REASONING AND PLANNING IN DYNAMIC DOMAINS: An Experiment with a Mobile Robot

Technical Note 380

April 1987

By: Michael P. Georgeff
Amy L. Lansky
Marcel J. Schoppers*

Representation and Reasoning Program
Artificial Intelligence Center
Computer and Information Sciences Division
and
Center for the Study of Language and Information

APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED

This research has been made possible by a gift from the System Development Foundation, by the Office of Naval Research under Contract No. N00014-85-C-0251, by the National Aeronautics and Space Administration, Ames Research Center, under Contract No. NAS2-12521, and by FMC Corporation under Contract No. FMC-147466.

The views and conclusions contained in this paper are those of the author and should not be interpreted as representative of the official policies, either expressed or implied, of the Office of Naval Research, NASA, or the United States Government.

---

* Now affiliated with Advanced Decision Systems, Mountain View, California.

Abstract

In this paper, the reasoning and planning capabilities of an autonomous mobile robot are described. The reasoning system that controls the robot is designed to exhibit the kind of behavior expected of a rational agent, and is endowed with the psychological attitudes of belief, desire, and intention. Because these attitudes are explicitly represented, they can be manipulated and reasoned about, resulting in complex goal-directed and reflective behaviors. Unlike most planning systems, the plans or intentions formed by the robot need only be partly elaborated before it decides to act. This allows the robot to avoid overly strong expectations about the environment, overly constrained plans of action, and other forms of overcommitment common to previous planners. In addition, the robot is continuously reactive and has the ability to change its goals and intentions as situations warrant. Thus, while the system architecture allows for reasoning about means and ends in much the same way as traditional planners, it also possesses the reactivity required for survival in highly dynamic and uncertain worlds. The system has been tested with SRI's autonomous robot (Flakey) in a space station scenario involving navigation and the performance of emergency tasks.

1 Introduction

The ability to act appropriately in dynamic environments is critical for the survival of all living creatures. For lower life forms, it seems that sufficient capability is provided by stimulus-response and feedback mechanisms. Higher life forms, however, must be able to anticipate future events and situations, and form plans of action to achieve their goals. The design of reasoning and planning systems that are *embedded* in the world and must operate effectively under real-time constraints can thus be seen as fundamental to the development of intelligent autonomous machines.

In this paper, we describe a system for reasoning about and performing complex tasks in dynamic environments, and show how it can be applied to the control of an autonomous mobile robot. The system, called a *Procedural Reasoning System* (PRS), is endowed with the attitudes of belief, desire, and intention. At any given instant, the actions being considered by PRS depend not only on its current desires or goals, but also on its beliefs and previously formed intentions. PRS also has the ability to reason about its own internal state – that is, to reflect upon its own beliefs, desires, and intentions, modifying these as it chooses. This architecture allows PRS to reason about means and ends in much the same way as do traditional planners, but provides the reactivity that is essential for survival in complex real-world domains.

For our task domain, we envisaged a robot in a space station, fulfilling the role of an astronaut's assistant. When asked to get a wrench, for example, the robot determines where the wrench is kept, plans a route to that location, and goes there. If the wrench is not where expected, the robot may reason further about how to obtain information as to its whereabouts. It then either returns to the astronaut with the desired tool or explains why it could not be retrieved.

In another scenario, the robot may be midway through the task of retrieving the wrench when it notices a malfunction light for one of the jets in the reactant control system of the space station. It reasons that handling this malfunction is a higher-priority task than retrieving the wrench and therefore sets about diagnosing the fault and correcting it. Having done this, it resumes its original task, finally telling the astronaut.

To accomplish these tasks, the robot must not only be able to create and execute plans, but must be willing to interrupt or abandon a plan when circumstances demand

it. Moreover, because the robot's world is continuously changing and other agents and processes can issue demands at arbitrary times, performance of these tasks requires an architecture that is both highly reactive and goal-directed.

We have used PRS with the new SRI robot, Flakey, to exhibit much of the behavior described in the foregoing scenarios, including both the navigational and malfunction-handling tasks. In the next section, we discuss some of the problems inherent in traditional planning systems. The architecture and operation of PRS are then described and Flakey's primitive capabilities delineated. We then give a more detailed analysis of the problems posed by this application and a review of our progress to date. We concentrate on the navigational task; the knowledge base used for jet malfunction handling is described elsewhere [16,17].

2 Previous Approaches

Most existing architectures for embedded planning systems consist of a plan constructor and a plan executor. As a rule, the plan constructor formulates an entire course of action before commencing execution of the plan [12,29,31]. The plan itself is typically composed of primitive actions – that is, actions that are directly performable by the system. The rationale for this approach, of course, is to ensure that the planned sequence of actions will actually achieve the prescribed goal. As the plan is executed, the system performs these primitive actions by calling various low-level routines. Execution is usually monitored to ensure that these routines will culminate in the desired effects; if they do not, the system can return control to the plan constructor so that it may modify the existing plan appropriately.

Various techniques have been developed for monitoring the execution of plans and replanning upon noticing plan failure [12,31]. The most common approach is to retain an explicit description of the conditions that must hold for plan execution to proceed correctly. Throughout execution, these conditions are checked periodically. If any one of them turns out to be unexpectedly false, a replanning module is invoked. This module uses various plan modification operators to change the plan or, alternatively, it can return to some earlier stage in the plan formation process and attempt to reconstruct the plan in accordance with the changed conditions.

One problem with these schemes is that, in many domains, much of the information about how best to achieve a given goal is acquired during plan execution. For example, in planning to get from home to the airport, the particular sequence of actions to be performed depends on information acquired on the way – such as which turnoff to take, which lane to get into, when to slow down or speed up, and so on. Traditional planners can cope with this uncertainty in only two ways: (1) by building highly conditional plans, most of whose branches will never be used, or (2) by leaving low-level tasks to be accomplished by fixed primitive operators that are themselves highly conditional (e.g., the intermediate-level actions (ILA) used by SHAKEY [24]). The first approach only works in limited domains – the environment is usually too dynamic to anticipate all possible contingencies. The second approach simply relegates the problem to the primitive operators themselves, and does not provide any mechanism by which the higher-level planner can control their behavior.

To overcome this problem, at least in part, there has been some work on developing planning systems that interleave plan formation and execution [9,10]. Such systems are better suited to uncertain worlds than the kind of system described above, as decisions can be deferred until they *have* to be made. The reason for deferring decisions is that an agent can acquire *more* information as time passes; thus, the quality of its decisions can be expected only to improve. Of course, because of the need to coordinate some activities in advance and because of practical restrictions on the amount of decision-making that can be accommodated during task execution, there are limitations on the degree to which such decisions may be deferred.

Real-time constraints pose yet further problems for traditionally structured systems. First, the planning techniques typically used by these systems are very time-consuming, requiring exponential search through potentially enormous problem spaces. 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. In real-world domains, unanticipated events are the norm rather than the exception, necessitating frequent replanning. The real-time constraints imposed by dynamic environments also require that a situated system be able to react quickly to environmental changes. This means that the system should be able to *notice* critical changes in the environment within an appropriately small interval of time. However, most embedded planning systems provide no mechanisms for reacting in a timely manner to new situations or goals during plan execution, let alone during plan formation.

In addition, most existing systems are overcommitted to the planning phase of their operations; no matter what the situation or how urgent the need for action, these systems *always* spend as much time as necessary to plan and reason about achieving a given goal before performing any external actions whatsoever. They lack the ability to decide when to stop planning or to reason about possible compromises between further planning and longer available execution time.

Furthermore, existing systems are usually committed to one particular planning technique, and thus cannot opt for different methods in different situations. For example, in some situations it may be best to ignore the possible side effects of actions, whereas, in other cases, reasoning about these may be critical to success of the plan. Indeed, some plans are more efficiently constructed in representational spaces that are quite different from those used to guide the robot's activities in the real world [3].

Traditional planning systems also rely excessively on constructing plans solely from knowledge about the primitive actions performable by the robot. However, many plans are not constructed from first principles, but have been acquired in a variety of other ways – for example, by being told, by learning, or through training. Furthermore, these plans may be very complex, involving a variety of control constructs (such as iteration and recursion) that are normally not part of the repertoire of conventional planning systems. Thus, although it is obviously desirable that an embedded system be capable of forming plans from first principles, it is also important that the system possess a wealth of precompiled *procedural knowledge* about how to function in the world [16].

