A Procedural Knowledge Approach to Task-level Control

Karen L. Myers
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
myers@ai.sri.com

Abstract

Effective task-level control is critical for robots that are to engage in purposeful activity in real-world environments. This paper describes PRS-Lite, a task-level controller grounded in a procedural knowledge approach to action description. The controller embodies much of the philosophy that underlies the Procedural Reasoning System (PRS) but in a minimalist fashion. Several features of PRS-Lite distinguish it from its predecessor, including a richer goal semantics and a generalized control regime. Both of these features are critical for supporting the management of continuous processes employed in current-generation robots. PRS-Lite has been used extensively as a task-level controller for a robot whose underlying behaviors are implemented as fuzzy rules. Tasks to which it has been applied include vision-based tracking, autonomous exploration, and complex delivery scenarios.

Introduction

In recent years, there has been a convergence of design methodologies for certain aspects of robot control. Most architectures employ a suite of continuous processes (often called *behaviors*) that provide low-level, situated control for the physical actions effected by the system. Above that, there is a recognized need for *task-level* control: marshaling the lower-level processes in the pursuit of specific objectives. In addition, many argue the need for a higher-level deliberative layer to provide look-ahead planning capabilities.

Our interests lie with task-level control. An adequate task-level controller for a robot must satisfy certain minimum requirements. The controller must support both discrete actions (such as performing an utterance) and continuous processes (such as following a wall). The controller must support concurrent actions, since a robot must generally perform many activities in parallel. Goal-driven and event-driven operation should be seamlessly integrated to enable a mixture of taskability and reactiveness. Timely response to changes in the world is critical for both task execution and survival. Finally, the structure of the control framework should enable understanding by external observers of the intent that underlies the robot's actions.

This paper describes PRS-Lite, a task-level controller for a mobile robot that embodies a *procedural knowledge* approach to action description (Georgeff & Lansky 1986). PRS-Lite satisfies the requirements for task-level control while providing a natural, expressive framework in which to specify purposeful activity. PRS-Lite has been used extensively for a range of complex tasks on a mobile robot platform, thus validating its fundamental design.

PRS-Lite is derived loosely from the family of Procedural Reasoning Systems (Georgeff & Ingrand 1989; Myers 1993), which we refer to collectively as PRS-Classic. PRS-Classic satisfies many (but not all) of the requisite properties for a task controller, thus justifying its choice as a design basis from which to begin. Indeed, PRS-Classic was used previously as a task controller for a mobile robot (Georgeff & Lansky 1987). However, that effort assumed highly restrictive conditions, including a benign operating environment and complete, accurate world maps.

PRS-Lite and PRS-Classic are similar in spirit. Both manage the invocation and execution of procedural representations of the knowledge required to achieve individual goals. Both provide the smooth integration of goal-driven and event-driven activity, while remaining responsive to unexpected changes in the world. However, the embodiment of the procedural knowledge philosophy in the two systems is markedly different. PRS-Classic is a large, general-purpose, mature system that was designed for use in a broad range of control applications. It provides many sophisticated services, including a multiagent architecture, multitasking, metalevel reasoning capabilities, and rich interactive control via graphical interfaces. PRS-Lite is a minimalist redesign that omits certain of these features for the

benefit of compactness and efficiency. For example, while metalevel reasoning can be valuable in certain situations, its support incurs a heavy cost of deliberation. A key objective in designing PRS-Lite was to retain the mixture of goal-directed and reactive activity, but in a more streamlined setting. Computational efficiency was an overriding concern, being critical for timely response to unpredictable runtime events. In particular, the robot for which PRS-Lite was designed requires a sense-act cycle of 100 ms, which must encompass all effector, perception, interpretation, and control actions.

PRS-Lite is not simply a subset of PRS-Classic.[1] Indeed, certain of the requirements for robot control are absent from PRS-Classic. One problem is PRS-Classic's assumption of atomicity for its primitive actions, making it unsuitable for the control of continuous processes. A related problem is its goal semantics: goals either succeed or fail, with their outcome affecting the overall flow of control in the system. As has been noted (Firby 1994; Gat 1992), this semantics is inappropriate for managing continuous processes. PRS-Lite employs an alternative goal semantics that supports both atomic actions and continuous processes, as well as a control regime divorced from any notion of goal success or failure.

