Decision-Making in an Embedded Reasoning System

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

Francois Felix Ingrand
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025

Abstract

The development of reasoning systems that can reason and plan in a continuously changing environment is emerging as an important area of research in Artificial Intelligence. This paper describes some of the features of a Procedural Reasoning System (PRS) that enables it to operate effectively in such environments. The basic system design is first described and it is shown how this architecture supports both goal-directed reasoning and the ability to react rapidly to unanticipated changes in the environment. The decision-making capabilities of the system are then discussed and it is indicated how the system integrates these components in a manner that takes account of the bounds on both resources and knowledge that typify most real-time operations. The system has been applied to handling malfunctions on the space shuttle, threat assessment, and the control of an autonomous robot.

1 Introduction

While there has been an increasing amount of interest in the development of reasoning systems suited to real-time operations, very few currently exhibit the kind of behavioral properties that we would expect of such systems [Laffey *et a*/., 1988]. Much of this derives from their inability to reason effectively about actions and plans. On the other hand, most existing planning systems cannot operate under the stringent constraints on both information availability and decision time that are typical of real-time applications. As a rule, these systems formulate an entire course of action before commencing execution of the plan [Wilkins, 1988]. However, in real-world domains, information about how best to achieve some given goal (and thus how best to complete a plan) can often only be acquired after executing some initial part of that plan.

Real-time constraints pose yet further problems for traditionally structured planning and reasoning systems. First, the planning and deductive techniques typically used by these systems are very time consuming. While this may be acceptable in some situations, it is not suited to domains where replanning is frequently necessary and where system viability depends on readiness to act. Second, traditional planning systems usually provide no mechanisms for responding to new situations or goals during plan execution, let alone during plan formation. Yet the very survival of an autonomous system may depend on its ability to react to new situations and to modify its goals and intentions accordingly [Georgeff and Lansky, 1987].

A number of systems developed for the control of robots and other real-time processes do have a high degree of reactivity [Brooks, 1985, Kaelbling, 1987]. Such architectures could lead to more viable and robust systems than traditionally structured planning systems. Yet most of this work has not addressed the issues of general problem-solving and commonsense reasoning; the work is instead almost exclusively devoted to problems of navigation and the execution of low-level actions.

There has been some recent work in the design of systems that attempt to integrate goal-directed reasoning and reactive behavior. For example, the RAP system [Firby, 1987] invokes tasks on the basis of system beliefs about the current world situation and then expands these tasks hierarchically. It has many features in common with the system described in this paper. However, it does not appear to provide sufficiently powerful mechanisms for balancing the decision-making requirements against the constraints on time and information that are typical of complex domains.

As Bratman, Israel, and Pollack [1988] remark: "... Rational agents must perform both means-end reasoning and weigh alternative courses of action; so an adequate architecture of intelligent agents must

This research is partly supported by the National Aeronautics and Space Administration, Ames Research Center, under Contract No. NAS2-12521.

Also a member of the Artificial Intelligence Center, SRI International, and the Center for the Study of Language and Information, Stanford University.

therefore include capabilities for both. The design of such an architecture must also specify how these capacities interact. But there is yet another problem. All this must be done in a way that recognizes the fact that agents are [bounded both in resources and knowledge].'" In this paper, we describe a system architecture that aims to achieve this balance between acting and decision-making.

2 Procedural Reasoning Systems

The problem domain we discuss in this paper is the task of malfunction handling for the Reaction Control System (RCS) of NASA's space shuttle. The shuttle contains three such systems, one forward and two aft. Each is a relatively complex propulsion system that is used to control the attitude of the shuttle. The astronaut handles system malfunctions by carrying out prespecified malfunction handling procedures. These procedures can be viewed as unelaborated plans of action, and are designed to be executed in a complex and changing environment.

To tackle this problem, we developed an embedded reasoning system called PRS (Procedural Reasoning System). Early versions of the system have been described before [Georgeff and Lansky, 198Ga, Georgeff and Lansky, 1980b, Georgeff and Lansky, 1987] and the full application is described in a recent report [Georgeff and Ingrand, 1988].

