Task Execution: Interfacing to Reactive Skill Networks

R. James Firby
University of Chicago
1100 East 58th St.
Chicago, IL 60637
firby@cs.uchicago.edu (312) 702-6209, FAX (312) 702-8487
and
Marc G. Slack
The MITRE Corporation
7525 Colshire Drive
McLean, VI 22012-3481
(703) 883-7738
slack@mitre.org

Keywords: Task Execution, Robotics, Autonomous Agents

Abstract

We describe an implemented two layer architecture for real-time task execution in physical agents. The system integrates the RAP system with a set of dynamically configurable reactive processes called skills. Thus the notion of a primitive action is replaced by the selection of a set of skills that interact directly with each other and with the world to form a reactive agent.

Using this approach, interpretation of sensory information becomes highly context dependent. For example, a short reading on a forward sensor might be a threat to be avoided or it may be the ticket counter being approached. To address this issue the RAP system has been augmented to allow RAPs at all levels to modify memory. This modification allows sensory information to be interpreted at the appropriate level of abstraction. We have implemented this system and used to control a robot navigating the halls at MITRE.

1 Introduction

We have constructed a two level control system that uses two different types of representation. The low-level consists of continuous processes, called skills [10], that are designed for describing control tasks that use real-time feedback. The high-level consists of a task execution system that uses symbolic reactive plans called RAPs (Reactive Action Packages) [4]. RAPs are designed for describing task control plans at the level of networks of discrete steps. Two levels are used because it is easier to describe low-level robot tasks in terms of continuous processes and higher-level tasks in terms of hierarchical reactive plans.

This paper describes the skill system and the way it interfaces to the RAP system with particular emphasis on the way the RAP system maintains a reasonable model of the situation that the robot is in. In a multi-level architecture it is crucial to be able to synchronize the processing that takes place at each level. The constructs described in this paper allow the slower RAP system to maintain a symbolic model of the current situation while at the same time freeing the low-level skills to respond to the environment in real-time.

The complete system has been implemented and tested on a mobile robot at MITRE doing a set of simple hallway navigation tasks.

1.1 A Simple Navigation Domain

The robot tasks in the navigation domain involve moving between rooms and into the elevator along the hallway connected to the MITRE Autonomous Systems Laboratory. The robot uses sonar data for obstacle avoidance and a laser scanner with bar-coded tags for landmark recognition. The laser landmarks consist of two coded tags. The laser scanning system returns only the angle to the tags in the robot's coordinate system. Using these angles in combination with a rough estimate of the distance between the tags allows the robot to roughly determine the distance and direction to the landmark when both tags are visible. Figure 1 depicts this domain.

1.2 A Two Level Architecture

Within the navigation domain, all of the processes that actually move the robot are implemented as continuous skills. The primary navigation skills are Navigation Template-based (or NaT-based) [9] processes that use sonar information to avoid obstacles while moving to a given coordinate or moving in a particular direction. There is also a special sonar-based skill for aligning with a doorway and moving through it. At this level of description, skills view