Numerous proposals for the design of task-level controllers have been tendered in recent years, including (Firby 1994; Gat 1992; Nilsson 1994; Payton 1990; Simmons 1994). PRS-Lite has several characteristics that distinguish it from these efforts. One is that PRS-Lite was designed to control behaviors implemented using the paradigm of fuzzy logic. Such behaviors present unique problems and opportunities for a task-level controller. A second characteristic is the underlying design philosophy: while most robot controllers have been developed bottom-up in a problem-driven manner, PRS-Lite was derived from a mature, sophisticated reactive control system. In particular, PRS-Classic has been used extensively in a range of demanding applications requiring the integration of reactive and goal-oriented activity, thus boding well for the general applicability of the derivative PRS-Lite system.[2] In contrast, most recent task controllers were developed and tested exclusively in the context of robot taskability.

[1]The representations used by the two systems are not compatible at present (*i.e.*, neither can run with the other's procedure definitions).

[2]Applications of PRS-Classic include real-time tracking (Garvey & Myers 1993), a monitoring and control system for the Reaction Control System of the NASA Space Shuttle (Georgeff & Ingrand 1988), a controller for naval battle management (Ingrand, Goldberg, & Lee 1989), and crisis action planning (Wilkins *et al.* 1995).

Task-level Control in the Saphira Architecture

PRS-Lite is the task-level controller for Flakey, SRI's custom-built mobile robot. Flakey's current perception capabilities include a ring of 12 sonar sensors along its base and a stereo camera pair mounted on a pan/tilt head. Flakey is also equipped with a speaker-independent speech recognition system (Nuance) and a speech synthesis program; together these two systems enable humans to interact with Flakey in a relatively natural manner.

We briefly describe those aspects of Flakey's overall software architecture, called Saphira (Konolige *et al.* 1996), that impact the design and behavior of PRS-Lite. Saphira employs a layered control model, consisting of an *effector level* focused on basic physical actions, a *behavior level*, and a *task level*. On the representational side, Flakey applies perception and interpretation methods to construct and maintain an internal model of its environment, called the Local Perceptual Space (LPS). The LPS combines low-level occupancy grid information (Moravec & Elfes 1985), mid-level features (such as surfaces), and fully interpreted high-level structures such as corridors and doorways. The LPS can also store 'imaginary' features (called *artifacts*) used for control purposes, such as points in space to which the robot should navigate. The behavior and task controllers make extensive use of the LPS to ground their actions in the real world.

Behaviors provide the lowest level of managed control in the system. Examples include behaviors to follow a wall, to avoid collisions, and to track a person from visual input. Behaviors are implemented as sets of fuzzy rules whose antecedents consist of fuzzy formulas constructed from fuzzy predicates and logical connectives, and whose consequents are fuzzy sets of control values related to effector-level concepts such as velocity or orientation (Saffiotti, Ruspini, & Konolige 1993). This approach provides the means to condition the strength of a response on the strength of the underlying stimulus. Since different rules may recommend different levels of response, a 'defuzzification' method is used to adjudicate the control values across rules, yielding an integrated control value. This integration enables graceful blending of activities and smooth transitions between different sets of behaviors. At any point in time, several behaviors are generally *enabled*, with their level of activation determined by the fuzzy controller.

Task-level control involves coordinating the various modules of the system (behaviors, speech recognition, speech generation, and visual processing) in the pursuit of specific high-level objectives, using the LPS as

a shared repository of information about the environment. The activities of the speech and visual processing modules can be modeled as discrete actions. Thus, the focus of task management is to integrate these actions with the continuous processes that implement behaviors. PRS-Lite fills this role in the Saphira architecture.