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) 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 those plans that have been chosen for [eventual] execution (Figure 1). 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.

2.1 The System Database

The contents of the PRS database may be viewed as representing the current beliefs of the system. Typically, these will include facts about static properties of the application domain, such as the structure of some subsystems or the physical laws that must be obeyed by certain mechanical components. Other beliefs are acquired by PRS itself as it executes its KAs. These will typically be current observations about the world or conclusions derived by the system from these observations, and these may change over time. The knowledge contained in the database is represented in first-order predicate calculus.

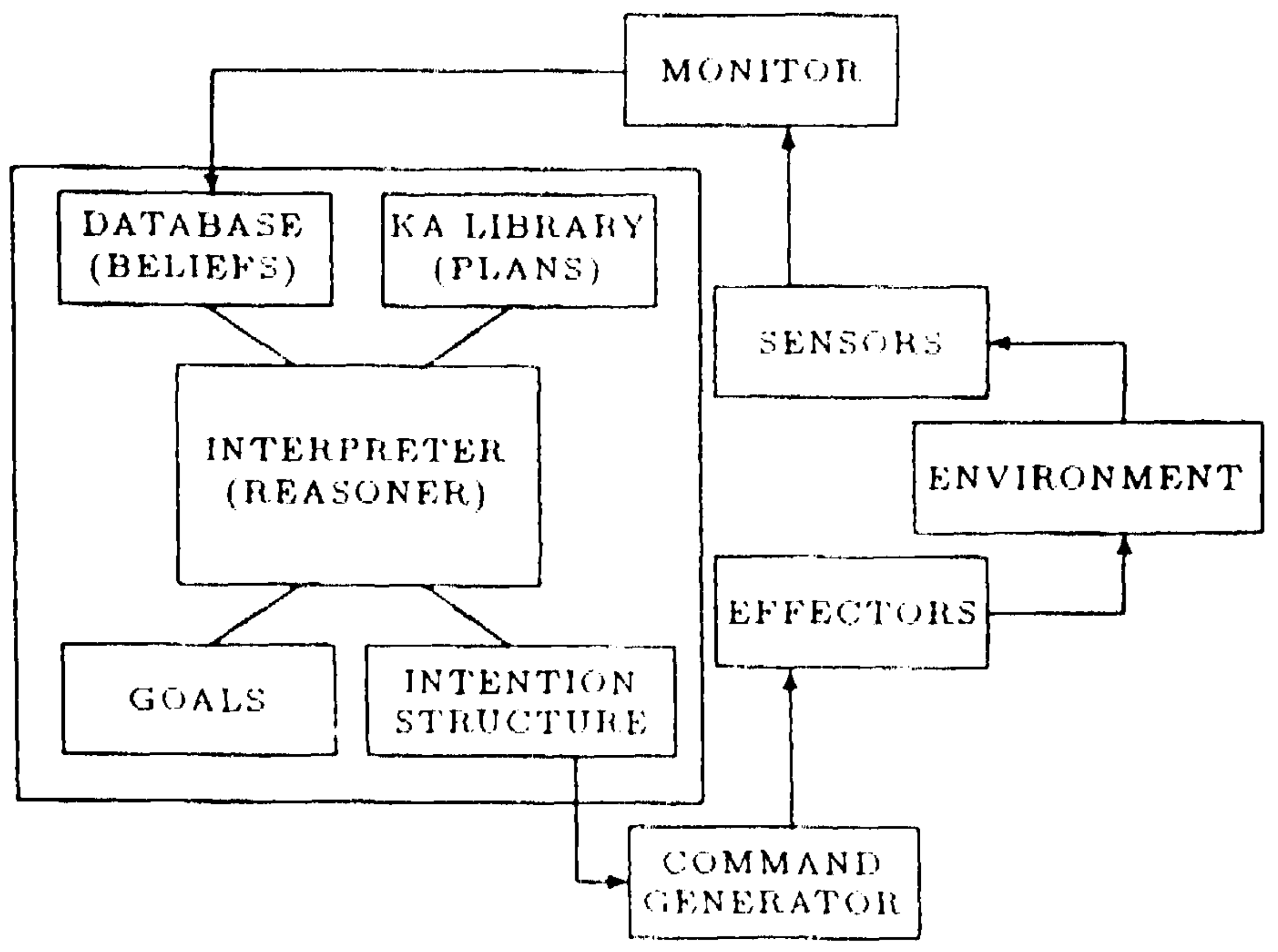

Figure 1: System Structure

State descriptions that describe internal system states are called *melalevel* expressions. These typically describe the beliefs, goals, and intentions of the system, as well as other important processing information.