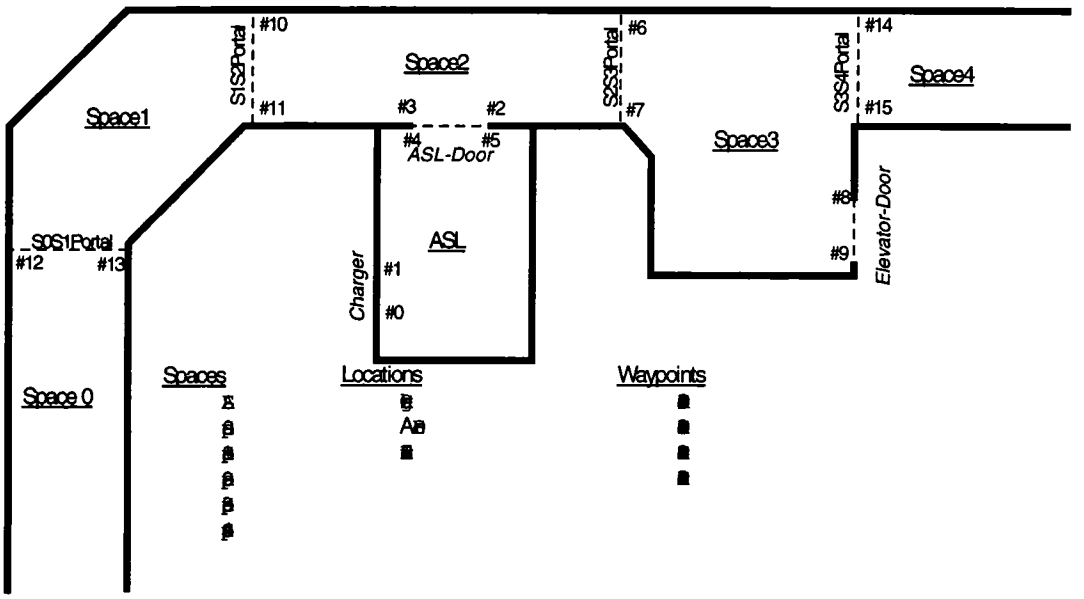

Figure 1: The Navigation Task Domain

the world in terms of sonar readings and dead-reckoned coordinates. There is no notion of routes to follow, rooms or elevators.

The RAP execution system, on the other hand, contains plans for moving from one landmark to another within a space and builds routes for the robot to follow to move into new spaces or to find landmarks that are currently out of sight.

This division of task representation is captured in the two level architecture shown in Figure 2.

The levels communicate with one another through a well-defined interface:

- Skills are enabled and disabled by the RAP system to set up different configurations of the control system to accomplish different tasks
- Enabled skills generate signals that are sent back up to the RAP system to tell it when tasks are complete or failing and when important state changes occur.

The rest of this paper discusses the ways this simple interface can be used to achieve a variety of tasks and to keep the RAP memory and execution state in sync with what is happening at the skill level and in the world.

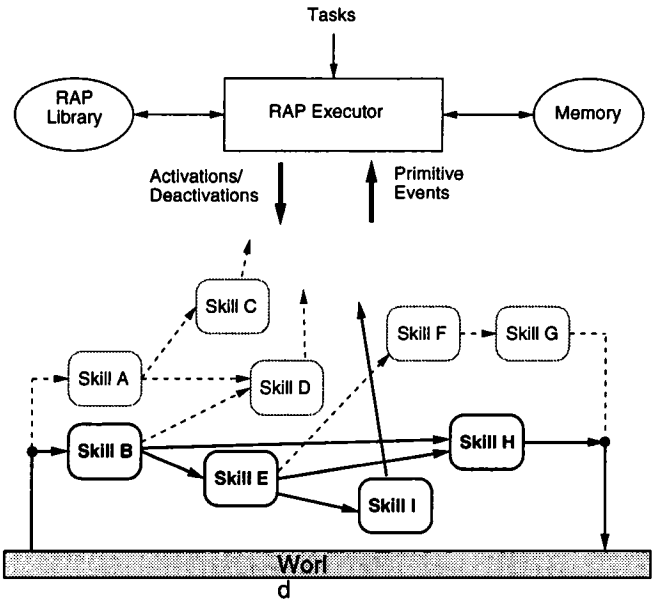

Figure 2: The Two-Layer Architecture

2 Reactive Skill Networks

Consider the process of navigating through a hallway. A reactive agent for accomplishing this task can be constructed with a few interacting skills (see Figure 3). The `Track Hallway` skill takes the robot's sensor information and maintains a robot-relative estimate of the hallway's orientation and width. The `Detect Obstacles` skill, maintains a robot relative model of the local obstacles using the sensor information. Finally, the `Follow Trajectory` skill takes as inputs the "model" of the hall and obstacles to produce drive and steer commands which drive the robot along the hallway while avoiding obstacles. Such skill networks should be viewed as active sets of interacting concurrent processes. In reality, it does not matter if the skills are running on the same CPU or on different CPUs, as long as their operating frequency is sufficient to achieve the necessary reactivity for the operating domain.