PRS-Lite

The representational basis of PRS-Lite is the *activity schema*, a parameterized specification of procedural knowledge for attaining a declared objective. This procedural knowledge is represented as an ordered list of *goal-sets*, whose successive satisfaction will yield the overall objective of the schema. A goal-set is composed of one or more *goal statements* (or *goals*), each consisting of a goal operator applied to a list of arguments. A goal-set can be optionally identified by a label unique to its schema. Intuitively, a goal-set corresponds to an ordered sequence of goals that are to be achieved as an atomic unit. A compiler transforms the schema specifications into parameterized finite-state machines (performing optimizations where appropriate), whose arcs are labeled with individual goal-sets to be achieved. Activity schemas are launched by instantiating their parameters and *intending* them into the system. Such instantiated schemas are referred to as *intentions*. Multiple intentions can be active simultaneously, providing a multitasking capability.

Different modalities of goals are supported, as summarized in Figure 1. Broadly speaking, goal types can be categorized as *action* or *sequencing*. Action goals ground out in either executable functions (called *primitive actions*), tests on the state of the environment (as represented in the internal world model), or the activation/deactivation of intentions and behaviors. This last ability enables the hierarchical decomposition of goals. Sequencing goals provide conditional goal execution and sophisticated goal ordering beyond the default of linear processing, as well as various forms of parallelism.

Overall, PRS-Lite can be used to generate and manage a forest of directed graphs whose nodes each represent a goal-set from some activity schema. The root node of each graph represents a top-level goal, with its successors generated by hierarchical goal refinement. We refer to the set of active graphs at a given point in time as the current *intention structures* for the system. The leaf nodes of the intention structures are called the *current nodes*, and their associated goal-sets the *current goal-sets*. Note that an intention structure can have multiple current nodes because of the inclusion of parallel sequencing goals in the schema definition language.

Action Goals:

Test	check a condition
Execute	execute a primitive action
=	assignment of local variables
Wait-for	wait for a condition
Intend	dispatch intentions (block/nonblock)
Unintend	terminate intentions

Sequencing Goals:

If	conditional goals
And	parallel goals (with join)
Split	parallel goals (without join)
Goto	branching to labeled goals

Figure 1: PRS-Lite Goal Modalities

An *executor* manages intentions at runtime. It repeatedly operates a short processing cycle in which it considers the current goal-sets in each intention structure, performs any actions required to achieve their constituent goals, and updates the set of current nodes (as appropriate). The decision to limit processing to a single goal-set for each leaf node in an intention structure ensures overall responsiveness to new events. Given the granularity of processing, responsiveness is dependent on the number of active intentions, the degree of parallelism in those intentions, the size of goal-sets, and the underlying primitive actions that are executed. The design has proven adequate for the tasks considered to date, as discussed in greater detail later in the paper.

Goal Modalities

Action goals supply the most basic operations in the system. **Test** goals provide the ability to test beliefs about the current state of the external world. Within Saphira, beliefs are characterized by a combination of the Local Perceptual Space and a set of environment variables managed by PRS-Lite. **Execute** goals provide for the execution of primitive actions, which may perform internal bookkeeping, the setting of environment variables, or the triggering of specific external actions by the system. External actions for Flakey include the generation of an utterance by the speech synthesis module, and the invocation of a route planner. = goals enable the binding of local variables within an intention.

Intend goals lead to the activation of both intentions and behaviors. As such, they enable the hierarchical expansion of intentions through repeated goal refinement. **Unintend** goals provide the complementary ability to explicitly terminate active intentions before

they run their full course. This ability is critical when operating in dynamic, unpredictable domains, where rapid switching among activities is essential. Intentions can be assigned priorities that determine the order in which they are processed by the executor. Intentions can also be named when activated, allowing them to be referenced by other intentions (in particular, **Unintend** goals).

A critical feature of **Intend** is that it supports the invocation of intentions in either *nonblocking* or *blocking* mode. In nonblocking mode, the intention is activated and control proceeds to the next goal. In essence, the nonblocking intention is spawned as an independent process within the context of the parent intention; the nonblocking intention will persist either until it completes or its parent intention terminates. In contrast, blocking mode disables updating of the current goal within the parent intention until the child completes. Any intentions activated earlier by the parent intention will continue to be processed. Degrees of blocking are also supported: an intention can be blocked until a designated success criteria is satisfied. This capability is valuable for controlling behaviors implemented as fuzzy rules, which provide a natural metric for defining degrees of success (namely, the fuzzy predicates that model the world state). As a simple illustration, Flakey has a `face-direction` behavior that orients the robot toward a designated heading. This behavior is invoked with different thresholds of blocking in different contexts, depending on how critical it is to be precisely oriented to that heading.