2.2 Goals

Goals are expressed as conditions over some interval of time (i.e., over some sequence of world states) and are described by applying various temporal operators to state descriptions. This allows us to represent a wide variety of goals, including goals of achievement, goals of maintenance, and goals to test for given conditions. A given action (or sequence of actions) is said to *succeed* in achieving a given goal if its execution results in a behavior that satisfies the goal description.

As with state descriptions, goal descriptions are not restricted to specifying desired behaviors of the external environment but can also characterize the internal behavior of the system. Such descriptions are called metalevel goal specifications.

2.3 Knowledge Areas

Knowledge about how to accomplish given goals or rea"t to certain situations is represented in PRS by declarative procedure specifications called *Knowledge Areas* (KAs). 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 (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 allows richer control constructs (including conditional selection, iteration, and recursion) than most plan representations.

The invocation condition contains a *triggering* part describing the *events* that must occur for the KA to be executed. Usually, these consist of the acquisition of some new goal (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.

Some KAs have no bodies. These are so-called primitive KAs and have associated with them some primitive action that is directly performable by the system.

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, desires, 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.

2.4 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 the RCS, the system 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.

The set of intentions comprising the intention structure form a partial ordering with possibly multiple least elements (called the *roots* of the structure). An intention earlier in the ordering must be either realized or dropped (and thus disappear from the intention structure) before intentions appearing later in the ordering can be executed. This precedence relationship between intentions enables the system to establish priorities and other relationships between intentions.

2.5 The System Interpreter

The PRS interpreter runs the entire system. From a conceptual standpoint, it operates in a relatively simple way. At any particular time, certain goals are active in the system and certain beliefs are held in the system database. Given these extant goals and beliefs, a subset of KAs in the system will be applicable (i.e., will be invoked). One or more of these applicable KAs will then be chosen for execution and thus will be placed on the intention structure.

In determining KA applicability, the interpreter will not automatically perform any deduction. Both beliefs and goals are matched directly with invocation conditions by using unification only. This allows appropriate KAs to be selected very quickly and guarantees a certain degree of reactivity. If we allowed arbitrary deductions to be made, we could no longer furnish such a guarantee. However, PRS is always able to perform any deductions it chooses by invoking appropriate metalevel KAs. These metalevel KAs are themselves interruptible, so that the reactivity of the system is retained.

Once selected, the chosen KAs are inserted into the intention structure. If a selected KA arose due to the acquisition of a new intrinsic goal[1] or a new belief, it will be inserted into the intention structure as a new intention. For example, this will be the case for a KA that is invoked by the activation of some caution-warning alarm during shuttle operations. Otherwise, the KA instance must have arisen as a result of some operational goal of some existing intention, and will be pushed onto the stack of KAs comprising that, intention.

Finally, an intention at one of the roots of the intention structure is selected for further execution. The next step of that intention will comprise either a primitive action or one or more unelaborated subgoals. If the former, the action is directly initiated; if the latter, these subgoals are p>osted as new operational goals of the system.

Execution of primitive actions can effect not only the external world but also the internal state of the system. For example, a primitive action may operate directly on the beliefs, goals, or intentions of the system. Alternatively, the action may indirectly affect the system's state as a result of the knowledge gained by its interaction with the external world.

At this point the interpreter cycle begins again: the newly established goals and beliefs trigger new KAs, one or more of these are selected and placed on the intention structure, and finally an intention is selected from that structure and partially executed.

'An *intrinsic goal* is one that is not a means to an already intended end, whereas an *operational goal* is a subgoal of some already existing intention.