Now consider a similar skill network for maneuvering through a doorway. Figure 4 shows a skill network which allows the robot to move through a doorway without bumping into any obstacles. This skill network uses both the `Detect Obstacles` and the `Follow Trajectory` skills, however the `Track Hallway` skill has been replaced with the `Track Door Jambs` skill. This new skill network tracks the door jambs and obstacles in real-time, allowing the robot to achieve motion along a trajectory which moves the robot through the doorway while avoiding obstacles.

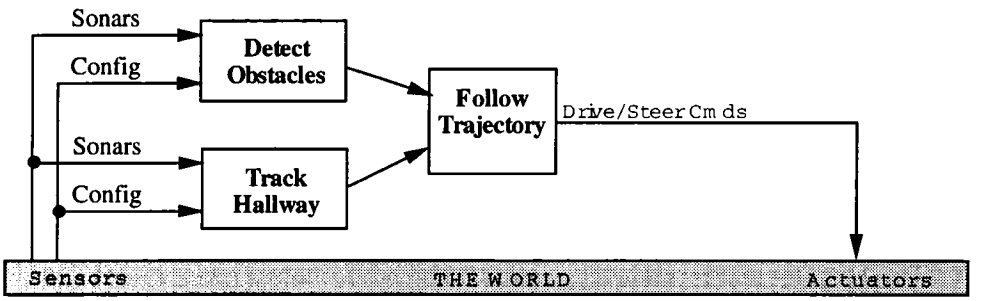

Figure 3: Skill network for navigation along a hallway

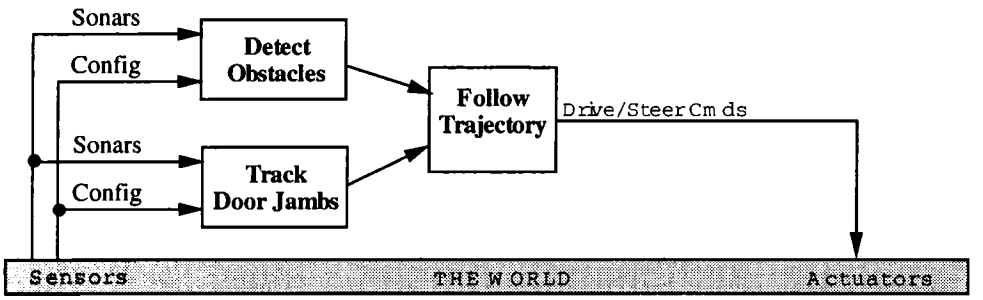

Figure 4: Skill network for maneuvering through a doorway

If constructed properly such skill networks allow the robot to follow halls and enter doors independent of the specific hallway or door that the robot encounters[1]. Still, there is the remaining problem of determining when to switch from one skill network to another. In a purely reactive system [2], both the follow hallway and the pass through door "behaviors" would be running in parallel. Activation of the appropriate behavior would be accomplished through the use of inhibitory links which would mute the outputs of the portions of the network not required in the current situation. However, it is unclear when the situation should change and as the domain of potential tasks increases so to does the complexity of these inhibitory links. This later point makes maintenance and expansion of such a network unwieldy. Rather than having such context switching embedded implicitly within the reactive network, we have opted for an explicit approach to context switching.

Revisiting the skill network for following a hallway (see Figure 3), one sees that information is flowing from sensors to actuators via some computational

[1]We have demonstrated this point using "Uncle Bob" our Denning mobile robotic platform.