Another disadvantage of most systems is that they commit themselves strongly to the plans they have adopted. While such systems may be reactive in the limited sense of being able to replan so as to accomplish fixed goals, they are unable to change their focus completely and pursue new goals when the situation warrants. Indeed, the very survival of an autonomous system may depend on its ability to modify its goals and intentions according to the situation. For example, in the scenario described above, the robot must be capable of deferring the task of fetching a wrench when it notices something more critical that necessitates immediate attention (such as a jet failure). The robot thus needs to be able to reason about its current intentions, changing and modifying these in the light of its changing beliefs and goals. While many existing planners have replanning capabilities, none have accommodated modifications of a system's underlying set of goal priorities. Even systems that interleave planning and execution are still strongly committed to achieving the goals initially given them.

A number of systems developed for the control of robots do have a high degree of reactivity [1,2]. Even SHAKEY [24] utilized reactive procedures (ILAs) to realize the primitive actions of the high-level planner (STRIPS). This idea is pursued further in some recent work by Nilsson [25]. Another approach is advocated by Brooks [6], who proposes decomposition of the problem into *task-achieving* units whereby distinct behaviors of the robot are realized separately, each making use of the robot's sensors, effectors, and reasoning capabilities as needed. This is in contrast to the traditional approach in which the system is structured according to *functional* capabilities, resulting in separate modules for performing such tasks as perception, planning, and task execution. Kaelbling [19] proposes an interesting hybrid architecture based on similar ideas.

These kinds of architectures could lead to more viable and robust systems than the traditional robot-control systems. Yet most of this work has not addressed the issues of general problem-solving and commonsense reasoning; the research is instead almost exclusively devoted to problems of navigation and the execution of low-level actions. These techniques have yet to be extended or integrated with systems that can change goal priorities completely, modify, defer, or abandon its plans, and reason about what is best to do in light of the immediate situation.

In real-world environments in which it is necessary to process continuous streams of sensory information and to control devices without interruption, the ability to perform multiple activities concurrently is also crucial. Although some traditional planning systems do generate plans that allow for parallel execution (e.g., SIPE [30]), they are not designed for allocating the planning and reasoning functions to separate subsystems. Some recent work, however, has concentrated on some of these issues. For example, Durfee, Lesser, and Corkhill [11] show that, by reasoning about the local plans of other subsystems, individual subsystems can form partial global plans that lead to satisfactory performance, even in rapidly changing environments. Lansky's GEMPLAN system also explicitly partitions the planning search space and plan representation according to regions of activity [22].

In sum, existing planning systems incorporate many useful techniques for constructing plans of action in a great variety of domains. However, most approaches to embedding these planners in dynamic environments are not robust enough nor sufficiently reactive to be useful in many real-world applications. On the other hand, the more reactive systems developed in robotics are well suited to handling the low-level

sensor and effector activities of a robot. Nevertheless, it is not yet clear how these techniques could be used for performing some of the higher-level reasoning desired of complex problem-solving systems. To reconcile these two extremes, it is necessary to develop reactive reasoning and planning systems that can utilize both kinds of capabilities whenever they are needed.

3 A Reactive Planning System

The system we used for controlling and carrying out the high-level reasoning of the robot is called a *Procedural Reasoning System* (PRS) [16,17]. PRS consists of a *data base* containing current *beliefs* or facts about the world, a set of current *goals* or *desires* to be realized, a set of *procedures* (which, for historical reasons, are called Knowledge Areas or KAs) describing how certain sequences of actions and tests may be performed to achieve given goals or to react to particular situations, and an *interpreter* (or *inference mechanism*) for manipulating these components. At any given moment, the system will also have a *process stack* (containing all currently active KAs), which can be viewed as the system's current *intentions* for achieving its goals or reacting to some observed situation.

The basic structure of PRS is shown in Figure 1. A brief description of each component and its usage is given below.

3.1 The System Data Base

The contents of the PRS data base may be viewed as representing the current beliefs of the system. Some of these beliefs may be provided initially by the system user. Typically, these will include facts about static properties of the application domain — for example, the structure of some subsystem or the physical laws that must be obeyed by certain mechanical components. Other beliefs are derived 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. For example, at some times PRS may believe that it is in a particular hallway, at other times in another. Updates to the data base therefore necessitate the use of consistency maintenance techniques.

The data base itself consists of a set of *state descriptions* describing what is [believed to be] true at the current instant. We use first-order predicate calculus for the

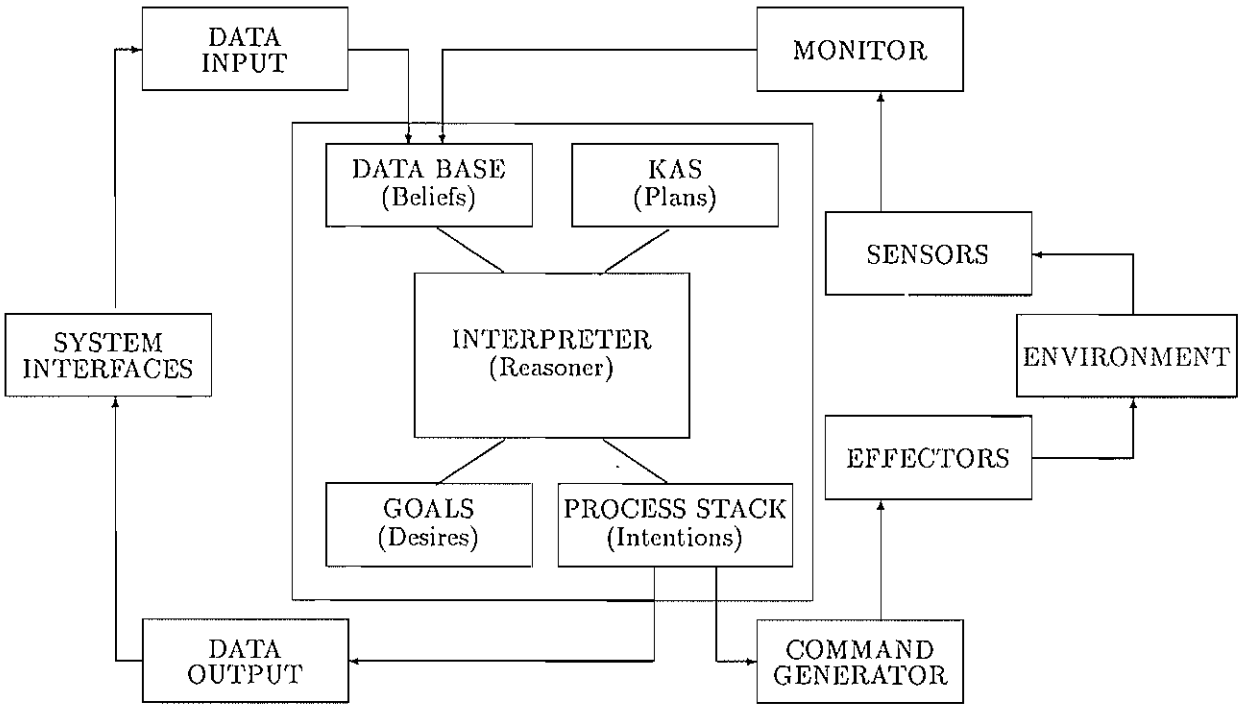


Figure 1: System Structure

state description language. Free variables, represented by symbols prefixed with $, are assumed to be universally quantified. The statement

```
(∨ (¬ (on $x table)) (red (color $x)))
```

for example, represents states of the world in which every object on the table is red.

State descriptions that describe *internal* system states are called *metalevel* expressions. The basic metalevel predicates and functions are predefined by the system. For example, the metalevel expression `(goal g)` is true if `g` is a current goal of the system.

3.2 Goals

Goals appear both on the system goal stack and in the representation of KAs. Unlike most AI planning systems, PRS goals represent desired *behaviors* of the system, rather than static world states that are to be [eventually] achieved. Hence goals are expressed as conditions over some interval of time (i.e., over some sequence of world states).

Goal behaviors may be described in two ways. One is to apply a *temporal predicate* to an n-tuple of terms. Each temporal predicate denotes an *action type* or a *set* of state sequences. That is, an expression like "`(walk a b)`" can be considered to denote the set of state sequences that embody walking actions from point `a` to `b`.

A behavior description can also be formed by applying a temporal operator to a state description. Three temporal operators are currently being used. The expression `(!`p`)`, where p is some state description (possibly involving logical connectives), is true of a sequence of states if p is true of the last state in the sequence; that is, it denotes those behaviors that *achieve* p. Thus we might use the behavior description `(!(walked a b))` rather than decide to use `(walk a b)`. Similarly, `(?`p`)` is true if p is true of the first state in the sequence – that is, it can be considered to denote those behaviors that result from a successful *test* for p. Finally, `(#`p`)` is true if p is preserved (maintained invariant) throughout the sequence.

Behavior descriptions can be combined by means of the logical operators ∧ and ∨. These denote, respectively, the intersection and union of the component behaviors. For example, `(!(walked a b)` ∧ `#(holding cup1))` describes those behaviors in which one walks from `a` to `b` while holding a particular cup.