Wait-for goals enable limited suspension of an intention until a certain condition or event occurs. The **Wait-for** goal modality is critical in the framework in that it enables synchronization among concurrent intentions through the use of shared variables. An illustration is provided in the next section.

The sequencing goals enable more sophisticated goal-ordering and selection mechanisms than does the default of linear processing of goal-sets. Sequencing goals can be nested to arbitrary depths, yielding a rich framework for specifying control strategies. **If** goals support conditional activation of a goal. **Goto** goals support non-linear flow of control within an activity schema, by allowing a current goal-set to be mapped to any other labeled goal-set in the schema. Iteration can be specified through appropriate combinations of **If** and **Goto** goals. Two forms of parallelism are provided via the **Split** and **And** goal modalities. **Split** parallelism spawns sets of independent concurrent goals, with control in the parent intention proceeding to the successor goal-set. Each thread of activity for the spawned goals continues until it completes, or the parent intention terminates (similar in spirit to the nonblocking mode of intending). In contrast, **And** parallelism treats the parallel goals as a unit; processing of the parent intention suspends until each of the threads occasioned by the **And** subgoals terminates.

Methodology and Use

Goal-directed behavior is produced by intending schemas for satisfying individual tasks. Reactive, event-directed behavior is produced by launching intentions that employ **Wait-for** goals to suspend until some condition or event transpires.

A common idiom for the design of activity schemas is to define an umbrella intention for a specific objective, which in turn invokes both a lower-level intention for achieving the objective, and one or more 'monitor' intentions (thus combining goal- and event-driven activities). Monitors use **Wait-for** goals to detect changes in the world that could influence the actions required to achieve the overall objective of the top-level schema. Certain monitors identify failure conditions that would invalidate the approach being taken for the current task. Others provide reactivity to unexpected events that require immediate attention. Monitors can also check for serendipitous effects that eliminate the need for certain goals in active intentions, and modify the intention structures accordingly.

To illustrate the use of the various goal modalities and idioms, Figure 2 presents simplified versions of activity schemas used by Flakey to perform basic navigation tasks. The schema `:plan-and-execute` encodes a procedure for generating and robustly executing a plan to navigate to a designated destination. The destination is specified as a parameter to the schema, and is represented as an artifact in the LPS, thus linking abstract notions of place with the robot's beliefs about its environment.

The initial goal-set in the schema (with the label `:plan`) employs an **And** goal applied to three subgoals to perform certain initializations. The first **Execute** subgoal invokes a function `say` that performs a speech generation command. The second **Execute** goal initializes the environment variable `*failed-execution*`, which is used to encode information about the status of plan execution. This variable is an example of state information within PRS-Lite that provides coordination among intentions (as described further below). The final subgoal invokes a function `find-path` that produces a topological plan for navigating from the current locale to the destination. Navigation within Saphira is at the level of regions (doors, junctions, hallways); the route planner produces a sequence of such regions that should be

```
(defintention :plan-and-execute
  :params (dest)
  :goals
  `((:plan
     (AND
      (EXECUTE
        (say "Planning path to ~a" dest))
      (EXECUTE (setq *failed-execution* nil))
      (= plan (find-path *cur-region* dest)))
     )
    (IF (null plan) (GOTO :finale))
    (INTEND :monitor-planex () :blocking nil)
    (INTEND :follow-path ((path . plan))
    :blocking t :name follow-it)
    (IF *failed-execution* (GOTO :plan))
    (:finale
     (IF (null plan)
       (EXECUTE (say "No passable routes")))
     )  ))

(defintention :monitor-planex
  :params ()
  :goals
  `((:monitor (WAIT-FOR *failed-execution*))
    (:cleanup (UNINTEND 'follow-it)) ))
```

Figure 2: Activity Schemas for Directed Navigation

traversed to reach the target destination.