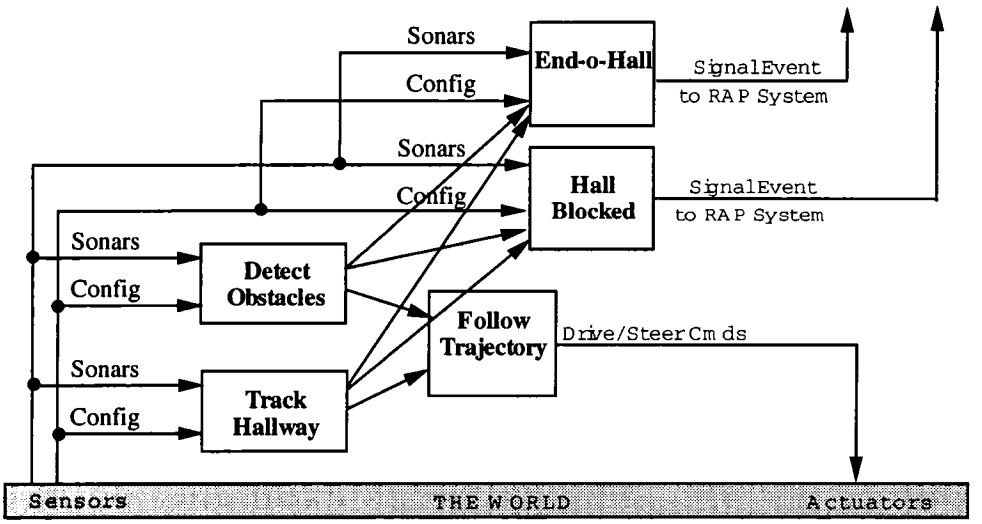

Figure 5: Augmented skill network for navigation along a hallway

transforms. This is typical of reactive control. However, for the robot to accomplish anything other than hallway following, this task of hallway following should be maintained until some environmental condition is met. For example, the hallway following task might be defined as "follow hall until end or until failure". This raises the question of detecting termination conditions.

In our approach, termination conditions are detected through the addition of two specialized skills referred to as event skills or simply events. Events can take inputs from any of the sensors as well as the outputs of other skills in order to trap specific conditions in the world. Once trapped the event sends an asynchronous message to the RAP layer in the architecture. Figure 5 shows a skill network which will follow a hallway and trap conditions which indicate that the end of the hall has been reached or that the hallway appears to be blocked. These event skills allow the RAP layer of the architecture to trap the termination conditions for the task and modify the skill network accordingly. This interaction between the skill layer and the RAP layer allows the RAP layer to dynamically define the "reactive agent" for the task at hand.

Skills are modular pieced of code (typically C code) which have a declaration file describing their inputs, outputs, and parameters. The parameters are values that can be manipulated by the RAP system to allow situated configuration of the skills performance. The inputs of to a skill always come from other skills. In the case of information coming from a sensor (i.e., the world) some skill or set of skills is responsible for that interface and is transparent to the external view of the skill. The routing of the skill's outputs determines the type of the skill. If the outputs are directed to other skills

then the skill is a computational block. If the outputs of the skill are directed back to the RAP system then the skill is known as an event. Event skills also include a test to determine if the output of the skill should be passed to the RAP system. This later feature keeps the high frequency processing of the skill network from overwhelming the RAP system while at the same time allowing the system to trap all transient signals which might be critical to the overall operation of the robot. More information on the skill infrastructure and development environment can be found in [3, 11].

2.1 Skill Networks for Navigation

Building on these ideas, the basic skills used in the navigation experiment described above are:

- WATCH-FOR-LANDMARK watches for a particular laser landmark and generates the signal `landmark-visible`.
- MOVE-IN-DIRECTION moves the robot in a particular direction while avoiding obstacles. It generates the signals `movement-complete` and, in some situations, `stuck`.
- MOVE-TO-LANDMARK moves to a given distance and orientation with respect to a visible laser landmark while avoiding obstacles. It generates the signals `movement-complete`,`lost-landmark`, `landmark-visible` or `stuck`.
- MOVE-THRU-DOORWAY orients to a doorway and moves through it, generating the signals `movement-complete`, `door-closed`, or `stuck`.

These four skills and their related events form the basis of the reactive portion of the robot. In reality these skills are constructed from a number of smaller interacting skills, however for purposes of this discussion that fact can be ignored. In the next section we will see how the RAP system manipulates these skills to accomplish non-local tasks.

