Reasoning and Planning in Dynamic Domains: An Experiment With a Mobile Robot

M.P. Georgeff,* A.L. Lansky,* and M.J. Schoppers
Stanford Research Institute International
Menlo Park, CA 94025

Abstract

We describe progress made toward having an autonomous mobile robot reason and plan complex tasks in real-world environments. To cope with the dynamic and uncertain nature of the world we use a highly reactive system to which is attributed 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 system need only be partly elaborated before it decides to act. This allows the system to avoid overly strong expectations about the environment, overly constrained plans of action, and other forms of over-commitment common to previous planners. In addition, the system 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 complex *real-world domains*.

We have tested the system using SRI's autonomous robot (Flakey) in a scenario involving navigation and the performance of an emergency task in a space station scenario.

1 Introduction

One feature that is critical to the survival of all living creatures is their ability to act appropriately in dynamic environments. For lower life forms, it seems that sufficient capability is provided by stimulus-response and feedback mechanisms. Higher life forms, however, require more complex abilities, such as reasoning about the future and forming plans of action to achieve their goals. The design of reasoning and planning systems that are *situated* in environments with 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 demonstrate how the system can be applied to the control of an autonomous mobile robot. The system, called a procedural reasoning system (PRS), is endowed with the psychological attitudes of belief, desire, and intention. At any instant, the actions that the system considers performing depend not only on the current desires or goals of the system, but also on its beliefs and previously formed intentions. The system also has the ability to reason about its own internal state – that is, to reflect on its own beliefs, desires, and intentions and to modify these as it chooses. This architecture allows the system to reason about means and ends in much the same way as traditional planners, but provides the reactivity required for survival in complex real-world domains.

As the task domain, we envisage a robot in a space station acting as an astronaut's assistant. When asked to get a wrench, for example, the robot works out where the wrench is kept, plans a route to get it, and goes there. If the wrench is not where expected, the robot may reason further about how to obtain information on its whereabouts. It then finally returns to the astronaut with the wrench 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 of higher priority than retrieving the wrench and sets about diagnosing the fault and correcting it. Having done this, it continues with its original task, finally telling the astronaut what has happened.

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 world in which the robot is situated is continuously changing and because other agents and processes can issue demands at arbitrary times, performance of these tasks requires an architecture that is highly reactive as well as goal-directed.

We have used PRS with the new SRI robot, Flakey, to accomplish much of the two scenarios described above, including both the navigation and malfunction-handling tasks. In the next section, we discuss some of the problems with traditional planning systems. The architecture and operation of PRS is then described, and Flakey's primitive capabilities are delineated. We then give a more detailed analysis of the problems posed by this application and our progress to date. We concentrate on the navigation task; the knowledge base used for the jet malfunction handling is described *elsewhere* [15,17].

2 Previous approaches

Most architectures for intelligent autonomous systems consist of a plan constructor and a plan executor. Typically, the plan constructor plans an entire course of action before commencing execution of the plan [1,11,25,28,30,32,33,34]. The plan itself is usually composed of primitive actions – that is, actions

* Also affiliated with the Center for the Study of Language and Information, Stanford University, Stanford, California.

This research has been made possible in part by a gift from the System Development Foundation, the Office of Naval Research under contract no. N00014-85-C-0251 and FMC under contract no. FMC-147466.

that are directly performable by the system. The motivation for this approach, of course, is to ensure that the planned sequence of actions actually achieves the prescribed goal. As the plan is executed, the system performs the primitive actions in the plan by calling various low-level routines. Usually, execution is monitored to ensure that these routines achieve the desired effects; if they do not, the system may return control to the plan constructor to modify the existing plan appropriately. There are, however, a number of serious drawbacks with this architecture as the basis for the design of autonomous agents.

First, this kind of planning is very time consuming, requiring exponential search through potentially enormous problem spaces. It is thus usual for classical AI planners to spend considerable time thinking before performing any effector actions. 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. Furthermore, the real-time constraints of the domain often require almost immediate reaction to changed circumstances, allowing insufficient time for this kind of planning.

Second, in real-world 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 performed depends on information acquired on the way – such as which turnoff to take, which lane to get into, when to slow down and speed up, and so on. Traditional planners can only cope with this uncertainty in 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 (ILAs) used by SHAKEY [23]). The former case is combinatorially explosive or simply cannot be done – the world around us is simply too dynamic to anticipate all circumstances. The latter, as usually implemented, seriously restricts flexibility and reasoning capabilities. Of course, in situations where we can paint ourselves into a corner, some preplanning is necessary. But even this need not involve expanding plans down to the level of primitive operators; indeed, we may do the planning in quite a different abstraction space than that used to guide our actions in the real world (see, for example, the representations in the missionaries and cannibals problem discussed by Amarel [3]).

A third drawback of traditional planning systems is that they usually provide no mechanisms for reacting to new situations or goals during plan execution, let alone during plan formation. For example, many robots (e.g., SHAKEY [23]) effectively shut off their abilities to react to new situations and goals while moving from one location to another. Only low-level feedback mechanisms and emergency sensors such as collision detectors remain enabled. Such disregard for sensory input is particularly undesirable in realistic environments in which unpredictable events may occur or other agents may be active – because of innaccurate information about the actual state of the world, actions may be chosen that are inappropriate to achieving the goals of the system. By remaining continuously aware of the environment, an agent can modify its actions and goals as the situation warrants.