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 reassess 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's 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 happens quite frequently as emergencies of various degrees of severity occur in the process of handling other, less critical tasks.

We have, of course, left out a crucial component of the system's reasoning: how does it make the various selections and decisions mentioned above[7] We now turn to this and other important features of PRS.

3 Integrating Decision-Making and Means-Ends Reasoning

PRS is designed so that, in the absence of any decision knowledge being provided to the system, it nevertheless continues to function in an acceptable way as an embedded reasoning system. As more and more decision knowledge is added to the system, it can more effectively choose its actions, but always with a guaranteed upper bound on reaction time.

This is accomplished by embedding in the system interpreter certain fixed decision-making processes that are stringently bounded in execution time, yet which can be *overridden* whenever the system can bring more powerful decision-making knowledge to bear. Bratman et al. [1988] have outlined a similar design philosophy, in which they distinguish between a computationally efficient *compatibility filter* and a possibly computationally complex *filter override mechanism.* In PRS, the fixed decision-making processes are hardwired into the basic system interpreter, whereas the knowledge to override or elaborate these decisions is contained in appropriate metalevel KAs.

The way these metalevel KAs are brought to bear on any particular problem is via their invocation criteria. These criteria may depend both on conditions obtaining in the external world and, more typically, on conditions relating to the internal state of the system. Such conditions might include, for example, the applicability of multiple KAs in the current situation, the failure to achieve a certain goal, or the awakening of some previously suspended intention.

In some of these cases (such as the latter two of those mentioned above), these conditions will be known at the beginning of each interpretation cycle. Rut others (such as the number of KAs applicable at a given moment) can only be determined partway through this cycle. Thus, the actual manner of invocation of metalevel KAs is somewhat complex [Georgeff and Ingrand, 1988], but eventually results in the selection of a single KA at some level in the metahierarchy. This KA is then executed just as any other KA. In the process of execution, such metalevel KAs typically make choices about the adoption of lower-level KAs (which may themselves be inetaKAs), the posting of alternative subgoals, or the best way to respond to goal failures. Any of these decisions may require an arbitrary amount of processing or deduction, yet system reactivity is always guaranteed (see Section 5.1).

Thus, when decisions have to be made in the absence of any information about what is best to do, the system interpreter defaults to some fixed decision procedure. However, if there is any knowledge that can be brought to bear on the decision (via appropriate metalevel KAs), this will override the default action and determine the selection. In this way, the addition of appropriate metalevel KAs enables the system to make more informed choices at the cost of longer decision times.

This approach to the design of PRS is important in another way. We can now construct a relatively simple basic interpreter, knowing that the system is *not* going to always be constrained to behave in the way dictated by this component — its decisions can, in situations determined by the invocation conditions of the metalevel KAs, be overridden. Of course, it is important that the basic interpreter be able to handle appropriately the most commonly occurring situations so that the metalevel KAs are utilized only in the more exceptional circumstances.

Before concluding this section, it is important to note that, the decision-making behavior of PRS is strongly influenced by the choice of the invocation conditions of metalevel KAs. For example, if these conditions are such that the decision-making metaKAs are frequently invoked, PRS will spend more time making decisions than otherwise. It will thus tend to act in a *cautious* manner [Bratman *et ai,* 1988], carefully choosing what actions to perform next. If, on the other hand, these metalevel KAs are rarely invoked, PRS will act in a *bold* manner, rapidly choosing its actions in response to the changing world in which it, is embedded.

4 The Nature and Role of Intentions

Intentions play a significant role in PRS. In this section we examine their use both in managing events in the real world and in limiting the amount of decision making that, has to be undertaken, given the real-time constraints of the domain.

4.1 Intention States

In PRS, an intention can be in one of three possible states: active, suspended, or conditionally suspended. An *active intention* is one that is available for execution as soon as it becomes a root of the intention structure. A *suspended intention* is one that PRS has adopted hut for which no decision has been made a* to when it should be carried out; that is, it. must be explicitly activated before it can be executed. A *conditionally suspended intention* (or, simply, a *conditional intention)* is one that is temporarily suspended until some specified condition, called the *activation condition* of the intention, is satisfied.