3 Configuring Skill Networks with RAPs

The ability to create information processing networks is only part of the solution. Once such networks can be constructed they need to be controlled and configured by a system which has an understanding of the current context and the current set of tasks that the robot is trying to accomplish. In our approach, this skill management job is handled by a Reactive Action Package

(or RAP) interpreter [4]. This provides the robot programmer with a task-oriented language for describing and controlling the robot's actions. The interface between these two systems is straight forward as is evident in the following code which depicts the RAP for following a hallway.

```
(define-rap (follow-hall-to-end)
  (method
    (primitive
      (enable (:detect-obstacles)
              (:track-hallway)
              (:follow-trajectory))
      (wait-for (:end-o-hall)   :succeed)
      (wait-for (:hall-blocked) :fail)
      (disable (:detect-obstacles)
               (:track-hallway)
               (:follow-trajectory)))))
```

When the above RAP comes up for execution it first enables the three skills and the two events forming the skill network depicted in Figure 5. Then it blocks (i.e., the `wait-for` syntax) waiting for one of two events to occur. If the end of the hallway is reached then the RAP continues disabling the three skills[2] and returning a succeed state to the parent RAP or to the top level. If however, the hall is detected to be blocked then after the skills are disabled the parent RAP or top level is returned a failure state. As a final note to this example notice that there is no reference in the RAP to the way in which the skills are connected together. This is assumed to be implicit in the skills them selves. More on this last point follows.

3.1 Using RAPs to Navigate

Built up on top of these "primitive" RAPs, is a hierarchy of RAP plans to:

- Move to a landmark in the current space.
- Move into a neighboring space.
- Move through a set of connecting spaces to a destination landmark that cannot currently be seen.

For example, to move from a doorway to a known landmark in the same space, the RAP system looks in its memory for an approximate direction and distance to move from the door to be within sight of the landmark. It then

[2]note that the events are implicitly deactivated and removed whenever the task step which spawned them is completed

enable both the MOVE-IN-DIRECTION NaT-based skill and the WATCH-FOR-LANDMARK skill to watch for the landmark. When a `landmark-detected` signal is generated, the RAP system disables both of these skills and enables the MOVE-TO-LANDMARK NaT-based skill using real-time feedback from the laser tag system.

In effect, the RAP system is selecting which reactive agent will best achieve the next subtask in the current situation. We feel that this is an advantage over more traditional approaches to reactive control [1, 2, 8] because it limits the scope of the problem of constructing the reactive portion of the robot. Rather than having to embed a reactive agent for moving through a door with one which can maneuver through a hallway, this approach allows these two reactive agents to be coded independently and activated only in their appropriate context.

3.2 Path Planning with RAPs in the Navigation Domain

An interesting result of dividing up the navigation problem between RAPs and skills the way we have done, is that the RAPs deals almost entirely with the concepts of "spaces", "doorways", and "landmarks" while the skills deal almost entirely with dead-reckoning distance and sonar readings, using landmarks only as targets and not representing doorways or spaces at all. The skills take care of moving, landmark detection, and obstacle avoidance, while the RAP system concerns itself with choosing appropriate way-points for the skills to use within a space and finding sequences of the spaces to travel through to reach distant locations.

The RAP system uses a library of plans for navigating within a space, for moving to a door in the current space and for passing through doors into an adjacent space. To travel longer distances, the RAP system uses a simple breadth-first path planner to find a sequence of spaces that will lead from the current space to a goal location. This resulting sequence is encoded as a standard RAP method and becomes part of the system's repertoire for use in the same situation in the future. The ability to call expert problem solvers to generate a method for a RAP task when no known method applies, is a standard part of the RAP system [4]. In the navigation domain, the path-planner is used to create new methods for the RAP TRAVEL-TO-DISTANT-SPACE when there is no existing method for getting to the goal space from the current space.

To represent the structure of the domain a number of propositions are represented in the RAP memory.

