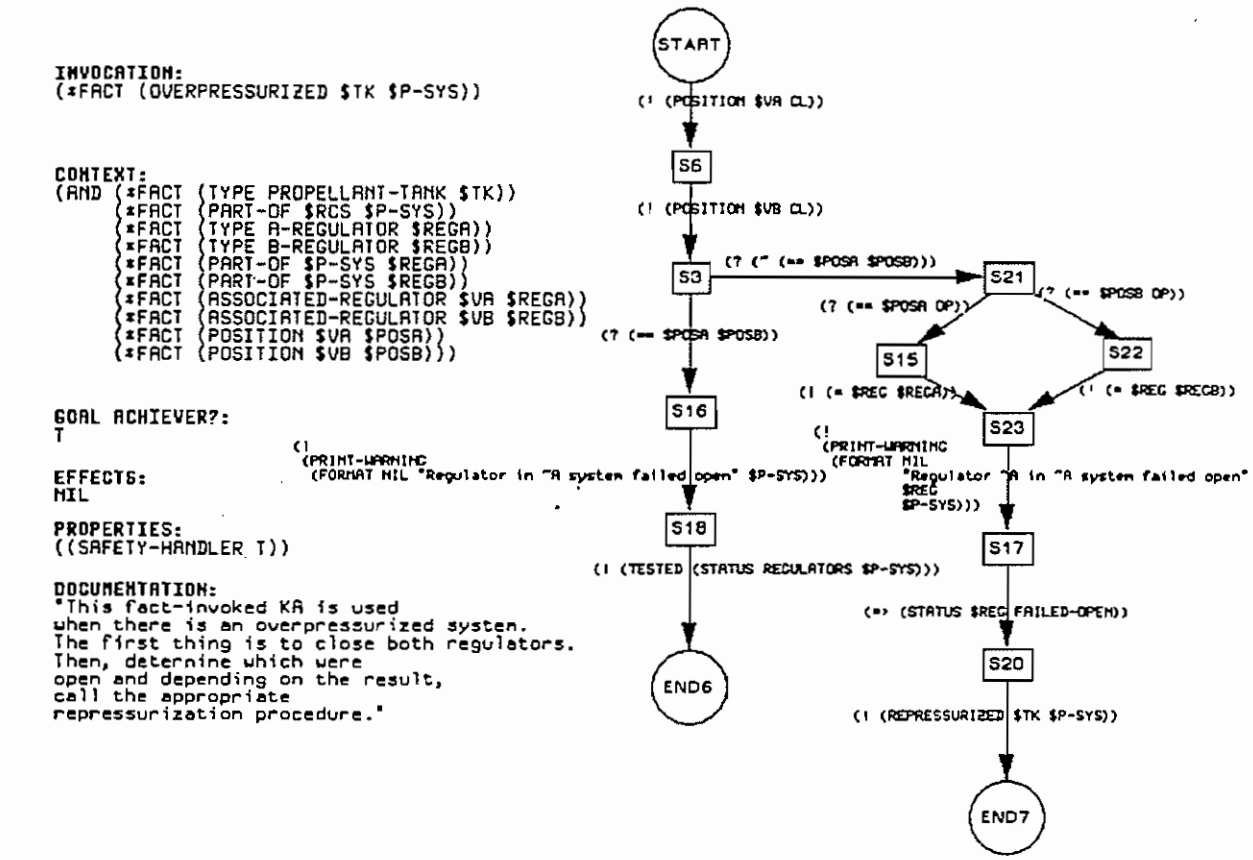

Figure 10: KA for Overpressurized Propellant Tank

the task (intention) is awakened, and execution continued. Thus, the valve switch is finally placed in the GPC position and the overpressurization problem resolved.

5.4 Isolating a System Leak

Let's assume that there is a leak in the RCS. Usually, the leak will cause a pressure drop in the system that will trigger a caution-warning alarm. The KA that responds to this alarm will first try to differentiate between a failed regulator and a leak in the system. If it determines that the system has a leak, it will then establish the goal to isolate that leak. This, in turn, triggers another KA that first attempts to secure the system This involves requesting that the astronauts close all valves in the leaking system.

Again, the PRS system INTERFACE will, throughout each process of closing a valve, check that the valve has indeed closed and that the corresponding talkbacks are registering closed.

As soon as the system has been secured, PRS identifies the leaking section by checking for decreasing pressure in each section of the RCS in turn.

6 Conclusion

The experiments described above provided a severe and positive test of the system's ability to operate proficiently in real time, to weigh alternative courses of action, to coordinate its activities, and to modify its intentions in response to a continuously changing environment. In addition, PRS met every criterion outlined by Laffey et al. [1988] for evaluating real-time reasoning systems: high performance, guaranteed response, temporal reasoning capabilities, support for asynchronous inputs, interrupt handling, continuous operation, handling of noisy (possibly inaccurate) data, and shift of focus of attention.

We believe that the following features of PRS played an important role in achieving these results.