As with state descriptions, behavior descriptions are not restricted to describing the external environment, but can also characterize the internal behavior of the system. Such behavior specifications are called metalevel behavior specifications. One important metalevel behavior is described by an expression of the form `(=> p)`. This specifies a behavior that places the state description p in the system data base. Another way of describing this behavior might be `(!(believe p))`.

3.3 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). Each KA consists of a *body*, which describes the steps of the procedure, and an *invocation condition* that specifies under what situations the KA is useful.

The body of a KA is represented as a graphic network and can be viewed as a plan or plan schema. However, it differs in a very important way from the plans produced by most AI planners: it does not consist of possible sequences of primitive actions, but rather of possible sequences of *subgoals* to be achieved. Thus, the bodies of KAs are much more like the high-level "operators" used in such planning systems as NOAH [27] and SIPE [30]. They differ in that (1) the subgoals appearing in the body can be described by complex temporal expressions and (2) the allowed control constructs are richer and include conditionals, loops, and recursion. One important advantage of using abstract subgoals instead of fixed calls to actions is that the knowledge expressed in any given KA is largely independent of other KAs, thereby providing a very high degree of modularity. It is thus possible to build domain knowledge incrementally, with each component KA having a well-defined and easily understood semantics.

The invocation part of a KA contains an arbitrarily complex logical expression describing under what conditions the KA is useful. Usually this consists of some conditions on current system goals (in which case, the KA is invoked in a goal-directed fashion) or current system beliefs (resulting in data-directed or *reactive* invocation), and may involve both. Together the invocation condition and body of a KA express a declarative fact about the effects of performing certain sequences of actions under certain conditions.[1]

[1] More specifically, a KA can be viewed as a statement attesting that, if the facts described in the invocation part of the KA hold at the beginning of execution, achieving the sequence of goals described

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 relevant KAs, determining how to achieve a conjunction or disjunction of goals, and computing the amount of additional reasoning that can be undertaken, given the real-time constraints of the problem domain. Metalevel KAs may of course utilize knowledge specifically related to the problem domain. In addition to user-supplied KAs, each PRS application contains a set of system-defined default KAs. These are typically domain-independent metalevel KAs.

3.4 The Process Stack and 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 data base. Given these extant goals and beliefs, a subset of KAs in the system will be relevant (i.e., applicable). One of these relevant KAs will then be chosen for execution by placing it on the process stack.

In determining KA applicability, the interpreter will not automatically perform any deduction. Both beliefs and goals are matched 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 *choose* whether or not to deduce (1) further consequences of the set of explicit beliefs held in the data base or (2) what other goals would imply attainment of any goals already decided upon. Any such deductions, however, would have to be made by appropriate metalevel KAs, which themselves would be interruptible. In this way, the reactivity of the system is retained.

In the course of executing the chosen KA, new subgoals will be posted and new beliefs derived. When new goals are pushed onto the goal stack, the interpreter checks to see if any new KAs are relevant, chooses one, places it on the process stack, and begins executing it. Likewise, whenever a new belief is added to the data base, the

by some path in the KA body will also achieve the goals given in the invocation part. A more complete description of the declarative and operational semantics of PRS can be found elsewhere [16].

interpreter will perform appropriate consistency maintenance procedures and possibly activate other relevant KAs. During this process, various metalevel KAs may also be called upon to make choices among alternative paths of execution, choose among multiple applicable KAs, decompose composite goals into achievable components, and make other decisions.

This results in an interleaving of plan selection, formation, and execution. In essence, the system forms a partial overall plan, determines a means of accomplishing the first subgoal of the plan, acts on this, further expands the near-term plan of action, executes further, and so on. At any time, the plans the system is intending to execute (i.e., the selected KAs) are both *partial* and *hierarchical* — that is, while certain general goals have been decided upon, the specific means for achieving these ends have been left open for future deliberation. By finding and executing relevant procedures only when needed and only when sufficient information is available for making prudent decisions, the system stands a better chance of achieving its goals under real-time constraints.

Unless some new fact or request activates some new KA, PRS will try to fulfill any intentions it has previously decided upon. But if some important new fact or request does become known, PRS will reassess its goals and intentions, and then 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 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 the space station scenario, this happens quite frequently as emergencies of various degrees of severity occur in the process of handling other, less critical tasks. PRS can even alter its intentions regarding its own reasoning processes – for example, it may decide that, given the current situation, it has no time for further reasoning and so must act immediately.

PRS has also 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, PRS is able to notice newly applicable KAs after every primitive action it takes. Given that each such primitive action has an execution time of at most t, and given that the PRS interpreter takes at most time s to unify KAs with its current beliefs and goals, PRS has a guaranteed *reactivity delay* of at most $t + s$. Of course, this represents only the maximum amount of time PRS may take to *notice* requests

from other agents or changes in the environment. It may take longer for PRS to decide what to do and to perform some appropriate action.

3.5 Multiple Asynchronous PRSs

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 messages are written into the data base of the receiving PRS, which must then decide what to do, if anything, with the new information. As a rule, this decision is made by a fact-invoked KA (in the receiving PRS), which responds upon receipt of the external message. In accordance with such factors as the reliability of the sender, the type of message, and the beliefs, goals, and current intentions of the receiver, it is determined what to do about the message – for example, to acquire a new belief, establish a new goal, or modify intentions.

4 Flakey the Robot

Flakey, designed and built in SRI's Artificial Intelligence Center, is being used by several research teams to test a number of ideas on software organization. It contains two onboard computers, a Sun II workstation (with 42 Mbyte disk) and a Z80 microprocessor. The Z80 is the low-level controller, receiving instructions from and returning current information to the Sun. The Sun, in turn, can be connected to an Ethernet cable, allowing the robot to operate in either stand-alone or remote-control modes. The Sun can also be accessed from a small console on Flakey itself.

The Z80 controls 12 sonars, 16 bumper contacts, and 2 stepper motors for the left and right wheels. Voice output and video input are managed by the Sun. A robot arm will be added in the future. The application described here uses only the sonars, voice, and wheels.

The 12 sonars are located approximately 5 inches off the ground – four facing forward, four backward, and two on each side. To obtain a sonar reading, the Sun

must issue a request to the Z80 and then wait until the result has been returned. While waiting, the Sun can continue with other processing. At present, the Sun can obtain no more than a few sonar readings per second.

The motors for the left and right wheels can be controlled independently, again by having the Sun send a request to the Z80. For each wheel a desired distance, a maximum forward speed, and a desired acceleration can be specified. The Z80 uses the given acceleration to achieve the maximum speed compatible with the desired distance.

Directional changes are accomplished by requesting different speeds for the two wheels. When the robot is stationary, this can be reduced to a simple rotation; when the robot is moving, more complex algorithms are required. Changes in direction are much more difficult to program when they must be negotiated during a forward acceleration.

As well as receiving the desired values of distance, speed, and acceleration from the Sun, the Z80 transmits current actual values to the Sun. This is done by means of interrupts that occur at a rate of approximately fifty times per second. The Z80 also runs a position integrator, thus making available the robot's position and orientation relative to particular reference axes. In line with our desire to avoid reliance on dead reckoning, however, we did not use the position integrator for the top-level navigation task; it was employed, however, for such low-level tasks as estimating the robot's alignment within a hallway.

There is significant noise in every measurement available to the Sun. The sonars, while generally accurate to within 5 millimeters, can occasionally return invalid readings and can also fail to see an object if the angle of incidence is too high. Furthermore, Flakey's sonars sense the closest object within a 30-degree cone, so that open doorways are not seen until the sonars are well past the doorpost. Similarly, Flakey will stop within about 5 millimeters of the requested distance and will travel at speeds that vary by up to 10 millimeters per second above and below the requested maximum speed.