Indeed, the very survival of an autonomous system may depend on its ability to react quickly to new situations and to modify its goals and intentions accordingly. 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 needs 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 possibly changing beliefs and goals. While many existing planners have replanning capabilities, none have accomodated modifications to the system's underlying set of goal priorities.

Finally, current planners are overcommitted to the planning strategy itself – 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 do not have the ability to decide when to stop planning, nor to reason about the trade-offs between further planning and longer available execution time. Furthermore, they are committed to one particular planning strategy, and cannot opt for different methods in different situations. This clearly mitigates against survival in the real world.

In sum, the central problem with traditional planning systems may be viewed as one of overcommitment. These systems have strong expectations about the behavior of the environment and make strong assumptions about the future success of their own actions. They are strongly committed to their goals and intentions and, except in certain simple ways, cannot modify them as circumstances demand. This would be fine if it were possible to build plans that accommodate all the complexities to which an agent must be responsive; unfortunately, in most real-world domains, the construction of such plans is infeasible.

Of course, we are not suggesting that preplanning, followed by later replanning, can be completely avoided: because of unanticipated changes in the environment, an agent will often have to reconsider its goals or its intended means of achieving these. This is a property of the environment that an agent can do little about. If the agent did not make *some* assumptions about the behavior of the environment, there is little chance it would ever be able to act. On the other hand an agent should not make too many assumptions about the environment – to the extent possible, decisions should be deferred until they *have* to be made. The reason for deferring decisions is that an agent can only acquire *more* information as time passes; thus, the quality of its decisions can only be expected to improve. Of course, there are limitations resulting from the need to coordinate activities in advance and the difficulty of manipulating excessive amounts of information, but some degree of deferred decision-making is clearly desirable.

There has been some work on developing planning systems that interleave plan formation and execution [10,21,29]. While these systems can cope far better with uncertain worlds than traditional planners, they are still strongly committed to achieving the goals that were initially set them. They have no mechanisms for changing focus, adopting different goals, or reacting to sudden and unexpected changes in their environment. The reactive systems used in robotics also handle changes in situation better than traditional planning systems [2,7,18]. Even SHAKEY [23] utilized reactive procedures (ILAs) to realize the primitive actions of the high-level planner (STRIPS), and this idea is pursued further in some recent work by Nilsson [24]. However, there is no indication of how these systems could reason rationally about their future behaviors, such as to weigh the pros and cons of taking one course of action over another.

3 Knowledge Representation

The system we used for controlling and carrying out the high-level reasoning of the robot is called a *Procedural Reasoning System* (PRS) [15].[1] 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.[2] Later sections will give examples of PRS use in the the robot scenario.

[1] Flakey is being used in a variety of experiments at SRI, and PRS is just one of various systems being employed for controlling Flakey.

[2] A more formal description of PRS may be found in [17].

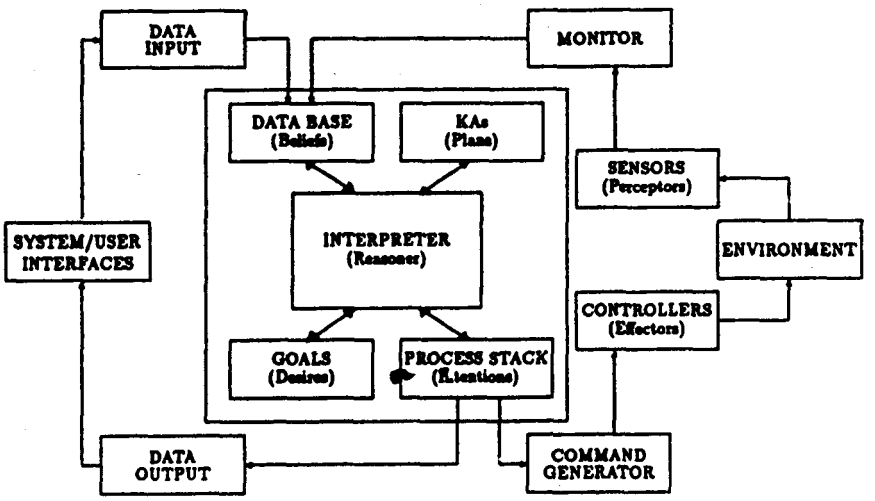

Figure 1: System Structure

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. Consequently, at some times the system may believe that it is in a particular hallway, and 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 of time. We use first-order predicate calculus for the 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. Data base queries are done using unification over the set of data base predicates.

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., 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 ∧ and ∨. 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*, describing 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 graphical 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 planning systems such as NOAH [28] and SIPE [34]. They differ in that (1) the subgoals appearing in the body can be described by complex temporal expressions (i.e., the goal expressions described in the preceding section), and (2) the allowed control constructs are much richer, and include conditionals, loops, and recursion. One important advantage of using abstract subgoals rather than 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.

The set of KAs in a PRS application system consists not only 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, a typical metalevel KA would supply a method for choosing between multiple relevant KAs, or how to achieve a conjunction of goals, or how much further planning or reasoning can be undertaken given the real-time constraints of the problem domain. These metalevel KAs may, of course, utilize domain-specific knowledge as well. 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 system interpreter runs the entire system. From a conceptual viewpoint, 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 (applicable). One of these KAs will then be chosen for execution.