After performing the necessary initializations, the schema intends a nonblocking monitor intention `:monitor-planex`, followed by a blocking intention `:follow-path`, in the spirit of the umbrella idiom described above. This latter intention (not shown here) cycles through the computed path, launching various lower-level intentions as required to navigate between successive regions in the generated path. The lower-level intentions may encounter difficulties, which they signal by setting the environment variable `*failed-execution*`. The **Wait-for** goal in the `:monitor-planex` intention would detect such an event, and then process the goal **(Unintend** `'follow-it`**)**. Satisfying this goal would deactivate the `:follow-path` intention, with the monitor intention then terminating. If no lower-level intention signals a failure, the blocking intention `:follow-path` will eventually complete, enabling processing of the remainder of the `:plan-and-execute` intention. The `:monitor-planex` intention is terminated automatically when its parent intention `:plan-and-execute` terminates.

Figure 3 displays a snapshot of the intention structures at a point during a run in which Flakey uses the above schemas to execute a delivery task.[3] Each line in the display consists of: an initial marker, indicating whether the intention is blocking (∗) or nonblocking (∘), the name of the activity schema (*e.g.*, `Deliver-Object`), a unique identifier for the particular instantiation of the schema (*e.g.*, a label such as `Follow-It` if one was specified in the **Intend** goal, else an assigned name such as `I3674`), and either the next state of execution (for an intention) or `B` (for a behavior). At the instant captured by this display, PRS-Lite has two intentions active at the highest-level (corresponding to two distinct user-specified objectives): `Deliver-Object` and `Avoid`. The `Avoid` intention has only one active thread at this point, namely the behavior for avoiding collisions (`Avoid-Collision`). Note though that in the past or future, this intention may trigger many other activities. Of more interest is the state of execution for the `Deliver-Object` intention. At its topmost level, this parent intention has the single child intention `Plan-and-Execute`, which in turn is executing the `:follow-path` schema while simultaneously monitoring for execution failures (via `Monitor-Planex`). As part of the path-following schema, the robot is currently moving from a corridor to a junction, which in turn has activated an intention to move toward a specific target. At the lowest level, three behaviors are activate simultaneously, namely `Follow`, `Orient`, and `Keep-Off-With-Target`.

```
∗ Avoid (Avoid1) END
  ∗ Avoid-Collision (Collide) B

∗ Deliver-Object (I3674) S171
  ∗ Plan-And-Execute (I3675) S146
    ∘ Monitor-Planex (I3676) CLEANUP
    ∗ Follow-Path (Follow-It) S138
      ∗ Corridor-To-Junction (I3677) END
        ∗ Follow-To-Target (I3678) END
          ∘ Follow (Follow-To-Target) B
          ∘ Orient (Orient-To-Target) B
          ∘ Keep-Off-With-Target (Keep-Off) B
```

Figure 3: Snapshot of the Intention Structures during Execution of a Delivery Task

[3] To improve the understandability of the robot's actions, PRS-Lite maintains an *intention display* that summarizes the intention structures at the end of each execution cycle. The intention display provides a concise overview of the motivation for the actions being undertaken by the system at any point in time, thus conveying to an observer *why* the robot is behaving in a certain manner.

Evaluation

PRS-Lite provides a powerful and natural framework in which to specify and manage the purposeful activities of a robot. The system itself is compact (<500 lines of LISP code, including executor, compiler, and display manager), especially in comparison to PRS-Classic. It has proven to be sufficiently fast over a broad range of tasks, with the intention execution loop easily fitting into Flakey's overall cycle time of 100 ms. Precise figures for the cycle time of the PRS-Lite executor are not available, but the combination of behavior and task control lies somewhere in the range of 5 to 30 ms (on a Sparc-2 processor). This is very fast, considering that on average there are 10 to 15 intentions in operation, monitoring various conditions and coordinating behaviors.