5 The Domain Knowledge

The scenario described in the introduction includes problems of route planning, navigation to maintain the route, and such tasks as malfunction handling and requests for

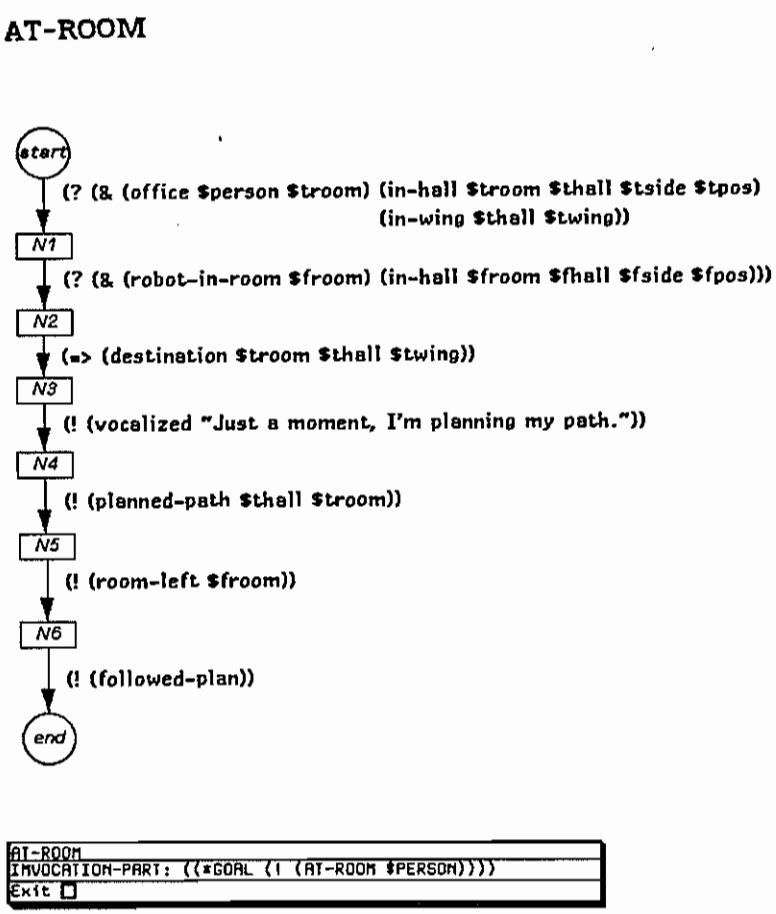


Figure 2: The Top-Level Strategy

information. We shall concentrate herein on the tasks of route planning and navigation. However, it is important to realize that the knowledge representation provided by PRS is used for reasoning about all tasks performed by the system.

The way the robot (Flakey, that is, under the control of PRS) solves the tasks of the space station scenario is roughly as follows. To reach a particular destination, it

knows that it must first plan a route and then navigate to the desired location (see the KA depicted in Figure 2). In planning the route, the robot uses knowledge of the station's topology to work out a path to the target location, as is typically done in navigational tasks for autonomous robots [7,23]. The topological knowledge, however, is of a very high-level form, stating which rooms are in which corridors and how the latter are connected. The route plan formed by the robot is also high-level, typically having the following form: "Travel to the end of the corridor, turn right, then go to the third room on the left." The robot's knowledge of the problem domain's topology is stored in its data base, while its knowledge of how to plan a route is represented in various route-planning KAs (see Figures 4, 5, and 6). This is quite different from the approach adopted by traditional AI planners, which would find a route by symbolically executing the actual operators that specify potential movements through rooms and along hallways.

A different set of KAs is used for interpreting and traversing the route mapped out by the route-planning KAs (see Figures 7, 8, 9, and 10). The navigational KAs carry out such tasks as sensing the environment, deciding when to turn, adjusting bearings where necessary, and counting doors and other openings.

Yet other KAs perform the various other tasks required of the robot. Many of these are described by us elsewhere [17]. Metalevel KAs choose among different means of realizing any given goal and determine the respective priority of tasks when mutually inconsistent goals arise (such as diagnosing a jet failure and fetching a wrench). If the robot's route plan fails, the route-planning KAs can again take over and replan a new route to the target destination. In the implementation described herein, however, we have not provided any KAs for re-establishing location once the robot has left its room of departure. Consequently, it does not possess any replanning capability at the present time.

5.1 The Planning Space

As stated above, the robot's route planning is done in a very abstract space containing only topological information about how the rooms and hallways connect. It is in fact the kind of map found in street or building directories, stripped of precise distances and angles. This is quite natural: when one thinks of going home from the office, one considers primarily the topology of the hallways, footpaths, and roads to be followed,

not precisely how long each of them is, nor the consequences of lateral drift; such factors involve too low a level of detail to be taken into account before setting out along the chosen route.

The three primary KAs used to plan paths are shown in Figures 4, 5, and 6. Given knowledge of the starting and end points, they first select some intermediate point. They then repeat the process for the two resulting subpaths until all paths have been reduced to straight-line trajectories along single hallways. Although it is not the planner we would advocate for more general route planning, it is quite sufficient for our purposes. Indeed, top-level route planning is the simplest aspect of the navigational task.

The topological information needed for route planning is stored in the system data base as a set of facts (beliefs) about the way wings, hallways, and rooms are connected. These include facts of the form `(conn j1 k1 j4 direct)` (i.e., hallway j1 is connected to hallway k1 directly via hallway j4), `(in-wing j1 jwing)` (i.e., hallway j1 is in wing jwing), and `(in-hall ej225 j1 east 14)` (i.e., room ej225 is in hall j1 on the east side of the hall, fourteen rooms from the end). A typical route plan constructed by the path-planning KAs is shown in Figure 3. This plan, formed to satisfy the goal of reaching a target room (ej270 in wing j2) from the robot's location after it exits its current room (i.e., just outside room ej233 in wing j1), was produced in less than a second. No further predictive planning is required for the robot to negotiate the path.

(path hall j1 j4)
(path hall j4 j2)
(path hall j2 ej270)

Figure 3: Route from ej233 to ej270

It is important to emphasize that, even during this relatively short route-planning stage, the robot remains continuously reactive. Thus, for example, should the robot notice indication of a jet failure on the space station, it might well decide to interrupt its route planning and attend instead to the task of remedying the jet problem.