In the course of traversing the body of the chosen KA, new subgoals will be posted and new beliefs will be derived. When new goals are pushed onto the goal stack, the interpreter checks to see if any new KAs are relevant, and executes them. Likewise, whenever a new belief is added to the data base, the interpreter will perform appropriate consistency-maintenance procedures and possibly activate new applicable KAs. During this process, various metalevel KAs may also be called to make choices between alternative paths of execution, to choose between multiple applicable KAs, to decompose composite goals into achievable components, and to make other decisions.

This results in an interleaving of plan selection, formation, and execution. In essence, the system forms a partial overall plan (chooses a KA), figures out near term means (tries to find out how to achieve the first subgoal), executes them, 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, specific questions about the means to achieve these ends are left open to future deliberation.

This approach has many advantages. First, systems generally lack sufficient knowledge to expand a plan of action to the lowest levels of detail – at least if the plan is expected to operate effectively in a real-world situation. The world around us is simply too dynamic to anticipate all circumstances. By finding and executing relevant procedures only when needed and only when sufficient information is available to make wise decisions, the system stands a better chance of achieving its goals under real-time constraints.

Because the system is repeatedly assessing its current set of goals, beliefs, and the applicability of KAs, the system also exhibits a very reactive form of behavior, rather than being merely goal-driven. By reactive, we mean more than a capability of modifying current plans in order to accomplish given goals; a reactive system should also be able to *completely change its focus* and pursue new goals when the situation warrants it. This is essential for domains in which emergencies can occur and is an integral component of human practical reasoning.

Because PRS expands plans dynamically and incrementally and also allows for new reactive KAs to respond when they are relevant, there are frequent opportunities for it to react to new situations and to change goals. The system is therefore able to modify its intentions rapidly on the basis of what it currently perceives as well as upon what it already believes, intends, and desires. It can even change its intentions regarding its own reasoning processes – for example, the system may decide that, given the current situation, it has no time for further reasoning and 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. PRS was therefore 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 with 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 with the new information, if anything.

Typically, this decision is made by a fact-invoked KA (in the receiving PRS), which responds upon receipt of the external message. On the basis of such factors as the reliability of the sender, the type of the 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.

We have found the ability to perform multiple activities concurrently to be crucial in the robot domain. Although some systems do generate plans, portions of which can be executed in parallel (e.g., NOAH [28] and SIPE [34]), our motivations for parallelism are quite different. In our case, the parallelism is essential for processing the constant stream of sensory information and for controlling devices continuously. That is, parallelism is required for the system's proper operation. In NOAH and SIPE, however, the parallelism is simply fortuitous and does not result from any demands on processing speed or distributed functionality.

4 Flakey the Robot

Flakey was designed and built within SRI's Artificial Intelligence Center, and is being used by several research teams to test software-organization ideas. It contains two onboard computers, a SUN II workstation (with 42Mb 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 manages 12 sonars, 16 bumper contacts, and 2 stepper motors for 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.

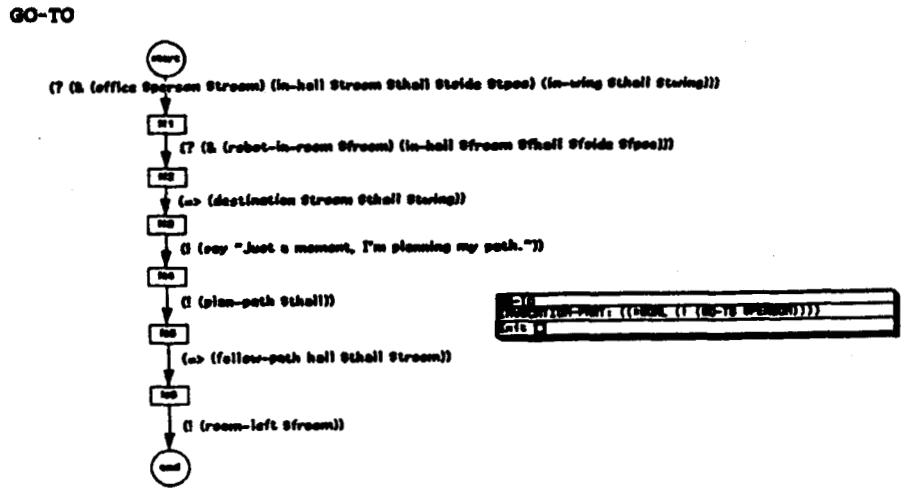

Figure 2: Top-level Strategy

The 12 sonars are located approximately 5 inches off the ground, 4 facing forward, 4 backward, and 2 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 one can specify a desired distance, a maximum forward speed, and a desired acceleration. The Z80 uses the given acceleration to achieve the maximum speed compatible with the desired distance.

Changing direction is done 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. Direction changes are much more difficult 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 using 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 wish to avoid reliance on dead reckoning, however, we did not use the position integrator for the top-level navigation task; it was used, 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 about 5 millimeters, can occasionally return invalid readings and can also fail to see an object if the angle of incidence is high enough. 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 which fluctuate up to 10 millimeters/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 keep on route, and various general tasks such as malfunction handling and requests for information. In this paper, we will concentrate on the route planning and navigation tasks. However, it is important to realize that the knowledge representation provided by PRS is used for reasoning about all tasks that the system performs.