In PRS, intentions can be suspended (conditionally or otherwise) by means of certain metalevel KAs. When a conditional intention is reactivated, it is necessary to decide whether or not to reorder the intention structure and what to do, if anything, with currently executing intentions. In the absence of any decision knowledge, it is desirable to ensure immediate response to the condition that activated the intention. Thus, the basic interpreter is designed to begin execution of the reactivated intention immediately, suspending all currently executing intentions. However, as with all other default decisions taken by the interpreter, metaKAs can respond to reactivation and choose to override this decision.

Similarly, various metalevel KAs exist for reordering intentions on the intention structure. In the absence of any decision knowledge the basic interpreter inserts new intentions at the root of the existing intention structure, and removes them when completed (see Section 4.3). However, metalevel KAs may choose to alter this ordering on the basis of task priorities, time availability, and so on.

4.2 The Role of Intentions

The role of commitment to previously adopted plans or intentions has been claimed to be a critical component of rational agency [Bratman, 1987, Cohen and Levesque, 1987]. It is also a critical component of the PRS architecture, allowing the system to meet the real-time constraints of a continuously changing world.

Unless some particular metalevel KA intervenes, PRS will perform its means-ends reasoning and other planning *in the context of its existing intentions.* For example, consider that PRS has adopted the intention of achieving a goal *g* by accomplishing the subgoals </i, <2> and #3, in that order. In the process of determining how to accomplish these subgoals, the system will not reconsider other means of achieving *g.* That is, it is *committed* to achieving *g by* doing #1, $</_2$, &nd #3, even if circumstances have so changed that there is now a better way to achieve *g* than the one chosen. The gain here is in reducing decision time — in highly dynamic domains it is not possible to continually reassess one's plans of action. What makes the approach workable is that the basis upon which one chooses a particular plan of action is more often correct than not.

It is not only in means-ends reasoning that PRS's commitment to its existing intentions is important. For example, in tackling some new task, it is often desirable that the means or time chosen for accomplishing that task take account of one's existing intentions towards the fulfillment of other tasks. In the RCS application, this happens, for example, when PRS receives a request for a pressure reading when it is in the process of evaluating the status of a suspected faulty transducer. In this case, PRS will either defer or suspend attending to that request (possibly advising the requester) until it has completed its evaluation of the transducer.

4.3 The Establishment and Dropping of Goals

One does not want to be committed to one's intentions forever. Thus, it is important to understand the way in which intentions are managed in PRS - in particular, how the goals comprising an intention are established and dropped.

As discussed in previous sections, intentions are established by a change in the system's goals or beliefs, and their execution will generate certain operational goals. Should the system attempt to achieve an operational goal and *fail,* that, goal will be reestablished, and another attempt made to achieve it. This will continue until the system comes to believe either that the goal is accomplished (through its own efforts or those of some other agent) or that the goal cannot be readily accomplished. Once this state is reached, the goal will be dropped.

This raises two related issues: what other attempts are made to achieve the goal and how does the system come to believe a goal cannot be readily accomplished? We shall answer the latter question first. One way for the system to believe that a goal cannot be accomplished (or, at least, that the goal is not worth pursuing further) is to deduce it using appropriate metalevel KAs. However, the system rarely has sufficient knowledge of the world to *prove* that a goal cannot, be achieved; this is thus unacceptable as the only (or even as the primary) manner of dropping goals (cf. the work of Cohen and Levesque [1987], who base their axiomatization of rational behavior solely on this kind of mechanism).

Another good reason for dropping a goal is simply to fail in all attempts at achieving the goal. This requires no more knowledge than the system acquires in its striving for the goal, and thus is appropriate to be used as the default method for the basic interpreter. But it brings us back to the former question: what attempts does the system make to achieve a given goal?

In PRS, the basic system interpreter tries, exactly once, every possible KA instance that can possibly achieve the goal. It does not ask that previously achieved goals be reachieved (in some other way), nor does it try the same KA instance more than once. In this sense, it is equivalent to a "fast-backtrack" parser. There are good reasons for these choices.