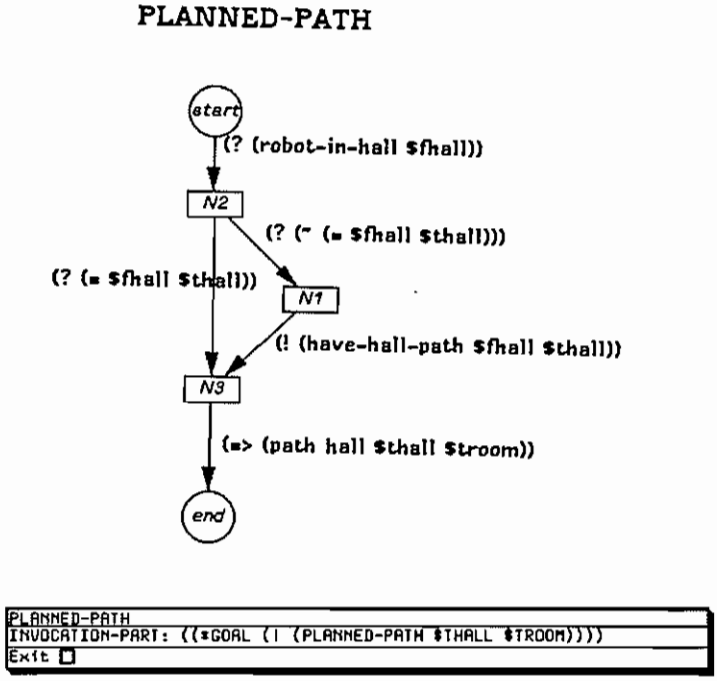


Figure 4: Route-Planning KA

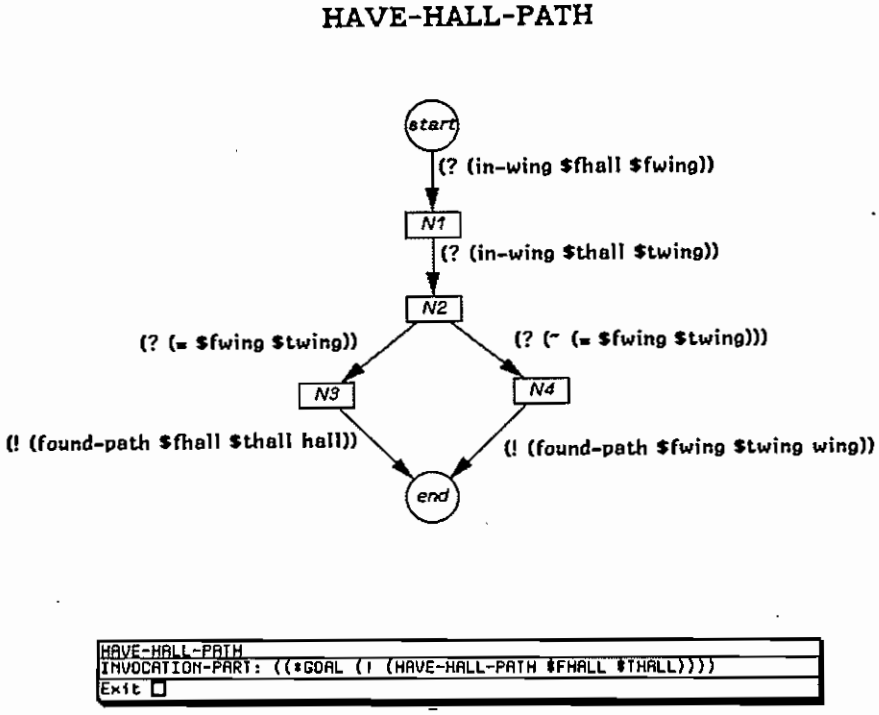


Figure 5: Route-Planning KA

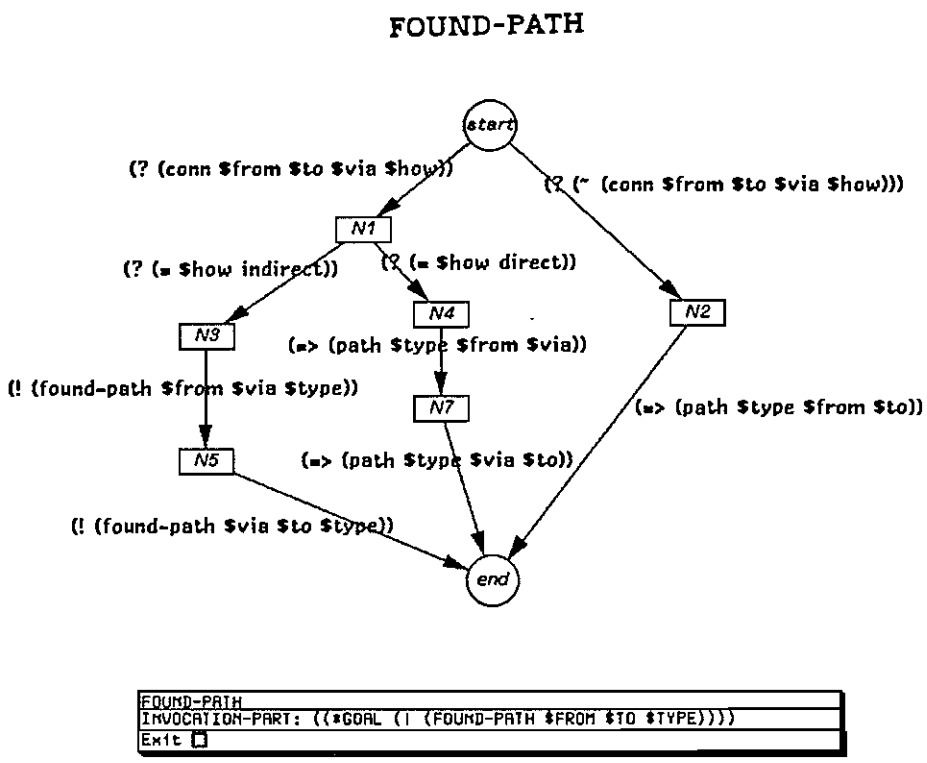


Figure 6: Route-Planning KA

5.2 Reactive, Goal-Directed Behavior

The KAs used to navigate the route fall into three classes: (1) those that interpret the route plan and establish intermediate target locations, (2) those that are used to follow the path, and (3) those that handle critical tasks such as obstacle avoidance and reacting to emergencies. Each KA manifests a self-contained behavior that may include both sensory and effector components. Moreover, the set of KAs is partitioned naturally according to its level of functionality [6]: low-level functions (emergency reactions, obstacle avoidance, etc.), middle-level functions (following already established paths and trajectories), and higher-level functions (route planning and figuring out how to execute a topological route). Many of these KAs can be simultaneously active, performing their function whenever they may be applicable. Thus, while the robot is trying to follow a path down a hallway, an obstacle avoidance procedure may simultaneously cause the robot to veer slightly from its original path.

Once a route plan has been decided upon it must be converted into some usable form. One approach would be to represent the plan shown in Figure 3 as a procedural KA containing the goals "go to the j1-j4 junction," "go to the j4-j2 junction," and so on. Another approach – the one we have adopted – is to define a group of KAs that react to the presence of a route plan (in the data base) by translating it into the appropriate sequence of subgoals. Each leg of the original route plan generates subgoals – such as turning a corner, travelling along the hallway, and updating the data base to indicate progress. The second group of navigational KAs reacts to these goals by actually doing the work of reading the sonars, interpreting the readings, counting doorways, aligning the robot in the hallway, and watching for obstacles up ahead.

For example, let us consider the KAs in Figures 7 8, and 9. After having used the KA in Figure 2 to plan a path, the robot acquires the goal `(!(room-left $froom))`, where the variable `$froom` is bound to some particular constant representing the room that the robot is trying to leave. The KA in Figure 9 will respond, causing the robot to perform the steps for leaving the given room. The last step in this KA will insert a fact into the system data base of the form `(current-origin $froom $fhall)`, where the variables are again bound to specific constants. Next, the KA in Figure 2 issues the command `(!(followed-plan))`. This activates the KA in Figure 7, which repeatedly activates the KA in Figure 8, following each leg of the plan until the goal destination is reached. Beliefs of the form `(current-origin $locale $spot)` are repeatedly used

to readjust the robot's bearings and knowledge about its whereabouts.

A third group of KAs reacts to contingencies encountered by the robot as it interprets and follows its path. For example, these will include KAs that respond to the presence of an obstacle ahead (see Figure 10) or the fact that an emergency light has been seen. These reactive KAs are invoked solely on the basis of certain facts' becoming known to the robot. Implicit in their invocation, however, is an underlying goal to "avoid obstacles" or "remain safe."

Since a fact-invoked KA can be executed as soon as its triggering facts are known, the KAs invoked by these contingencies can interrupt whatever else is happening. Of course, this may not always be desirable. Ideally, domain-specific metalevel KAs should ascertain whether and when pre-emption is appropriate, but, at this stage of the project, we have not used metalevel KAs other than those provided as PRS defaults (which result in immediate pre-emption). An alternative to pre-emption is to send a contingency message to another PRS instantiation that can process that message concurrently.

5.3 Parallelism and Mediation

Because of real-time constraints and the need for performing several tasks concurrently, it is desirable to use multiple instances of PRS running in parallel. In particular, parallelism can be used for handling contingencies without interrupting other ongoing tasks. Multiple PRS instantiations can also serve as information filters to protect other instantiations from a barrage of uninteresting sensory information. Moreover, parallelism provides for robustness and reliability.

For example, as the robot rolls down a hallway, it fires its sonars to determine how far it is from the walls, and also to count doors. Suppose it decides that the walls are too close and that, consequently, a change in course is warranted. Because the actions of speeding up and slowing down are not accomplished instantaneously, changing course may take as long as two seconds. This is long enough for the robot to roll past a doorway. If the procedure that monitors sonar readings is interrupted to effect the course change, the robot might completely miss sensing a door. Conversely, delaying course changes for the sake of sonar monitoring could make a collision with a wall inevitable. Of course, travel at lower speeds would solve the problem – but it would also render the robot too slow to be useful.