The way the robot (that is, Flakey 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 that route to the desired location (see the KA depicted in Figure 2). In planning the route, the robot uses knowledge of the topology of the station to work out a route to the target location, as is typically done in navigation tasks for autonomous robots [6,7,22]. The topological knowledge is of a very high-level form, stating which rooms are in which corridors and how corridors are connected. A plan formed by the robot is also high-level, typically having the following kind of 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 topology of the problem domain is stored in its data base, and 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 specifying possible movements through rooms and down hallways.

A different set of KAs is used for navigating the route mapped out by the route-planning KAs (see Figures 7, 8, and 9). The navigation KAs perform such tasks as sensing the environment, determining 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 between different means to realize any given goal, and determine the priority of tasks when mutually inconsistent goals (such as diagnosing a jet failure and fetching a wrench) arise. If the robot's route plan fails, the route-planning KAs can again take over and replan a route to the target destination. In the implementation described herein, however, we have not provided any KAs for reestablishing location once the robot has left its room of departure, and so it does not currently exhibit any replanning capability.

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 is, nor the consequences of drifting from side to side — that is too low a level of detail to be considered 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 start- and end-points, they first select some intermediate point. They then repeat the process for the two resulting subpaths until all paths are 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, the top-level route planning is probably the simplest aspect of the navigation task.

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

```
(follow-path hall ej233 j1)
(follow-path hall j1 j4)
(follow-path hall j4 j2)
(follow-path hall j2 ej270)
```

Figure 3: Route from ej233 to ej270

It is important to emphasize that, even during this relatively short 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.

5.2 Reactive, Goal-Directed Behavior

The KAs used to navigate the route fall into three classes: those that interpret the path plan and establish intermediate target locations, those that are used to follow the path, and those that handle critical tasks such as obstacle avoidance and reacting to emergencies. Each KA *manifests a self-contained* behavior, possibly including both sensory and effector components. Moreover, the set of KAs is naturally partitioned according to level of functionality (*cf.* [7]): low-level functions (emergency reactions, obstacle avoidance, etc.), middle-level functions (following already established paths and trajectories), and higher-level functions (figuring out how to execute a topological route). All of these KAs are 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 slightly from its original path.

Once a plan is formed by the route-planning KAs, that plan must be converted into some useable form. Ideally, the plan shown in Figure 3 should be represented as a procedural KA containing the goals "leave room ej233 and go into hall j1," "go to the j1-j4 junction," etc. Since it is not currently possible for KAs to create or modify other KAs, we have, instead, 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 plan generates subgoals such as turning a corner, travelling the hallway, and updating the data base to indicate progress. The second group of navigation KAs reacts to these goals by actually doing the work of reading the sonars, interpreting the readings, counting doorways, aligning the robot within the hallway, and watching for obstacles ahead.

For example, consider the KAs in Figures 7 and 8. After having planned out a path as directed by the KA in Figure 2, the robot is given a goal of the form `(! (room-left $froom))` (the variable `$froom` will be bound to some particular constant representing the room that the robot is trying to leave). The KA in Figure 8 will respond and actually perform steps for leaving the given room. The last step in this KA will insert a fact into the system database of the form `(origin $froom $fhall)` (again, the variables will be bound to specific constants). This fact alerts a path interpretation KA (depicted in Figure 7) that the robot is now ready to execute a leg of its path, and supplies the KA with the robot's starting position (i.e., the room adjacent to the robot, `$froom`, and the hall in which it stands, `$fhall`). Assuming that the facts describing a path have been placed in the database (for example, the set of facts in Figure 3), the fact-invoked `FIND-NEXT` KA in Figure 7 will respond and begin to interpret the path. It will then proceed and travel down the hallway as instructed. This will in turn establish a new origin position, thereby allowing for the next step of the path to be executed.

A third group of KAs reacts to contingencies observed by the robot as it interprets and executes its path. For example, these will include KAs that respond to the presence of an obstacle ahead (see Figure 9) 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 determine whether and when preemption is desirable, but, at this stage of the project, we have not used metalevel KAs besides those provided as PRS defaults (which give immediate preemption). An alternative to preemption is to send a contingency message to another PRS instantiation that can process that message in parallel.

5.3 Parallelism and Mediation

Because of the 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 be used as information filters to protect other instantiations from a barrage of uninteresting sensory information. (The need for such filters arises in many problem domains – for example, in monitoring sensors on the space shuttle [4].) The strongest reasons, however, have to do with the inherently parallel and largely independent nature of the various computations that must be performed in dynamic environments.

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 a change in course is warranted. Because speed changes cannot be 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