1. Location data: This information describes the laser landmarks. For example, `(location-data charger 0 1 1.0 "Charger")` represents the

charger location in memory. It indicates that the charger location is represented by the left bar code of `0` and a right bar code of `1`. Further the rough estimate of the distance between the tags is `1.0` meters. The final slot in this proposition is simply the print name of the location used for debugging as well as the robot's vocalizations.

2. Connection data: Representation of space also requires that the robot know which spaces are connected to which other spaces and which locations connect spaces together. For example, the `(connects asl-exit asl space2)` proposition indicates that the location denoted `asl-exit` connects the `asl` space to the `space2` space. Using this information the robot is able to find a sequence of spaces which will move the robot from its current space to its goal space.

3. Relative location: Finally there is the information required to allow the robot to maneuver from its current location to a new location within a given space. The information required to carry this out is contained in a `relative-loc` proposition. For example, `(relative-loc asl charger asl-exit 4.0 3.0)` indicates that in the space asl the relative location from the `charger` to the `asl-exit` is `3.0` meters in a direction `4.0` radians in the counter clockwise direction with respect to the orientation of the robot when it is positioned at the `charger`. This information does not encode the absolute position between the `charger` and the `asl-exit`, but rather a vector which will allow the robot to move into the spatial zone within which the `asl-exit` will be visible. For the example given the real absolute encoding of the relative location from the `charger` to the `asl-exit` would be `(relative-loc asl charger asl-exit 4.44 10.0)`. This encoding of relative location information is similar in some respects to Kuipers' [7] encoding of distinctive locations.

3.3 RAP Memory in the Navigation Domain

It is only possible to use the information discussed above for constructing navigation plans for RAPs if the robot also has a good understanding of its current place in the map. It must know the space it is in and an approximate dead-reckoning location within that space. Each time the robot moves through a door, either intentionally or unintentionally, it must update the space it is in. Also, whenever a landmark is detected, the robot must be able to update its model of what it is near and how it is oriented.

The RAP system maintains this model of the world based on the signals returned by the skills while they are running.

4 Discrete Steps as a Set of Skills

We have already discussed how the RAP system creates a discrete task step by enabling a set of skills and then waiting for a signal that says the desired goal for that set of skills has been achieved (or something is going wrong). The PRIMITIVE method type can be used in a RAP definition to specify a set of skills to be enabled and disabled when the method is selected for execution. WAIT-FOR clauses can also be specified in primitive RAPs, and in other RAP task-nets as well, to stop current processing of the method and wait for a specified signal to be generated as a skill event. RAP task-nets and WAIT-FOR clauses allow for quite general ordering of tasks contingent on skill outcomes and other events in the world. For more detail on this subject see [5].

However, creating discrete steps out of reactive agents is only part of the problem of interfacing to a skill network. The RAP system must also interpret the results of a taking a step in the world. Every reactive agent the RAP system invokes will move the robot and change its position, possibly moving it through a door and changing its space as well. Also, some skills will move the robot next to a landmark and thus to a known location. Some of these changes are directly detectable after the fact, a nearby landmark may remain visible, but some of the changes are inferences from the fact that the step appeared to complete successfully. Going through a door into a new space changes the robot's idea of what space it is in but the robot has no way of directly sensing what space it is in.

4.1 Interpreting Skill Set Results in Context

The proper interpretation of the results of executing a set of skills depends on the context in which the results are generated. The results of very specific skills may be easy to interpret independent of the goal they are used to achieve while generic skills will tend to generate signals with little intrinsic meaning. For example, the skill MOVE-THRU-DOORWAY generates the signal `movement-complete` when it has gone through a door. This signal does not tell the RAP system that the robot has moved from one space to another or where it is in the new space. The context in which signals and task completions appear holds the key to appropriate interpretation.