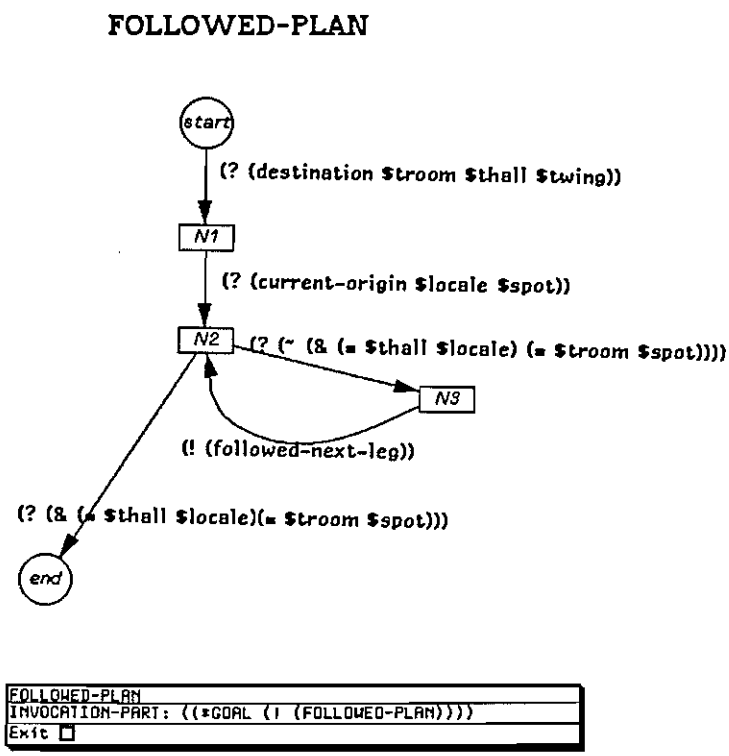


Figure 7: Plan Interpretation KA

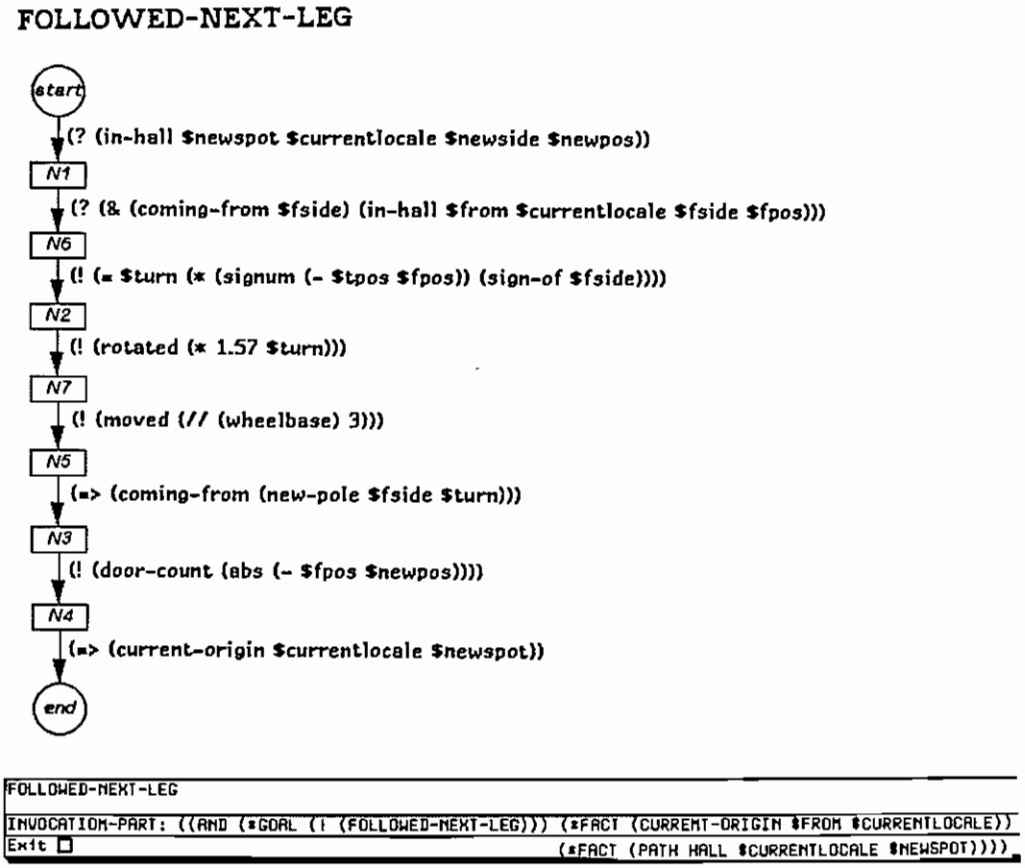
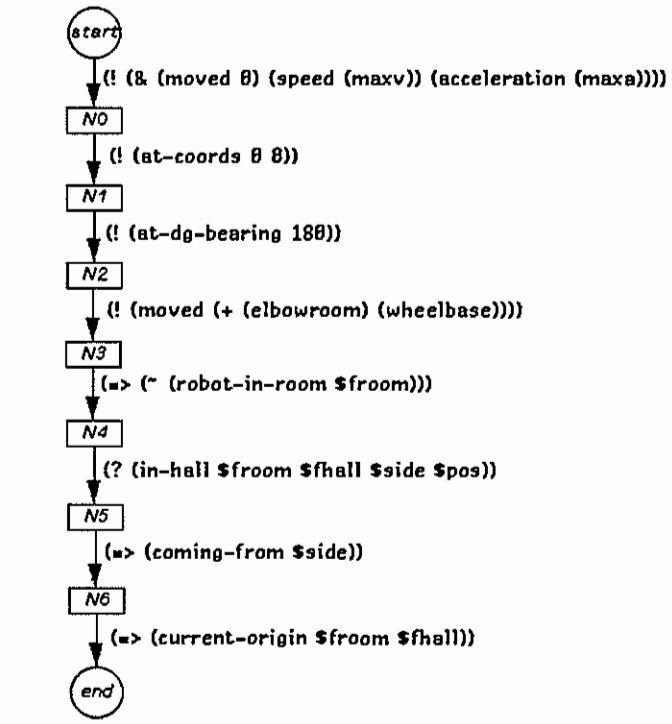


Figure 8: Plan Interpretation KA

ROOM-LEFT



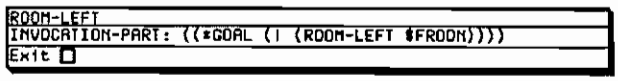


Figure 9: A Route Navigation KA

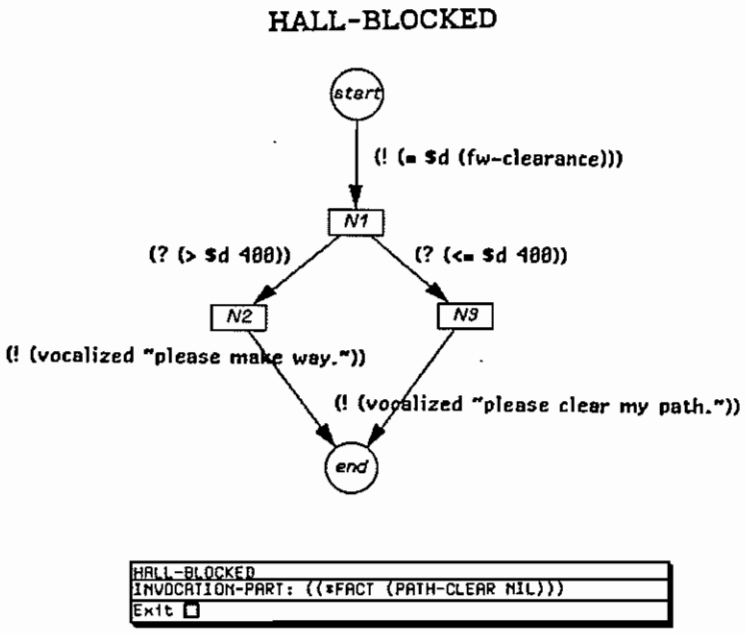


Figure 10: A Reactive KA

The most effective way to handle this problem is to allow multiple PRS instantiations to act concurrently. Running several instantiations asynchronously entails its own problems, however. For example, it is desirable to have one PRS instantiation devoted to the task of keeping the robot in the center of the hallway, while another drives the robot to the target location and adjusts speed commensurately (e.g., slowing down when approaching the target). Changes in course are effected by changing the relative velocities of the two wheels, depending on their current velocity; changes in speed are effected by changing the acceleration of the wheels. The problem is that, if both tasks need to be performed at once, the required wheel operations may interfere with one another. This is an interesting example of a situation in which domain-independent decomposition operators will not work: because of the real-time constraints of the domain, it is not suitable to achieve one goal (say, a change in direction) and subsequently achieve the other (change in speed); neither can each goal be attained independently, as the means for doing so interact with one another.

To mediate between these interacting goals, we chose to implement a third PRS to control the wheels. This PRS instantiation can be viewed as a virtual controller that is capable of accepting requests for changes in either speed or direction asynchronously, and can fulfill both kinds of requests simultaneously. In this respect, it serves as a special-purpose solution to a particular kind of conjunctive goal; goals to change both speed and direction are decomposed into independent goals to change the speeds of the left and right wheels respectively.

Related to the problem of interacting goals is that of goal conflict: just as one may have possibly conflicting beliefs about a situation that needs to be resolved (the problem of situation assessment), one may also have conflicting goals (or desires) that require mediation [19]. For example, the virtual controller often gets conflicting speed requests from KAs: the hallway-traversal KA might request that a certain velocity be maintained, the KA that detects proximity to the target may request a decrease in velocity, and the KA that detects obstacles could request that the robot stop altogether. At the same time, other KAs might request changes in direction to stay in the center of the hall or to pass around small obstacles.

To reconcile these conflicting goals, the virtual controller has to be able to reason about their relative urgency and criticality. This, in turn, may involve further communication with the goal-requesting systems. Our present solution is to define domain-dependent mediators wherever necessary; at present, however, no general technique for

solving this problem has been attempted.

5.4 Coping with Reality

In our initial implementation of the robot application, we used a single PRS instantiation interacting with a robot simulator, all running on the Symbolics 3600. This worked well, confirming the system's ability to control complex autonomous devices. We then began work on driving the real robot. Speed constraints and requirements led to the use of multiple PRS instantiations. Indeed, this transfer process took considerably longer than estimated. Two major problems caused this divergence between expectations and reality.

