REACTIVE REASONING AND PLANNING

Michael P. Georgeff
Amy L. Lansky

Artificial Intelligence Center, SRI International
333 Ravenswood Avenue, Menlo Park, California
Center for the Study of Language and Information, Stanford University

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. 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 highly dynamic and uncertain worlds.

For our the 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 [8]. In this paper, we concentrate on the navigational task; the knowledge base used for jet malfunction handling is described elsewhere [6,7].

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 [5,12,14]. 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;

This research has been made possible by a gift from the System Development Foundation, the Office of Naval Research under Contract N00014-85-C-0251, by the National Aeronautics and Space Administration, Ames Research Center, under Contract NAS2-12521, and FMC under Contract FMC-147466.

if they do not, the system can return control to the plan constructor so that it may modify the existing plan appropriately.

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. To overcome this problem, at least in part, there has been some work on developing planning systems that interleave plan formation and execution [3,4]. 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 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.

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 [6].

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.

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.

A number of systems developed for the control of robots do have a high degree of reactivity [1]. Even SHAKEY [10] 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 [11]. Another approach is advocated by Brooks [2], 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. Kaelbling [9] 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 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) [6,7]. The system 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 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.

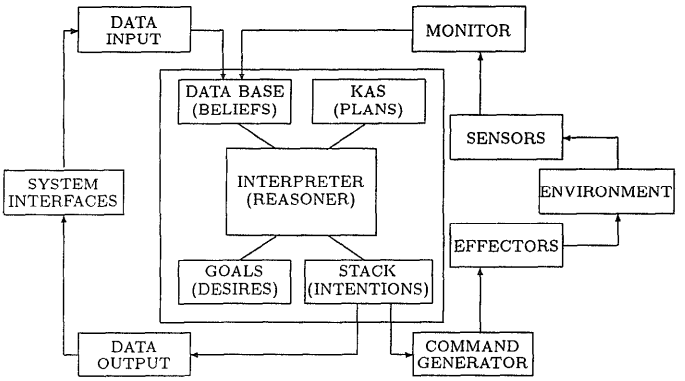

Figure 1: System Structure

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 some mechanical components must obey. 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.

The data base itself consists of a set of *state descriptions* describing what is believed to be true at the current instant of time. We use first-order predicate calculus for the state description language. Data base queries are handled using unification over the set of data base facts. 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 on some interval of time (i.e., on 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 which 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 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 `(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 using the logical operators $\wedge$ and $\vee$. These denote, respectively, the intersection and union of the composite behaviors.

As with state descriptions, behavior descriptions are not restricted to describing the external environment, but can also be used to describe 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 `(!(`*belief* 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 traditional planning systems [13]. 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.

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.

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 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 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 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 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.

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. 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.

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.

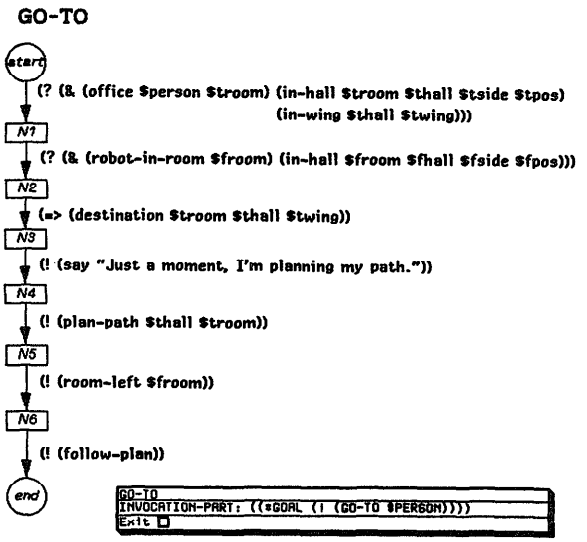

Figure 2: The Top-Level Strategy

4 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 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 (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. The topological knowledge is not detailed, stating simply 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. Throughout this predictive-planning stage, the robot remains continuously reactive. Thus, for example, should the robot notice indication of a jet failure on the space station, it may well decide to interrupt its route planning and attend instead to the task of remedying the jet problem.

Once a plan is formed by the route-planning KAs, that plan must be used to guide the activities of the robot. To achieve this,

we defined a group of KAs that react to the presence of a 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.