PLAN-PATH

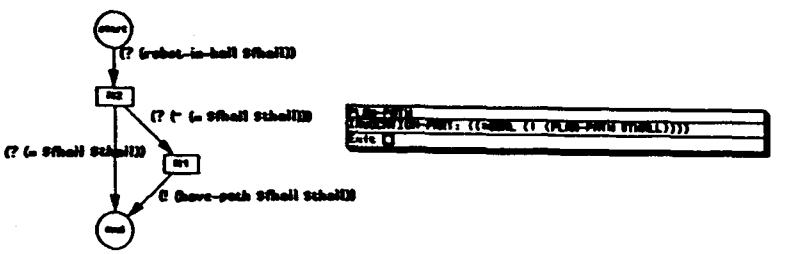

Figure 4: Path Planning KA

HAVE-PATH

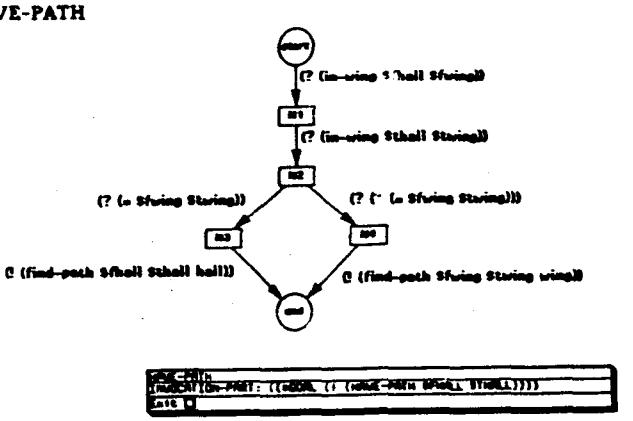

Figure 5: Path Planning KA

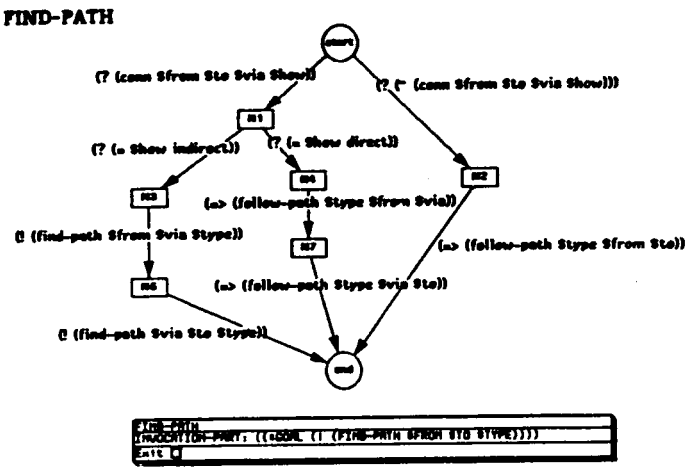

Figure 6: Path Planning KA

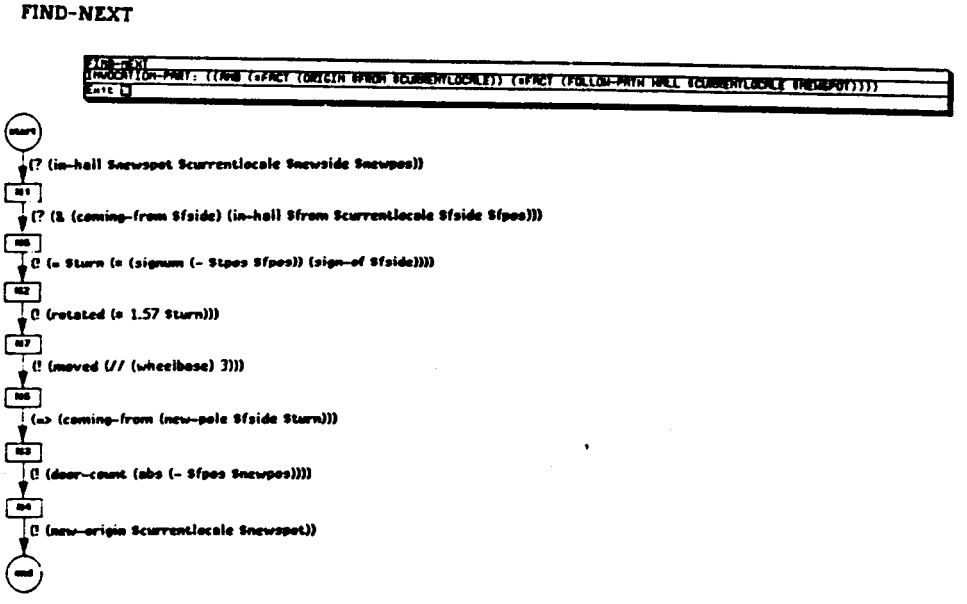

Figure 7: Plan Interpretation KA

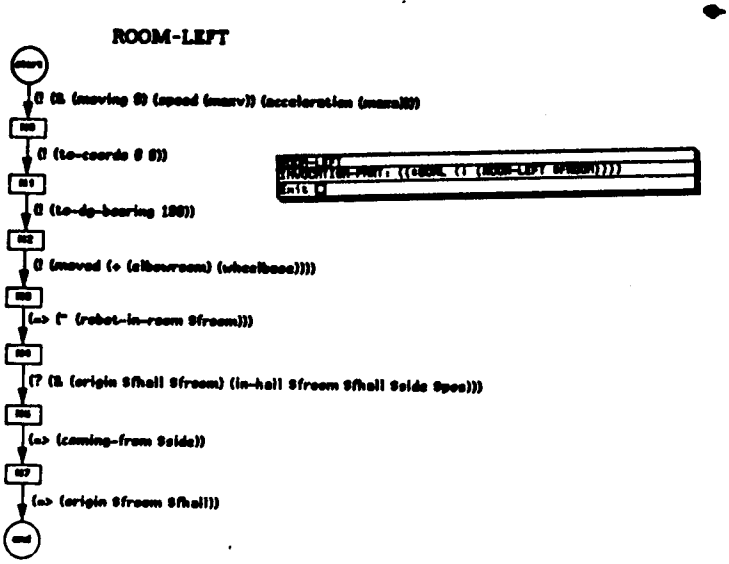

Figure 8: Route Navigation KA

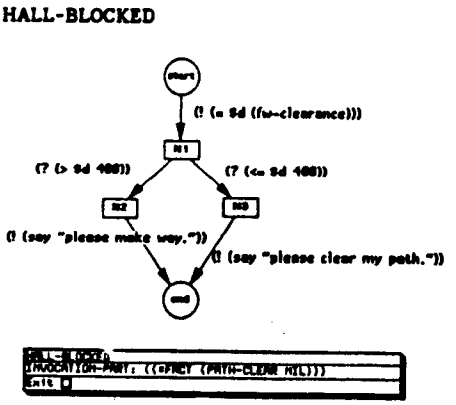

Figure 9: Reactive KA

course change, the robot might completely miss a door reading. Conversely, delaying course changes for the sake of sonar monitoring could make a collision with a wall inevitable. Of course, travelling at lower speeds would solve the problem, but would also render the robot too slow to be useful.