First, because PRS was implemented on a Lisp machine, interaction with Flakey could be effected only via an Ethernet cable. Software for remote procedure calls over the ethernet limited communication to 15 function calls per second – too slowly for a timely response to sensor input. Consequently, we had to transfer much of the PRS functionality to Flakey's Sun. This required translating the functionality of the lower-level KAs into C code, as well as into explicit coding for handling messages and clock signals. Unfortunately, Flakey's operating system also did not support interprocess communication at the bandwidth and with the efficiency we wanted. We were thus compelled to implement communication through shared memory, with all the concomitant synchronization code thereby entailed. After these efforts, the information flowing over the Ethernet was at the level of "move N doors" (PRS to Flakey) and "I'm stopping for an obstacle" (Flakey to PRS). Obviously, the translated system is no longer constructed solely from instances of PRS. As a result, our final implementation is considerably more constrained than the simulation version in its capacity to reason about its low-level actions and to react appropriately to changing goals.

The second obstacle to translating from our simulated application to one that could function in the physical world is the nature of the real world itself. A realistic environment is simply not controlled enough to allow efficient debugging. It is difficult to repeat experiments (and get the same bugs), time-delays become critical, and the behavior of real sensors and effectors can differ significantly from their simulated counterparts.

The configuration of our current application system is shown is Figure 11. Three machines are involved: a Symbolics 3600, a Sun, and a Z80, running six application

processes. The wheels and sonars, which may be regarded as physical processes, are also depicted. The rectangular box represents the Sun's shared-memory area; arrows represent interprocess communication.

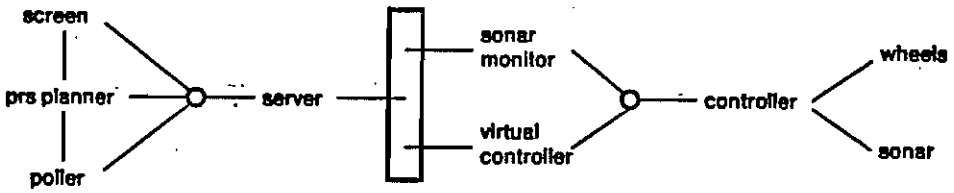


Figure 11: Processes Used in the Implementation

6 Discussion

The primary purpose of this research was to show that the abstract architecture of PRS was suited to reasoning and planning in dynamic and uncertain worlds. The experiments we performed with Flakey proved quite successful, and the robot managed to accomplish a considerable portion of the two scenarios described in Section 1, including both the navigational and malfunction-handling tasks.

In particular, the robot managed to plan a path to the target room, maneuver its way out of the room in which it was stationed, and navigate to its destination via a variety of hallways, intersections, and corners. It maintained alignment in the hallways, avoided obstacles, and stopped whenever its path was completely blocked. If it noticed a jet malfunction on the space station (simulated by human interaction via a keyboard), it would interrupt whatever it was doing (route planning, navigating the hallways, etc.) and attend to diagnosing the problem. The diagnosis performed by the

robot was quite complex and followed the actual procedures as used for NASA's space shuttle [17].

The features of PRS that, we believe, contributed most to this success were (1) its partial planning strategy, (2) its reactivity, (3) its use of procedural knowledge, and (4) its metalevel (reflective) capabilities. As mentioned earlier, when one operates in these kinds of environments it is simply impractical to plan in advance when there is a substantial prospect of plan invalidation. It is usually wiser to form a partial plan, then wait to see what the world will be like before expanding it further. The partial hierarchical planning strategy and the reflective reasoning capabilities of PRS proved to be well suited to the robot application, yet still allowed the system to plan ahead when necessary. By finding and executing relevant procedures only when sufficient information was available to make wise decisions, the system stood a better chance of achieving its goals under the stringent real-time constraints of the domain.

For example, the speed and direction of the robot were determined during plan execution, and depended on such things as the proximity of obstacles and the actual course of the robot. Even the method for determining the course depended on the immediate situation, such as whether the robot was between two hallway walls, adjacent to an open door, at a T-intersection, or passing an unknown obstacle. Similarly, the choice of how to normalize fuel or oxidant tank pressure while handling a jet failure depended on observations made during the diagnostic process.

Because PRS expands plans dynamically and incrementally, there were also frequent opportunities for it to react to new situations and changing goals. The system was therefore able to modify its intentions rapidly on the basis of what it perceived at any given moment as well as upon what it already believed, intended, and desired. For example, when the system noticed a jet-fail alarm while it was attempting to fetch a wrench, it had the ability to reason about the priorities of these tasks and, if it so decided, to suspend the wrench-fetching task while it attended to the jet failure. Indeed, the system even continued to monitor the world while it was *planning* its route and could interrupt the planning whenever the situation demanded.

The wealth of procedural knowledge possessed by the system was also critical in allowing the robot to operate effectively in real-time and to perform a variety of very complex tasks. In particular, the powerful control constructs allowed in KAs (such as conditionals, loops, and recursion) proved highly advantageous. For example, the

iterative constructs in PRS allowed us to specify plans such as "maintain speed at 400 mm/sec until N doorways have been observed." We were thus able to dispense with coordinate grids and dead reckoning: we could specify the robot's behaviors relative to conditions that changed over time.

PRS also makes it possible to have a large number of diverse KAs available for achieving a goal. Each may vary in its ability to accomplish a goal, as well as in its applicability in particular situations. Thus, if there is insufficient information about a given situation to allow one KA to be used, another (perhaps one less reliable) might be available instead. For example, if a topological map of an area had been unavailable for planning pnrposes, its absence would not necessarily have rendered the robot ineffective; there might, for example, have been some other KA that would have set the robot off in the general direction of the target. Parallelism and reactivity also helped in providing robustness. For example, if one PRS instantiation were busy planning a route, other instantiations could remain active, monitoring environmental changes, keeping the robot in a stable configuration, and avoiding dangers.

The metalevel reasoning capabilities of PRS were particularly important in managing the application of the various KAs in different situations. Such capabilities can be critical in deciding how best to meet the real-time constraints of a domain. However, the current system was really too simple to serve as an adequate test of the system's metalevel reasoning abilities; indeed, the system performed quite well with only a few [well-chosen] metalevel KAs.

PRS also meets many of the criteria of rational agency that have been advanced in the philosophical literature on practical reasoning [4]. Impelled by the need to explain resource-boundedness and inter- and intra-agent coordination, recent work in the philosophy of action has moved beyond belief-desire architectures for rational agents to consideration of the nature of plans and intentions, as well as the nature of intention formation.

In particular, plans are viewed as being subject to two kinds of constraints: *consistency constraints* and those that require *means-ends coherence.* Consistency constraints specify that an agent's plans must be both internally consistent and consistent with its beliefs and goals. Moreover, it should be possible for an agent's plans to be executed successfully (that is, it should choose truly applicable plans) in a world in which its beliefs are true. Means-ends coherence is required when a particular plan of action is

adopted. Though they are partial, plans have to be filled in to a certain extent, as time goes by, with subplans that concern such plan elements as means, preliminary steps, and relatively detailed courses of action. These subplans must be at least as extensive as the agent believes is necessary for successful plan execution.

These constraints on the beliefs, desires (goals), and intentions of an agent are largely realized by PRS and, as such the system can be viewed as an implementation of a rational agent. In addition, our notion of intention satisfies the major requirements put forth by Bratman [4], who considers intentions to have the following properties:

- They lead to action
- They are parts of larger plans
- They involve commitment
- They constrain the adoption of other intentions
- They are adopted relative to the beliefs, goals, and other intentions of the system.

Of course, the system we used for controlling Flakey did not come close to manifesting the behavioral complexity of real rational agents; nevertheless, it did cope remarkably well with the dynamic and uncertain world in which it was embedded.

PRS also achieves the same kind of task decomposition advocated by Brooks [6], in which distinct *behaviors* of the robot are realized separately, each making use of the robot's sensory, effector, and reasoning capabilities as needed. Each KA defines a particular behavior of the system; that behavior can involve both the processing of sensory information and the execution of effector actions. In the robot application, for example, we used a KA that manifested a behavior to remain clear of obstacles, another KA whose behavior was to keep a straight course in a corridor, and yet another that chose and traversed routes from one room to another.

PRS thus provides the same positive benefits as the system proposed by Brooks [5]:

- "There are many parallel paths of control through the system [Many different procedures can be used in a given situation]. Thus, the performance of the system in a given situation is not dependent on the performance of the weakest link in

that situation. Rather, it is dependent on the strongest relevant behavior for the situation."

- "Often more than one behavior is appropriate in a given situation. The fact that behaviors can be generated by parallel systems [multiple PRS instances] provides redundancy and robustness to the overall system. There is no central bottleneck [through which all the processing or reasoning must occur]."
- "With some discipline in structuring the decomposition, the individual task-achieving behaviors can run on separate pieces of hardware. Thus, [the design] leads to a natural mapping of the complete intelligent system onto a parallel machine. The benefits are threefold: (1) redundancy again; (2) speedup; and (3) a naturally extensible system."