Procedural reasoning: The representation of procedural knowledge using KAs is a very powerful way to describe the actions and procedures that should be executed to accomplish specific goals or to respond to certain critical events. One essential feature of the representation is that the elements of these procedures are described in terms of their *behaviors* rather than in terms of arbitrarily named actions or subroutines. For example, to achieve the goal "close all affected manifolds," it is essential to be able to reason about the intended set of manifolds and how the goal is then to be achieved; a call to a specialized procedure for every variant of this goal is simply too complex and too prone to error. Furthermore, a descriptive (declarative) representation of goals provides robustness as different procedures (KAs) can be used to accomplish the goal depending on the mode of operation,

the availability of resources, or the time required to perform the task. Moreover, because the purpose of each step in the procedure is so represented, other processes can independently decide how to achieve their own goals without thwarting that plan; indeed, they may even decide to assist.

Reactive and goal-directed reasoning: The capability of being simultaneously data- and goal-driven is a critical feature of PRS. PRS provides goal-driven reasoning when explicit goals must be achieved, such as closing a valve, or repressurizing a system. At the same time, the reactive capabilities of PRS allow it to respond to critical events that occur, even when PRS is itself attending to some other task. This capability of reacting to new events makes the system highly adaptive to situation changes: any plan can be interrupted and reconsidered in the light of new incoming information.

Real-time reasoning: One of the most important measures in real-time applications is reaction time; if events are not handled in a timely fashion, the process can go out of control. PRS has been designed so that such a guarantee can be furnished. Although PRS can execute complex conditional plans, the inference mechanism used in PRS guarantees that any new event is noticed in a bounded time [Georgeff and Ingrand, 1989; Georgeff and Ingrand, 1988]. While the system is executing any procedure, it monitors new incoming events and goals. Given that the real time behavior of the metalevel KAs used in a PRS application can be analyzed, the user can prove that his application can operate in real time: any new event is taken care of in a bounded time.

Reasoning about multiple tasks: The intention structure used in PRS enables the system to attend to more than one task at a time. These multiple intentions are usually tightly coupled and the order in which they are executed can be very important. Some may require immediate execution on the basis of urgency; others may have to be scheduled later than others because they depend on the results produced by the earlier tasks. Potential interactions among concurrently executing intentions can also be critical in deciding the most appropriate ordering of tasks. PRS provides the mechanisms to examine and manipulate the intention structure directly; the user can thus specify any kind of priority or scheduling scheme desired.

Metalevel reasoning: The provision of metalevel KAs allows the system to control its problem solving strategies in arbitrarily sophisticated ways. These metalevel KAs follow the same syntax and semantics as application KAs, except that they deal with the control of the execution of PRS itself. Thus one can write metalevel KAs that can reason efficiently and effectively about the problem solving process being used. For example, one can have a KA to control in which order the applicable KAs are going to be executed. In the example presented in the subsection 5.2, the metalevel KA makes sure that the system carries on the testing task before the pressure update task, thus allowing the false alarm to be correctly recognized. Similarly, one can use metalevel KAs to choose among different ways to perform a given task, or how best to meet the real-time constraints of the domain given information on the expected time required for task execution.

Distributed reasoning: PRS is designed for distributed operations. Thus, different instances of PRS can he used in any application that requires the cooperation of more than one agent. The different PRS agents run asynchronously; their activity is therefore unconstrained a priori by that of their colleagues. A message passing mechanism is provided to make possible communication between the different PRS agents as well as with external modules such as simulators or monitors.

A number of critical research problems remain to be solved before the system will be reliable enough for use in actual space operations. The system is currently being extended to cover all malfunction handling procedures and flight rules concerning the RCS and is to be tested against the main shuttle simulators at Johnson Space Center in future work.

Acknowledgments

Pierre Bessiere, Amy Lansky, Anand Rao, Lorna Shinkle, Joshua Singer, Mabry Tyson, and Dave Wilkins helped in the development of PRS and extended the implementation as needed. Matthew Barry and Janet Lee helped in the development of the RCS application and Oscar Firschein provided constructive criticism of the paper.

References

[Firby, 1987] R. J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, Washington, 1987.

[Georgeff and Ingrand, 1988] M. P. Georgeff and F. F. Ingrand. Research on procedural reasoning systems. Final Report, Phase 1, for NASA Ames Research Center, Moffet Field, California, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1988.

[Georgeff and Ingrand, 1989] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. To appear.

[Georgeff and Lansky, 1986a] M. P. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation*, 74:1383–1398, 1986.

[Georgeff and Lansky, 1986b] M. P. Georgeff and A. L. Lansky. A system for reasoning in dynamic domains: Fault diagnosis on the space shuttle. Technical Note 375, Artificial Intelligence Center, SRI International, Menlo Park, California, 1986.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning: An experiment with a mobile robot. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, 1987.

[Georgeff, 1988] M. P. Georgeff. An embedded real-time reasoning system. In *Proceedings of the 12th IMACS World Congress*, Paris, France, 1988.

[Hayes-Roth, 1985] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[Kaelbling, 1987] L. P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 395–410. Morgan Kaufmann, Los Altos, California, 1987.

[Laffey *et al.*, 1988] T. J. Laffey, P. A. Cox, J. L. Schmidt, S. M. Kao, and J. Y. Read. Real-time knowledge-based systems. *AI Magazine*, 9(1):27–45, 1988.