Unlike off-line planning or the parsing of sen tences, PRS operates in the real world rather than an hypothetical one. Thus, the actions it takes cannot be withdrawn and alternative approaches tried in their stead. Once some goal has been achieved, it, has *really* been achieved — there is no point in in voking KAs solely because they represent alternative ways to achieve an already achieved goal.

The remaining issue concerns the number of tries we make of a single KA instance to achieve a given goal. It is, of course, quite possible that, where the first try does not succeed, the next will, even if we carry out exactly the same actions as we did the first time. However, to determine if retrying could succeed would, in most practical cases, require knowledge of the state of the world that goes well beyond that, available to the system. In the absence of such knowledge, the system therefore tries each applicable KA instance exactly once. (The RAP system [Firby, 1987], on the other hand, continues to reinvoke already-tried task networks; clearly, this presents the serious problem of indefinite looping.)

Once all attempts at achieving a given goal have been exhausted, it is still possible for some metalevel KA to respond to this failure and invoke yet other means to achieve the goal. For example, a meta-KA could invoke certain deductive machinery, or could decide to retry some KA instances that, although having been tried once, appear (for some reason known to the meta-KA) to be worth trying again. In this way, the provision of additional decision knowledge can always override the basic processing method described above.

5 Planning in Real Time

5.1 Guaranteed Reactivity

Response time is one of the most important measures in real-time applications; if events are not handled in a timely fashion, the system can go out of control. Yet few existing real-time systems are guaranteed to respond within a bounded time interval [Laffey *ct ai,* 1988].

Response time is the time the system takes to recognize and respond to an external event. Thus, a bound on *reaction time* (that is, the ability of a system to recognize or notice changes in its environment) is a prerequisite for providing a bound on response time. PRS has been designed to operate under a well-defined measure of reactivity. Because the interpreter continuously attempts to match KAs with any newly acquired beliefs or goals, the system is able to notice newly applicable KAs after every primitive action it takes.

Let *p* be an upper bound on the execution times of the primitive actions that the system is capable of performing. Let's also assume that *n* is an upper bound on the number of events that can occur in unit time, and that the PRS interpreter takes at most time *t* to select the set of KAs applicable to each event occurrence.[2] By calculating-the number of events that can occur in a single cycle of the PRS interpreter, it is not difficult to show [Georgeff and Ingrand, 1988] that the maximum *reactivity delay* is *AR ~ J>/(1* — n<), where we assume that *t* < *l/n.*

This means that, provided the number of events that occur in unit time is less than *l/t,* PRS will notice *every* event that occurs and is guaranteed to do so within a time interval *AR.* (If the event rate exceeds 1//, the system may not be able to keep abreast of the changes in the environment.) In the shuttle application, which includes over 100 KAs and 1000 facts about the RCS, the values of *p* and *t* are less than 0.1 seconds, giving a reactivity delay of at most 0.2 second for an event rate of 5 events per second.

Because metalevel procedures are treated just like any other, they too are subject to being interrupted after every primitive metalevel action they take. Thus, reactivity is guaranteed even when the system is choosing between alternative courses of action or performing deductions of arbitrary complexity.

Having reacted to some event, it is necessary for the system to respond to this event by performing some appropriate action. As the system can be performing other tasks at the time the critical event is observed, a choice has to be made concerning the possible termination or suspension of those tasks while the critical event is handled. Furthermore, it may be necessary to decided among different alternatives for handling the event. With appropriate metalevel KAs, it is possible to guarantee a bound on this decision time [Georgeff and Ingrand, 1988]. However, the construction of such algorithms remains one of the more interesting areas of research in the design of real-time systems [Bratman *et a*/., 1988, Dean and Boddy, 1988].

5.2 Planning or Not?

There has always been some confusion in the literature about the notion of planning, especially with respect to the kind of practical reasoning that PRS performs. In the AI literature, planning is viewed as the generation of a sequence of actions to achieve some given goal. The classical approach to this problem is to simulate the effects of performing the actions so as to ensure that their execution does indeed achieve the required goal.