The most effective way to handle this problem is to allow multiple PRS instantiations to execute concurrently. Running several instantiations asynchronously has 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, with another driving the robot to the target location and adjusting speed appropriately (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, and changes in speed by changing the accelerations 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 problem 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 achieved independently, as the means for accomplishing these goals interact with one another.

To mediate between interacting goals, we chose to implement a third PRS capable of accepting both speed and direction change requests asynchronously. This PRS could be viewed as a virtual controller. Because the virtual controller is in complete control of the wheels, it can issue instructions that achieve both kinds of requests at once. 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 left and right wheel speeds.

Related to the problem of interacting goals is that of goal conflict: just as one may have possibly conflicting beliefs about a situation that need to be resolved (the problem of situation assessment), one may also have conflicting goals (or desires) that need mediation [18]. For example, the virtual controller discussed above often gets conflicting speed requests from KAs: the hallway traversal KA might request that a certain velocity be maintained, the KA that detects approach of the target location 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 resolve these conflicting goals, the virtual controller has to be able to reason about their urgency and criticality. This, in turn, may involve further communication with the systems requesting these goals. Our present solution is to define domain-dependent mediators where necessary, but, at present, no general approach to this problem has been attempted.

5.4 Coping with Reality

Our initial implementation of the robot application used multiple PRS instances interacting with a robot simulator, all running on the Symbolics 3600. This worked well, and demonstrated the suitability of the system for controlling complex autonomous devices. That done, we began work on driving the real robot. This transfer 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 was confined to occur via an ethernet cable. Software for remote procedure calls over the ethernet limited communication to 15 function calls per second – too slow for timely response to sensor input. Consequently, we were forced to transfer much of the functionality of PRS to Flakey's SUN. This required translating the functionality of the lower-level KAs into C code, as well as explicit coding for message and clock-signal handling. Unfortunately Flakey's operating system also did not support interprocess communication at the bandwidth and efficiency we wanted. This forced us to implement communication through shared memory, with all the concommitant synchronization code needed. 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 solely constructed from instances of PRS. As a result, our final implementation is considerably more constrained than the simulation version in its ability to reason about its low-level actions and to react appropriately to changing goals.

The second obstacle to translating from our simulated application to the 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 foster efficient debugging. It is hard to repeat experiments (and get the same bugs), time delays become critical, and the behaviors of real sensors and effectors can differ significantly from simulated ones.

The configuration of our current application system is shown is Figure 10. Three machines are involved, a Symbolics 3600, a SUN, and a Z80, running six application processes. The wheels and sonars are also depicted, and may be regarded as physical processes. The rectangular box represents the SUN's shared memory area; arrows represent interprocess communication.

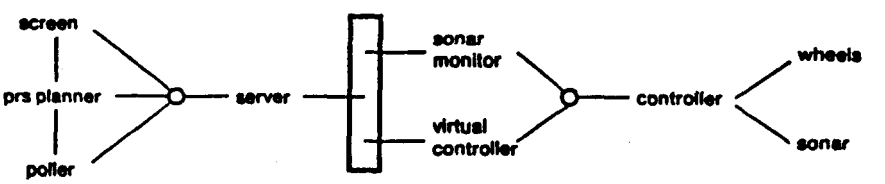

Figure 10: Processes Used in the Implementation

6 Discussion

The primary purpose of this research was to show that the BDI architecture of PRS, the partial hierarchical planning strategy it supports, and its metalevel (reflective) capabilities could be effective in real-world dynamic domains. Furthermore, the design of PRS meets some of the more important desiderata for autonomous systems: modularity, awareness, and robustness [18]. In this section, we will briefly compare our approach to other work in the areas of planning and robotics.

The partial hierarchical planning strategy and the reflective reasoning capabilities used by PRS allow many of the difficulties associated with traditional planning systems to be avoided, without denying the ability to plan ahead when necessary. By finding and executing relevant procedures only when sufficient information is available to make wise decisions, the system stands a better chance of achieving its goals under real-time constraints.

For example, the speed and direction of the robot is determined during plan execution, and depends on such things as proximity of obstacles and the actual course of the robot. Even the method for determining this course depends dynamically on the situation, such as whether the robot is 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 depends on observations made during the diagnostic process.

Because PRS expands plans dynamically and incrementally, there are also frequent opportunities to react to new situations and changing goals. The system is therefore able to modify its intentions (plans of action) rapidly on the basis of what it currently perceives as well as upon what it already believes, intends, and desires. For example, when the system notices a jet-fail alarm while it is attempting to fetch a wrench, it has the ability to reason about the priorities of these tasks, and, if so decided, suspend the wrench-fetching task while it attends to the jet failure. Indeed, the system even continues to monitor the world while it is route planning (in contrast to most robot systems), and this activity too can be interrupted if the situation so demands.