The context in which a set of skills is enabled and completes is defined by the hierarchy of active RAP plans in which the skills arise. Thus, the proper interpretation of the results of those skills can be made by running up the task hierarchy and inferring appropriate changes to memory based on the tasks encountered. In practice, there are two ways to make this happen:

- The RAP system could make a record of what RAP tasks have been executed and what results occurred. Then, when the current state of

the world needed to be checked, this record could be used to infer the appropriate state of the facts being queried.

- Or, as each RAP completes, the system could infer appropriate changes to the world model based on its results and make those changes immediately. The effect of hierarchical context would be taken care of when RAP tasks at higher levels of abstraction completed and made inferences at higher levels of abstraction.

As implemented in the two level architecture described in this paper, the RAP system does the latter.

4.2 RAP Memory Rules

To handle inference and memory updates, the RAP system uses *memory rules.* Memory rules can be thought of as "add/delete" lists associated with each RAP. Each RAP can have an associated memory rule and when it completes, the rule is executed to update memory according to the results of the RAP.

RAP memory rules must actually be conditional "add/delete" lists because the actual memory changes required may depend on the results of executing the RAP. For example, in the simplest case, executing the primitive RAP that enables the MOVE-TO-LANDMARK skill will place the robot in a new, known location if it succeeds but does not if it fails. This example is captured below:

```
(define-memory-rule (move-to-landmark ?loc ?speed ?standoff) :finish
  (match-result
    (SUCCEED
      (rule ((and (last-seetagsevent ?dist ?tangle)
                  (current-space robot ?space)
                  (orientation ?xpos ?ypos ?angle ?head))
             (mem-add (current-location robot ?loc))
             (mem-del (current-location))
             (mem-add (current-location ?loc ?space ?tangle
                                        ?xpos ?ypos)))))))
```

This memory rule states that if the result of the MOVE-TO-LANDMARK RAP is SUCCEED, then the current location of the robot should be updated to the landmark location. If the RAP did not succeed, nothing is changed.

In the RAP system, memory rules may contain arbitrary LISP code and can thus do arbitrary inference. However, the "add/delete" model is sufficient for the majority of actual cases encountered so far in our simple robotic tasks.

4.2.1 Updates that Depend on Detecting Events

Often, memory rules executed at the completion of a RAP task do not capture all of the changes that have occurred in the world. There are many events in

the world that do not leave lasting, easily detectable changes. For example, in the navigation domain, the laser scanner may detect landmark tags for one location while moving to another. If the system wants to record the latest known location of this landmark, then memory needs to be updated when the tags are detected and not after the RAP that started the robot moving completes.

To capture transient events, the RAP system extends the memory rule concept with the idea of event memory rules. Any RAP can have associated with it a memory rule to execute if a signal is received during the course of execution of the RAP. This event memory rule is called with a "result" that corresponds to the signal that arrived. For example:

```
(define-memory-rule (move-to-landmark ?loc ?speed ?standoff) :event
  (match-result
    ((landmark-visible ?mark ?dist ?angle)
      (rule (orientation ?xpos ?ypos ?rangle ?head)
            (mem-add
              (landmark-location ?mark
                 (new-x ?xpos ?ypos ?rangle ?head ?dist ?angle)
                 (new-y ?xpos ?ypos ?rangle ?head ?dist ?angle)))))))
```

Other events of importance in the navigation domain are being pushed through a doorway accidently while avoiding obstacles. Sometimes this event can be detected using sonars and when it is, memory must be updated to show that the robot is now in another space.

Event memory rules are also used to interpret streams of signals generated during the course of carrying out a task. For example, the current robot position might be updated continually by sending back signals that give the latest values.

4.3 Interpreting Unexpected Events

A problem that arises in trying to interpret changes in the world through the skill system and the RAP task hierarchy is that sometimes totally unexpected events occur. Given memory rules as they are defined above, there is no way to interpret an unexpected event because there will be no active RAP with an appropriate event memory rule.

One solution to this problem is to set up independent monitor RAPs that sit waiting for particular, otherwise unexpected, events to occur. This approach is discussed at some length in [4] and puts an event rule in place to change memory when the event is signaled.