However, this is not the only way to construct an effective plan. For example, the choice among alternative courses of action could be based on the expected time to complete the actions, or the likelihood of success of the plans as gained through experience. In any case, simply making the choice as to which course of action to pursue, no matter how one does it, constitutes forming a *plan* to achieving ones goals.

[2]As selection of KAs does not involve any general deduction beyond unification and evaluation of a boolean expression, an upper bound does indeed exist.

This is exactly the way PRS operates. The method of choosing between alternative courses of action is embedded in the metalevel KAs of the system and thus, in essence, the particular *approach* to forming plans is not hard-wired into the system. To the extent that the choice is made arbitrarily, one may wish to avoid calling this process "planning." But where it is based on any information at all, no matter how meager, the determination of an appropriate course of action is indeed a form of planning.

In the RCS example, the system decides between different courses of action depending on how the KA was invoked and what sort of priority it has. This is clearly quite a weak form of planning, and more complex meta-KAs — taking time availability, costs, and benefits into account could be expected to improve system reliability. However, it is interesting to observe just how weak the planning component can be when we have a wealth of experience (a rich set of object-level KAs) to assist us.

6 Conclusions

The system described above was implemented on a Symbolics 3600 Series LISP machine and has been used to detect and recover from most of the possible malfunctions of the RCS, including sensor faults, leaking components, and regulator and jet failures. This was accomplished by using multiple communicating instantiations of PRS and a simulator for providing real-time input to the system. Complete details of this large-scale application are given elsewhere [Georgeff and Ingran d, 1988].

The experiment 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 cont inuously 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.

The features of PRS that, we believe, contributed most to its success at this task were (1) its partial planning strategy, (2) its reactivity, (3) its use of procedural knowledge, and (4) its metalevel (reflective) capabilities. In particular, the manner in which the system integrates its means-ends reasoning with the use of decision knowledge is considered an important component of rational activity.

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. The project has also benefited greatly from our discussions with Michael Bratman, Phil Cohen, David Israel, and Martha Pollack.

References

[Bratman *et ai,* 1988] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Comp. Linguistics,* 1988.

[Bratman, 1987] M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Mass, 1987.

[Brooks, 1985] R. A. Brooks. A robust layered control system for a mobile robot. TR 864, AI Laboratory, Massachusetts Institute of Technology, Cambridge, Mass, 1985.

[Cohen and Levcsque, 1987] P. R. Cohen and 11. J. Levesque. Persistence, intention, and commitment. In *Reasoning about Actions and Plans,* pages 297-340. Morgan Kaufmarm, Los Altos, Cal, 1987.

[Dean and Boddy, 1988] T. D ean and M. Boddv. An analysis of time-dependent planning. In *Proc. Sixth National Conference on Artificial Intelligence,* pages 49-54, Seattle, Wash, 1987.

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

[Georgeff and Ingrand, 1988] M. P. CeorgefT and F. F. Ingrand. Research on procedural reason ing systems. Final Report, Phase 1, AI Center, SRI International, Menlo Park, Cal, 1988.

[Georgeff and Lansky, 1986a] M. P. Georgeff and A. L. Lansky. Procedural knowledge. *Proc. 1KEE Special Issue on Knowledge Representation,* 74:1383-1398,1986.

[Georgeff and Lansky, 1986b] M. P. CeorgefT and A. L. Lansky. A system for reasoning in dynamic domains: Fault diagnosis on tin; space shuttle. TN 375, AI Center, SRI International, Menlo Park, Cal, 1986.

[Georgeff and Lansky, 1987] M. P. G eorgeff and A. L. Lansky. Reactive reasoning and planning: An experiment with a mobile robot. In *Proc. Sixth National Conference on Artificial Intelligence,* Seattle, Wash, 1987.

[Kaelbling, 1987] L. P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning about Actions and Plans,* pages 395-410. Morgan Kaufmarm, Los Altos, Cal, 1987.

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

[Wilkins, 1988] D. E. Wilk ins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, Los Altos, Cal, 1988.