The powerful control constructs used in PRS procedure bodies (such as conditionals, loops, and recursion) are also advantageous. As a result, the robot can display behaviors of the form "do X until B becomes true." When X is "maintain speed at 400mm/sec" and B is "N doorways have been observed" we see why we could dispense with coordinate grids and dead reckoning: we could *define* the robot's behaviors in terms of conditions that changed over time. In contrast, classical planning systems often have difficulty in reasoning about such behavior and are thus restricted to using unchanging features such as fixed locations or distances.

PRS is also very robust in that there may be many different KAs available for achieving some given goal. Each may vary in its ability to accomplish the goal, and in its applicability and appropriateness in particular situations. Thus, if there is insufficient information about the current situation to allow one KA to be used, some other - perhaps less reliable - KA may be available instead. For example, if a topological map of an area is unavailable for planning purposes, the robot need not be rendered ineffective - there may, for example, be some other KA that sets the robot off in the general direction of the target. Parallelism and reactivity also help in providing robustness. For example, if one PRS instantiation is busy planning a route, lower-level instantiations can remain active, monitoring changes to the environment, keeping the robot in a stable configuration, and avoiding dangers.

The system we propose also meets many of the criteria of rational agency advanced in the philosophical literature on practical reasoning (e.g., see the work of Bratman [5]). Driven by the demands of explaining resource-boundedness and inter- and intra-agent coordination, recent work in the philosophy of action has moved beyond belief-desire architectures for rational agents and has provided insights into the nature of plans and intentions, and especially the nature of intention formation.

In particular, plans are viewed as being subject to two kinds of constraints: *consistency constraints* and requirements of *means-ends coherence*. That is, an agent's plans need to be both internally consistent and consistent with its beliefs and goals. It should be possible for an agent's plans to be successfully executed (that is, to achieve the more important goals of the system) in a world in which its beliefs are true. Secondly, plans, though partial, need to be filled in to a certain extent as time goes by, with sub-plans concerning means, preliminary steps, and relatively specific courses of action. These subplans must be at least as extensive as the agent believes is required to successfully execute the plan; otherwise they will suffer for means-ends incoherence.

These constraints on the beliefs, desires (goals), and intentions of an agent are realized by the system proposed herein, and as such it can be viewed as an implementation of a rational agent. In addition, the notion of intention we use meets the major requirements put forward by Bratman [5], 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, our system is far from manifesting the behavioral complexity of real rational agents; however, it is a step in the direction of a better understanding of rational action.

In contrast to most AI planning work, research in robotics *has* been very concerned with reactivity and feedback [2,18,23]. However, most of this work has not been concerned with general problem solving and commonsense reasoning - the work is almost exclusively devoted to problems of navigation and execution of low-level actions. Furthermore, the reactivity is not of the general form we advocated above; although the systems can adjust the *means* for achieving given goals depending on incoming sensory information, they do not exhibit the ability to completely change goal priorities, to modify, defer, or abandon current plans, or to reason about what is best to do in light of the current situation.

Recently, Brooks [7] has advanced some intriguing ideas concerning the structure of autonomous systems. Rather than the horizontal structure typical of most robot systems (where lower levels are restricted to performing basic sensory and effector processing, and the higher levels to planning and reasoning) Brooks advocates a vertical decomposition in which distinct *behaviors* of the robot are separately realized, each making use of the robot's sensory, effector, and reasoning capabilities as needed.

Similarly, PRS provides a vertical, rather than horizontal, decomposition of the robot task domain. Each KA defines a particular behavior of the system, and can involve both processing of sensory information and the execution of effector actions. For example, there is a KA that manifests a behavior to remain clear of obstacles, another KA whose behavior corresponds to keeping a straight course in a corridor, and yet another that chooses and traverses routes from one room to another. All these KAs use both sensors and effectors to greater or lesser degrees - there is no single subsystem that preprocesses the sensory data before sending it to the reasoning system, and there is no post-processing of plan information that determines actual effector actions.

In this sense, our system is very similar in structure to that proposed by Brooks - indeed, it can claim the same positive benefits [8]:

- 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 the behaviors are [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 our system and that advanced by Brooks is that we employ a much more general mechanism for selecting between 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 need to be compiled into a form closer to that envisaged by Brooks. Similarly, while our system naturally maps onto *coarse-grain parallel machines*, sophisticated compilation techniques would be required to map the lower-level functions onto highly parallel architectures.

Currently, 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, such as described by Cohen and Levesque [9]. Reasoning about process interference and synchronization is also important where concurrency is involved. The mechanisms developed by us for reasoning about these problems [12,13,14,19,20,31] could also potentially be integrated within PRS. Our future research plans include both work on communication and synchronization within the PRS framework.

Finally, in giving a description of the PRS architecture, it is important to note that the actual *implementation* of PRS is not of 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 demanding that such systems actually be structured with distinct data structures that explicitly represent these psychological attitudes (although, indeed, that is the way we have chosen to implement our system). 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 desired behavior of the robot. This description, suitably formalised,[3] could be realized in (or compiled into) any suitable implementation we choose. In 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 being carried out currently by Rosenschein and Kaelbling [26].

7 Limitations

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 that still need to be addressed.