However, putting a RAP in place to expect the event means that the event is no longer truly unexpected. This may seem like a hack, but in fact, it isn't generally possible to make sense out of a truly unexpected event. As argued

above, events typically need to be interpreted in context to have a meaning. In the navigation domain, the signal `movement-complete` has virtually no meaning without knowing what movement task is being executed at the time. If this signal were to arrive unexpectedly, it would have know meaning at all, except possibly that something unexpected had happened. Either an event is not unexpected and has a RAP task context in which to interpret it, or the event itself has a unique meaning, in which case it can be interpreted in a context-free manner. It is likely that real robotic systems will have some context-free signals, like a bump sensor triggering, but it might also be argued that such signals are not really unexpected and are designed to be readily interpreted.

Context-free events can be implemented more efficiently than introducing a monitor RAP to supply an appropriate event rule. Event memory rules can be indexed under the events they handle and can then be triggered directly when such an event is signaled. The implemented RAP system allows event memory rules unconnected to RAPs for this efficiency gain. For example, in the navigation domain, global position updates are unambiguously interpretable (at least some of the time) so they don't need a RAP to give their interpretation context.

5 Conclusion

By interfacing the RAP system to a model of continuous activity we have created an effective way to mediate between the symbolic nature of tasks plans and the continuous nature of real-time interaction with the world. Within this model, the skill system is responsible for handling the real-time interaction as well as trapping transient signals which indicate state transitions with respect to the current RAP task. Such control loops are best written in a language like C where it is easy to optimize code for real-time feedback loops.

The RAP system has the responsibility of interpreting the results of executing skills and events that occur while they are running to maintain a reasonably accurate interpretation of the system's current situation. The RAP system then uses that understanding of the situation to select appropriate sets of skills to act in that situation. An advantage of having the situational information maintained in the RAP system is that the skills are no longer responsible for interpretation of the situation, which is known to be difficult to encode in a distributed set of concurrent processes.

One reason it is difficult to interpret events in the skill system is that changes in the world do not have context-free meanings. However, the RAP system is in a position to interpret signals sent from the skill system in the context of the goal structure of the current reactive plan. The RAP system uses MEMORY-RULES triggered by task completion and signaled events to make

actual changes to memory.

References

[1] Ronald C. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *International Conference on Robotics and Automation*, Raleigh, NC, March 1987. IEEE.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), March 1986.

[3] C. Elsasser and M. G. Slack. Planning with Sequences of Situated Skills. In *Proceedings of the AAIA/NASA Conference on Intelligent Robots in Field, Factory, Service and Space*, March 1994.

[4] R. J. Firby. *Adaptive Execution in Complex Dynamic Worlds.* Ph.D. thesis, Yale University Department of Computer Science, January 1989.

[5] R. James Firby. Task networks for controlling continuous processes. In *Second International Conference on AI Planning Systems*, Chicago, IL, June 1994.

[6] E. Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots.* Ph.D. thesis, Virginia Polytechnic Institute Department of Computer Science, April 1991.

[7] Benjamin J. Kuipers and Yung-Tai Byun. A Robot Exploration and Mapping Strategy Based On a Semantic Hierarchy of Spatial Representations. *Robotics and Autonomous Systems*, Volume 8, 1991.

[8] D.W. Payton. An architecture for reflexive autonomous vehicle control. In *International Conference on Robotics and Automation*, San Francisco, CA, 1986. IEEE.

[9] M. G. Slack, *Situationally Driven Local Navigation for Mobile Robots.* Department of computer science, Virginia Polytechnic Institute, April 1990. also published as Jet Propulsion Laboratory Publication 90-17.

[10] M. G. Slack. Sequencing formally defined reactions for robotic activity: Integrating RAPS and GAPPS. In *Proceedings of the SPIE Conference on Sensor Fusion*, November 1992.

[11] S. Yu, M. G. Slack, and D. P. Miller. A streamlined software environment for situated skills. In *Proceedings of the AAIA/NASA Conference on Intelligent Robots in Field, Factory, Service and Space*, March 1994.