PRS-Lite was developed originally to support specification and runtime management of activities for the tasks of the 1993 NCAI robot competition (Konolige 1994). Since that time, it has been used extensively as the task-level controller onboard Flakey. We have written more than 50 nontrivial activity schemas, spanning tasks such as directed navigation, real-time visual tracking of people, autonomous exploration and map construction, and delivery assignments. One of the most complex involved a scenario in which the robot was charged with retrieving an object from one person and then delivering it to a second person, given only default information regarding the whereabouts of the first person. In particular, the robot had to act appropriately in situations where the individual was not at the default location by soliciting her whereabouts from a human, and then reasoning with that information to determine how to proceed.

In contrast to many recent systems for reactive task control, PRS-Lite does not support explicit runtime deliberation about goals and methods for achieving them. This approach was adopted both out of concern for the cost of deliberation and because such deliberation is not essential for task-level control. Responsibility rests with the activity schema designer to create adequate and comprehensive representations of actions that directly dispatch procedures as required. Thus, the representations in the system emphasize the essence of procedural reasoning systems: the declarative representation of procedural knowledge for achieving goals.

The expressiveness of PRS-Lite has proven mostly adequate for the tasks handled to date. However, one shortcoming is the inability to suspend execution of an intention at an arbitrary point, for continuation at some future time. A suspension capability would enable certain activities to be postponed when more urgent matters arise. For example, consider the occurrence of an unexpected event during the execution of a navigation plan, for which immediate attention is required. Ideally, we would like to suspend the path-following intention (and its derivative intentions and behaviors) temporarily, resuming with the path-following after the problem has been addressed. In the current framework, path-planning must be aborted then restarted after the problem is resolved. (The only alternative is to anticipate all possible problems directly in the activity schemas for plan execution – an unlikely prospect.)

Control of Fuzzy Behaviors

Behaviors composed from fuzzy rules introduce unique challenges for task-level control. One major advantage of the fuzzy rules approach is the smoothness of activity that results from rule blending. However, sequencing languages (including activity schemas) model task-level events as discrete actions connected by explicit transitions, even when the tasks themselves ground out in continuous processes. This separation of task and behavior levels leads to discontinuities in the resultant activities of the system. As an illustration, consider the task of navigating to an office in a given corridor, and then entering it. Flakey has activity schemas defined for the navigation and doorway-crossing, each of which employs an appropriate set of behaviors. A straightforward approach to the overall task would be to execute a navigation intention to reach the office, and then a doorway-crossing intention to enter the office. But the termination of the corridor navigation intention prior to the activation of the doorway-crossing intention leads to jerky, unnatural motion by the robot. For smoother operation, the doorway-crossing behaviors must be activated before the robot reaches the office (i.e., while the corridor behaviors are still active). The corridor navigation behaviors should terminate once the robot is sufficiently far down the corridor to enable successful completion of the doorway-crossing intention. Such intention-level blending can be specified in the PRS-Lite goal language, using a combination of monitors and thresholded blocking of intentions.

PRS-Lite vs PRS-Classic

Several features of PRS-Classic were consciously omitted from PRS-Litebecause of concerns regarding their computational overhead. However, certain of these constructs could enhance and improve PRS-Lite. Given the availability of unused cycle time in Saphira, it would be interesting to extend the system in these directions, and to perform experiments to assess their associated costs.

One extension would be to add a database to provide explicit, declarative representations of beliefs about the world. In PRS-Lite, this state information is stored in a combination of the Local Perceptual Space and a set of environment variables. This approach is adequate (although inelegant) for the current system, but a database would support more general reasoning capabilities.

A second extension would be to provide limited runtime deliberation to activate schemas. This would enable selection from multiple candidate schemas for achieving a goal, rather than the current approach of directly dispatching intentions. Such deliberation would require explicit declarations of the applicability conditions and effects of individual activity schemas. By itself, this extension would not directly increase the overall capabilities of the system (although it would improve modularity), since conditional branching can be used to embed the decision-making process in activity schemas explicitly. However, declarative specifications of effects and applicability conditions for schemas are necessary to enable automated composition of schemas by planners. We have performed some initial efforts in this regard, as described in the conclusions.