First, there is a class of facts our current system must be told; for instance, the robot's starting location. If the robot is initialised in some unknown position on the topological map, the planner will abort. It would be straightforward to solve these problems by including KAs that ask for the missing information, but a completely autonomous recovery would be a much more challenging problem. Possible approaches might involve exploration of the terrain (including movement around the neighboring area) and pattern matching onto known topological 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 are the same width and have that 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 is there a garbage can in front of a cupboard); that all doors are open and unobstructed; and that other objects move much slower 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 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 while accepting the fact that this ideal will rarely be achieved. 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 axioms relating robot speed and perceptive capabilities be provided to the system. However, it is not obvious how to explicitly represent the second kind of goal, in which the system attempts to maintain a particular condition but expects at best only to approximate it.

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

8 Acknowledgments

Joshua Singer and Mabry Tyson debugged early PRS code 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 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 and Stan Rosenschein also provided helpful advice and interesting comments.

References

[1] J. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, 26:832-843, 1982.

[2] J. Albus, *Brains, behavior, and robotics*, Byte Books, 1985.

[3] S. Amarel, "On Representations of Problems of Reasoning About Actions," in *Machine Intelligence 3*, ed. D. Michie, Edinburgh University Press, Edinburgh, 1968.

[4] G. Boy, "HORSES: A Human-Orbital Refueling System Expert System," Report 2, NASA Ames Research Center, Aero-Space Human Factors Research Division, Moffett Field, California, 1985.

[3] While the semantics of the KAs is formally defined [16,17], 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 desired behavior of the robot.

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

[6] R.A. Brooks, "Visual Map Making for a Mobile Robot," *Proceedings of IEEE Conference on Robotics and Automation*, RA-1, March 1985, pp.31-41.

[7] R. Brooks, "A Robust Layered Control System for a Mobile Robot," AI Memo 864, AI Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1985.

[8] R. Brooks, "Achieving Artificial Intelligence Through Building Robots," AI Memo 899, AI Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1986.

[9] P. R. Cohen and H.J. Levesque, "Speech Acts and Rationality," *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, Illinois, pp.49-60, 1985.

[10] P.R. Davis and R.T. Chien, "Using and Reusing Partial Plans," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, pp. 494, 1977.

[11] 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.

[12] M.P. Georgeff, "Communication and Interaction in Multi-Agent Planning," *Proceedings of AAAI-83, National Conference on Artificial Intelligence*, Washington, D.C., 1983.

[13] M. P. Georgeff, "A Theory of Action for Multiagent Planning," in *Proceedings of the Fourth National Conference on Artificial Intelligence*, Austin, Texas, 1984.

[14] M.P. Georgeff, "A Theory of Process," *Proceedings of the Workshop on Distributed Artificial Intelligence*, Sea Ranch, California (1985)

[15] 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, 1985.

[16] M. P. Georgeff, A. L. Lansky, and P. Bessiere, "Procedural Logic," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, 1985.

[17] M. P. Georgeff and A. L. Lansky, "Procedural Knowledge," to appear in *Proceedings of IEEE, Special Issue on Knowledge Representation*, October 1986.

[18] L. P. Kaelbling, "An Architecture for Intelligent Reactive Systems," 1986 Workshop on Planning and Reasoning about Action, Timberline, Oregon, July 1986.

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

[20] A. L. Lansky, "A Representation of Parallel Activity Based on Events, Structure, and Causality," in *Reasoning about Actions and Plans*, Proceedings of the 1986 Workshop on Planning and Reasoning about Action, Timberline, Oregon, Morgan Kaufmann Publishers, 1987.

[21] D. McDermott, "Flexibility and Efficiency in a Computer Program for Designing Circuits," Technical Report AI-TR-402, MIT AI LAb, Cambridge, Massachusetts, 1977.

[22] H P. Moravec, "The Stanford Cart and the CMU Rover," *Proceedings of the IEEE*, Vol 71, pp. 872-884, 1983.

[23] N. J. Nilsson, "Flakey the Robot," Technical Note 323, Artificial Intelligence Center, SRI International, Menlo Park, California, 1984.

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

[25] S. J. Rosenschein, "Plan Synthesis: A Logical Perspective," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 331-337, Vancouver, British Columbia, 1981.

[26] S. J. Rosenschein and L. P. Kaelbling, "The Synthesis of Digital Machines With Provable Epistemic Properties." *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, Asilomar, California, pp. 83-98.

[27] E.D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, Vol 5, pp. 115-135, 1974.

[28] E. D. Sacerdoti, *A Structure for Plans And Behavior*, Elsevier, North Holland, New York, 1977.

[29] S. M. Schieber, "Solving Problems in an Uncertain World," Undergraduate thesis, Applied Mathematics Department, Harvard College, Cambridge, Massachusetts, 1981.

[30] M. Stefik, "Planning With Constraints," *Artificial Intelligence*, 16:111-140, 1981.

[31] 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.

[32] A. Tate, "Goal Structure — Capturing the Intent of Plans," *Proceedings of the Sixth European Conference on Artificial Intelligence*, pp. 273-276, 1984.

[33] S. Vere, "Planning in Time: Windows and Durations for Activities and Goals," Technical Report, Jet Propulsion Laboratory, Pasadena, California. (Editor's Note: AIAA Paper 83A43951)

[34] D. E. Wilkins, "Domain Independent Planning: Representation and Plan Generation," *Artificial Intelligence*, 22:269-301, 1984.