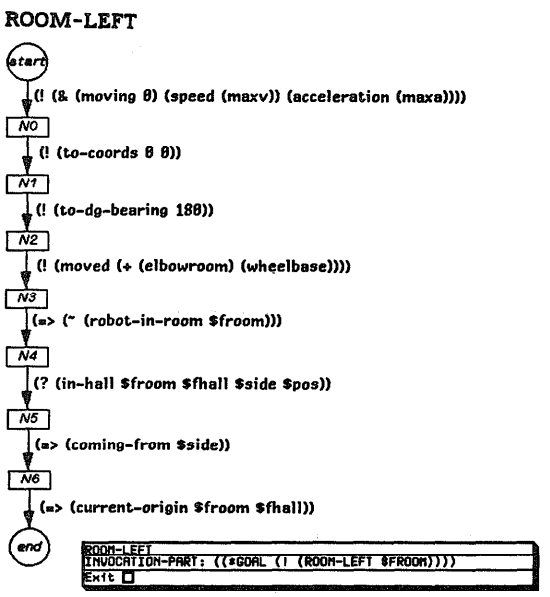

Figure 3: Route Navigation KA

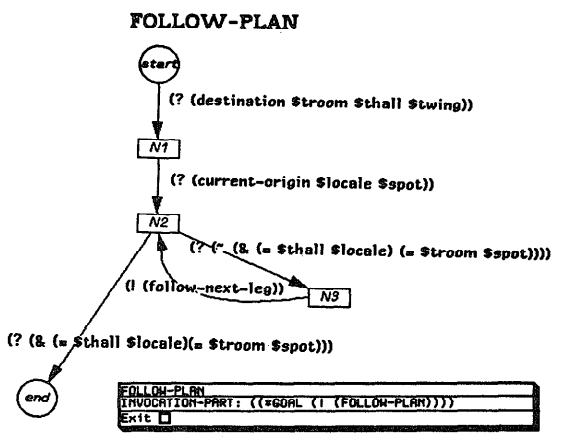

Figure 4: Plan Interpretation KA

For example, let us consider the KAs in Figures 3 and 4. 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 3 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 `(!(follow-plan))`. This activates the KA in Figure 4, which assures that each leg of the plan is followed until the goal destination is reached. Beliefs of the form `(current-origin $locale $spot)` are repeatedly updated 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. These will include KAs that respond to the presence of an obstacle ahead or the fact that an emergency light has been seen. Such 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."

Yet other KAs perform the various other tasks required of the robot [7]. 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). Each KA manifests a self-contained behavior, possibly including both sensory and effector components. Many of these KAs can be simultaneously active, performing their function whenever they may be applicable. Thus, while trying to follow a path down a hallway, an obstacle avoidance procedure may simultaneously cause the robot to veer from its original path. We elsewhere provide a more detailed description of the KAs used by the robot [8].

5 Discussion

The system as described here was implemented using the new SRI robot, Flakey, to accomplish much of the two scenarios described in the introduction. 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 the 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 actual procedures used for NASA's space shuttle [7].

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. 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, the system stood a better chance of achieving its goals under the stringent real-time constraints of the domain. For example, the method for determining the robot's course was dynamically influenced by the 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.

Because PRS expands plans dynamically and incrementally, there were also frequent opportunities for it to react to new situations and changing goals. 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. 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. 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. This has much in common with, and yields the same advantages as, the vertical robot architecture proposed by Brooks [2].

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.

Despite these encouraging results, the research is only in its initial stages and there are a number of limitations that still need to be addressed. First, there are many assumptions behind the procedures (KAs) used. For example, we have assumed that hallways are straight and corners rectangular and that all doors are open and unobstructed. A greater variety of KAs and increased parallelism would also 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. Finally, PRS does not reason about other subsystems (i.e., other PRS instantiations) in any but the simplest ways. However, the message-passing mechanisms we have employed should allow us to integrate more complex reasoning about interprocess communication.

Acknowledgments

Marcel Schoppers carried out the experiment described here. Pierre Bessiere, Joshua Singer, and Mabry Tyson helped in the development of PRS. Stan Reifel and Sandy Wells designed Flakey and its interfaces, and assisted with the implementation described herein. We have also benefited from our participation and interactions with members of CSLI's Rational Agency Group (RATAG), particularly 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] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] M. P. Georgeff, A. L. Lansky, and M. Schoppers. *Reasoning and Planning in Dynamic Domains: An Experiment with a Mobile Robot.* Technical Note 380, Artificial Intelligence Center, SRI International, Menlo Park, California, 1987.

[9] 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.

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

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

[12] 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.

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

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