One of the unique and heralded features of PRS-Classic is its capacity for general-purpose metalevel reasoning, a capability not included in PRS-Lite. Metalevel reasoning plays a critical role in PRS-Classic applications. Its two most important uses are to enable selection among multiple applicable procedures for a goal, and to override the default control strategy for dispatching procedures. Neither of these capabilities is necessary in PRS-Lite. The first is irrelevant, because intentions are dispatched directly rather than chosen by deliberation. The second is unnecessary because the sequencing goals in PRS-Lite enable specifications of extremely powerful control strategies, much more so than in the representations that underlie PRS-Classic (Wilkins & Myers 1995).

While PRS-Lite was conceived originally as a derivative of PRS-Classic, the experiences in designing and using PRS-Lite have provided insights into improving the original system. The most significant of these are techniques for controlling concurrent and interacting continuous processes, a more powerful control regime that supports both blocking and nonblocking parallelism, and the generalized goal semantics that replaces concepts of success or failure with levels of goal satisfaction.

Related Work

Managing Fuzzy Behaviors

The most closely related work on controlling fuzzy behaviors is the *control structures* paradigm (Saffiotti, Ruspini, & Konolige 1995). Control structures specify declarative statements of the contexts of applicability and goal conditions for behaviors implemented as fuzzy rules. Goal regression through these structures is performed to collect the behaviors required at any stage to achieve a designated task. Execution of the resultant task involves enabling this set of behaviors, with the hope that environmental conditions will lead to appropriate activation or suppression of behaviors.

The control structures approach provides smooth blending of behaviors across a task. However, the use of a fixed, global set of behaviors throughout the duration of a task has several drawbacks. Behaviors can become activated for reasons other than those that motivated their inclusion in the regression set. Such unintended activations may be serendipitous at times but can also produce harmful interactions. For instance, consider tasking a robot to navigate to the end of a corridor to retrieve some object, and then to deliver it to an office midway along that corridor. The second phase of this task requires a behavior for crossing a doorway, which would get activated whenever the robot is sufficiently close to it. But the enabling of this behavior throughout the task would cause the robot to enter the doorway prior to obtaining the delivery object located at the end of the hallway, since there is no goal context to restrict the behavior's activation.

The lack of an explicit goal state at runtime also limits the understandability of the robot's activities by an observer. To make this point more concrete, it is easier to interpret the actions of a robot given the intention structures of Figure 3 than a listing of the current behaviors and their activation levels. Such understandability of autonomous systems (both hardware and software) will be critical for their deployment in real-world settings, especially to enable human interactions with them (Myers & Konolige 1992). Finally, the control structures approach focuses on behavior management exclusively, providing no mechanisms for synchronizing with nonbehavior activities (such as control of speech generation and understanding).

Reactive Control Frameworks

Several task-level controllers for managing the activities of mobile robots have been designed and built in recent years. Of those, the most similar to PRS-Lite is the extended version of the RAP system that manages sets of interacting continuous processes (Firby 1994). The differences between this version of RAP

and the original (Firby 1987) mirror the differences between PRS-Lite and PRS-Classic. Key capabilities provided in the the second-generation systems that were not present in the first include the management of concurrent continuous processes, and the elimination of reliance on goal success or failure. PRS-Lite and the extended RAP system provide comparable expressive power but the underlying knowledge representations have different flavors. PRS-Lite builds many key control constructs into its goal language, whereas they are implicit in the RAP formalism. For instance, **Unintend** goals enable explicit termination of active processes; in RAP, termination is handled implicitly through appropriate combinations of other constructs. RAP similarly can be made to block tasks until a designated success criterion is satisfied, but there is no explicit operator for doing so in its underlying language (in contrast to the various modes of the **Intend** operator in PRS-Lite). PRS-Lite also provides richer sequencing operations than does RAP. We note that RAP supports runtime deliberation for enabling tasks, a feature that was intentionally excluded from PRS-Lite.

Conclusions

Work on reactive control has had a tendency to 'reinvent the wheel': research teams design new architectures from scratch, often influenced by the idiosyncrasies of the particular tasks at hand. In contrast, PRS-Lite is a task-level controller whose design capitalized on experience with a mature, sophisticated reactive control system.