The main difference between PRS and the Brooks architecture is that we employ a more general mechanism for selecting among appropriate behaviors than he does: whereas Brooks uses inhibitory and excitatory links to integrate the set of behaviors defined by each of the system's functional components, we use general metalevel procedures and communication protocols to perform the selection and integration. Of course, such generality will likely preclude meeting some of the real-time constraints of the environment, in which case the metalevel procedures might have to be compiled into a form closer to that envisaged by Brooks. Similarly, while our system maps naturally onto coarse-grained parallel machines, sophisticated compilation techniques would be required to map the lower-level functions onto highly parallel architectures.

Finally, we wish to emphasize that the *abstract* architecture of PRS, not its actual implementation, was our primary concern. That is, while we believe that attributing beliefs, desires, and intentions to autonomous systems can aid in specifying complex behaviors of these systems, and can assist in communicating and interacting with them, we are not insisting that such systems actually incorporate distinct data structures that explicitly represent these psychological attitudes. We can instead view the specification of the PRS system, together with the various metalevel and object-level KAs, simply as a *description* of the robot's desired behavior. This description, suitably formalized,[2] could be realized in (or compiled into) any suitable implementation we choose. In

[2]While the semantics of the KAs is formally defined [16,18], we have yet to formally specify the operation of the interpreter underlying PRS. If this were done, we would then have a completely formal specification of the robot's desired behavior.

particular, the beliefs, desires, and intentions of the robot may no longer be explicitly represented within the system. Some interesting work on this problem is now being done by Rosenschein and Kaelbling [26].

7 Limitations of the Current Implementation

The primary thrust of this work has been to evaluate an architecture for autonomous systems that provides a means of achieving goal-directed, yet reactive behavior. We have made enough progress to show that this approach works. However, the research is only in its initial stages and there are a number of limitations of the current implementation that still need to be addressed.

First, there is a class of facts that the current system must be told – for instance, the robot's starting location. If the robot is initialized in some unknown position on the topological map, the planner will abort. It would be easy to solve such problems by including KAs that ask for the missing information, but a completely autonomous recovery would be a much more challenging task. Possible approaches might involve exploration of the terrain (including movement within the neighboring area) and pattern matching onto known landmarks.

Second, there are many assumptions behind the procedures (KAs) used. For example, we have assumed that hallways are straight and corners rectangular; that all hallways maintain a fixed width for their entire length (except for doorways and intersecting halls); that there is only one layer of obstacles in front of any wall (nowhere, for instance, is there any unexpected object in front of a cupboard); that all doors are open and unobstructed; and that other objects move much more slowly than the robot.

We have also made assumptions that limit the robot's reactivity. For example, we assume that the robot does in fact arrive at the junction it planned to reach. If the robot miscounts the doorways, it will stop in the wrong place, turn, and start the next leg of its journey without realizing its mistake. The result is generally that the robot will be found begging a wall to "please make way." If the robot realized it was in the wrong position, it could replan to achieve its goals. However, because we assume that the door count is always right, the route planner is never reinvoked.

In addition, some goals are not made as explicit as one would like, but are implicit in the KAs used by the robot. For example, the robot is designed to move as fast as possible without miscounting doorways, and to travel along the center of the hallway. Such goals are not represented explicitly within KAs. Handling the first kind of goal ("move as fast as possible") would be relatively straightforward, requiring simply that the system be provided with axioms relating robot speed and perceptive capabilities. However, it is not clear just how one should explicitly represent the second kind of goal, whereby the system attempts to maintain a particular condition but at best expects only to approximate it.

The present system also does not reason about other subsystems (i.e., other PRS instantiations) in any but the most elementary manner. However, the message-passing mechanisms we have employed should allow us to integrate more complex reasoning about interprocess communication, such as described by Cohen and Levesque [8]. Reasoning about process interference and synchronization is also important when concurrency is involved. The mechanisms developed by us for reasoning about these problems [14,15,13,20,21,28] could also potentially be integrated within PRS. Our future research plans include work both on communication and on synchronization within the PRS framework.

Finally, a greater variety of KAs and increased parallelism would have been preferable, allowing the robot to perform its tasks under more demanding conditions. For example, we could have included many additional low-level procedures for, say, avoiding dangers and exploring the surroundings. This would have provided a much more severe test of the system's ability to coordinate various plans of action, modify intentions appropriately, and shift its focus of attention.

Acknowledgments

Pierre Bessiere, Joshua Singer and Mabry Tyson helped in the development of PRS and extended the implementation as needed. Stan Reifel and Sandy Wells designed Flakey and its interfaces, and assisted with the implementation described herein. We have also benefited greatly from our participation in and interactions with members of CSLI's Rational Agency Group (RATAG), especially Michael Bratman, Phil Cohen, Kurt Konolige, David Israel, and Martha Pollack. Leslie Pack Kaelbling, Stan Rosenschein, and Dave Wilkins also provided helpful advice and interesting comments.

References

[1] J. S. Albus. *Brains, Behavior, and Robotics.* McGraw-Hill, Peterborough, New Hampshire, 1981.

[2] J. S. Albus, A. J. Anthony, and R. N. Nagel. Theory and practice of hierarchical control. In *Proceedings of the Twenty-Third IEEE Computer Society International Conference*, 1981.

[3] S. Amarel. On representations of problems of reasoning about actions. *Machine Intelligence 3*, 1968. D. Michie, (ed.), Edinburgh University Press, Edinburgh, Scotland.

[4] M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Massachusetts, forthcoming.

[5] R. A. Brooks. *Achieving Artificial Intelligence Through Building Robots.* Technical Report 899, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1986.

[6] R. A. Brooks. *A Robust Layered Control System for a Mobile Robot.* Technical Report 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1985.

[7] R. A. Brooks. Visual map making for a mobile robot. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 824–829, St. Louis, Missouri, 1985.

[8] P. R. Cohen and H. J. Levesque. Speech acts and the recognition of shared plans. In *Proceedings of the Twenty Third Conference of the Association for Computational Linguistics*, pages 49–59, Stanford, California, 1985.

[9] P.R. Davis and R.T. Chien. Using and reusing partial plans. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 494, Cambridge, Massachussets, 1977.

[10] E. H. Durfee and V. R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 58–64, Philadelphia, Pennsylvania, 1986.

[11] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Increasing coherence in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 1025–1030, Los Angeles, California, 1985.

[12] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[13] M. P. Georgeff. Actions, processes, and causality. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Morgan Kaufmann, Los Altos, California, 1987.

[14] M. P. Georgeff. Communication and interaction in multiagent planning. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 125–129, Washington, D.C., 1983.

[15] M. P. Georgeff. A theory of action for multiagent planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, Austin, Texas, 1984.

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

[17] 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.

[18] M. P. Georgeff, A. L. Lansky, and P. Bessiere. A procedural logic. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, 1985.

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

[20] A. L. Lansky. *Behavioral Specification and Planning for Multiagent Domains.* Technical Note 360, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.

[21] A. L. Lansky. *A Representation of Parallel Activity Based on Events, Structure, and Causality.* Technical Note 401, Artificial Intelligence Center, SRI International, Menlo Park, California, 1986. Also in "Reasoning about actions and

plans," *Proceedings of the 1986 Workshop*, Timberline, Oregon, M. P. Georgeff and A. L. Lansky (eds), Morgan Kaufmann, Los Altos, California, 123-160, 1987.

[22] A. L. Lansky and D. S. Fogelsong. Localized representation and planning methods for parallel domains. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-87*, Seattle, Washington, 1987.

[23] H. P. Morave. The stanford cart and the cmu rover. In *Proceedings of the IEEE*, pages Volume 71, pp. 872–884, 1983.

[24] N. J. Nilsson. *Shakey the Robot.* Technical Note 323, Artificial Intelligence Center, SRI International, Menlo Park, California, 1984.

[25] N. J. Nilsson. *Triangle Tables: A Proposal for a Robot Programming Language.* Technical Note 347, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.

[26] S. J. Rosenschein and L. P. Kaelbling. A formal approach to the design of intelligent embedded systems. In *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, Monterey, California, 1985.

[27] E. D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier, North Holland, New York, 1977.

[28] C. J. Stuart. *Synchronization of Multiagent Plans Using A Temporal Logic Theorem Prover.* Technical Note 350, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.

[29] S. Vere. Planning in time: windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246–267, 1983.

[30] D. E. Wilkins. Domain independent planning: representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[31] D. E. Wilkins. Recovering from execution errors in SIPE. *Computational Intelligence*, 1:33–45, 1985.