PRS-Lite is a successful controller for several reasons. It provides a rich mechanism for composing and sequencing processes to achieve specific tasks while being responsive to unexpected events. It provides a natural goal-decomposition semantics that affords direct understandability of system activities, and uniform dispatching of task- and behavior-level activities that supports both discrete and continuous activities. One of its most unique features is the task-level blending of behaviors implemented as fuzzy rules, thus enabling smooth integration of overlapping, lower-level continuous processes. Finally, it has been exercised extensively in the role of task-level controller for a mobile robot.

There has been little need for planning capabilities in the tasks handled to date by our robot, other than basic route planning required for navigation tasks. Certainly, applications of a more demanding nature will require generalized look-ahead planning. We have used Sipe–2, a hierarchical generative planner (Wilkins 1988), in conjunction with PRS-Lite to compose more complex action sequences for achieving tasks that require such deliberation. For this integration, a designated set of PRS-Lite activity schemas serves as the primitive operators in Sipe–2 (in much the same way that Sipe–2 and PRS-Classic interact in the Cypress architecture (Wilkins *et al.* 1995)). These schemas are annotated with declarative specifications of their applicability conditions and effects. Additional activity schemas are defined for issuing a request to Sipe–2 to solve a particular planning problem, waiting for Sipe–2 to generate a plan, mapping the operators in that plan to activity schemas, and then invoking those schemas in the appropriate order. Monitoring by PRS-Lite intentions detects problems during plan execution, reinvoking Sipe–2 as required to generate modified plans. Sipe–2 cannot directly track changes in the world-state that arise during plan execution, so replanning requests issued by intentions are annotated with summaries of those aspects of the world-state that could impact plan generation. The planning and replanning capabilities have been used to date only in a restricted capacity (given the limited set of purposeful actions that our robot can perform), but illustrate the potential for generative planning based on activity schemas. We note that for nontrivial tasks, the amount of reasoning required to formulate a satisfactory plan will far exceed the short duration of the sense-act cycle required by a robot. A more flexible integration model is required in which the planner operates as an asynchronous process that communicates with the reactive controller when necessary.

References

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

Firby, R. J. 1994. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems.*

Garvey, T., and Myers, K. 1993. The intelligent information manager. Final Report SRI Project 8005, Artificial Intelligence Center, SRI International, Menlo Park, CA.

Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 809–815. MIT Press, Cambridge, MA.

Georgeff, M. P., and Ingrand, F. F. 1988. Research on procedural reasoning systems. Final Report Phase 1, Artificial Intelligence Center, SRI International, Menlo Park, CA.

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

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

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

Ingrand, F. F.; Goldberg, J.; and Lee, J. D. 1989. SRI/GRUMMAN Crew Members' Associate Program: Development of an authority manager. Final Report SRI Project 7025, Artificial Intelligence Center, SRI International, Menlo Park, CA.

Konolige, K.; Myers, K.; Ruspini, E.; and Saffiotti, A. 1996. The Saphira Architecture: A design for autonomy. *Journal of Experimental and Theoretical AI*. To Appear.

Konolige, K. 1994. Designing the 1993 NCAI robot competition. *AI Magazine*.

Moravec, H. P., and Elfes, A. E. 1985. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 116–121.

Myers, K. L., and Konolige, K. 1992. Semi-autonomous robots. In *Proceedings of the AAAI Fall Symposium Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*.

Myers, K. L. 1993. *User's Guide for the Procedural Reasoning System*. Artificial Intelligence Center, SRI International, Menlo Park, CA.

Nilsson, N. J. 1994. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research* 1:139–158.

Payton, D. W. 1990. Exploiting plans as resources for action. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*.

Saffiotti, A.; Ruspini, E. H.; and Konolige, K. 1993. Integrating reactivity and goal-directedness in a fuzzy controller. In *Proceedings of the 2nd Fuzzy-IEEE Conference*.

Saffiotti, A.; Ruspini, E. H.; and Konolige, K. 1995. A multivalued logic approach to integrating planning and control. *Artificial Intelligence* 76:481–526.

Simmons, R. G. 1994. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation* 10(1):34–43.

Wilkins, D. E., and Myers, K. L. 1995. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation* 5(6).

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI* 7(1):197–227.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann.