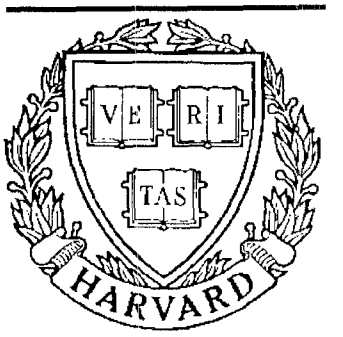
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Knowledge Strata Reactive Planning with a Multi-level Architecture

by L. Spector and J. Hendler

TR 90-78

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
26 NOV 1990		**00-00-1990 to 00-00-1990**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Knowledge Strata. Reactive Planning with a Multi-level Architecture	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Maryland,Systems Research Center,College Park,MD,20742**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**37**	

Knowledge Strata

Reactive Planning with a Multi-level Architecture[1]

Lee Spector (spector@cs.umd.edu)
James Hendler (hendler@cs.umd.edu)

11/26/90
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

This report demonstrates the use of "multi-level" or "layered" knowledge representation in Artificial Intelligence planning systems. Although multi-level representation schemes have been in use since the earliest days of AI, certain principles and advantages of knowledge stratification have never been made fully explicit. In this paper we examine issues of multi-level knowledge representation in the context of "reactive planning systems"; that is, in systems which extend the applicability of AI planning systems to complex, dynamic domains. The complexity and real-time requirements of reactive planning have lead several researchers to propose multi-level approaches. Our aim is to improve upon the state of the art in reactive planning by bringing to bear an analysis of the principles of multi-level event and action representation. Our work has lead to the implementation of a prototype architecture (called APE, for Abstraction-Partitioned Evaluator) and, within this architecture, a reactive planner (HomeBot) which operates in a household task domain.

1. Introduction

This report is about multi-level representation and reasoning architectures, and about their use in the design of intelligent, reactive systems. It has long been recognized that an AI system can profit from the division of its computational components into relatively independent modules, and from the decomposition of its knowledge into relatively disjoint areas of expertise. Systems which are partitioned in this manner may exhibit several desirable characteristics, including parallelizability, improved program comprehensibility, and significant reductions in the size of problem-solving search spaces. While these and other virtues of modularization are well known (see, for example, the discussions of "decomposable production systems" in [45]), there has been

[1]Supported in part by NSF grant IRI-8907890 and ONR grant N-00014-K-0560. The authors are also affiliated with the UM Systems Research Center.

little discussion of the principles of knowledge representation which permit such decomposition.

Among the various proposals for modularization which have appeared in the literature, schemes involving hierarchical, "layered", or "multi-level" decomposition have held a distinguished position. To say that a system is "hierarchically partitioned" is simply to say that an ordering relation (either partial or total) is defined over the set of modules into which the system is partitioned.[2] The nature of this ordering relation, and its effects on the functioning of the system, may vary from system to system. For example, while the ordering relation might define communication paths between modules, it might or might not also affect processor allocation, constitute a partitioning of the type-hierarchy for representations across all modules, etc. While some of the benefits of partitioned systems are independent of the existence of any hierarchical structure, the utility of hierarchies is well known, and hence we will concern ourselves only with hierarchical systems.[3] Of particular interest will be an analysis of the hierarchies that emerge in representing events in the world, and the exploitation of these hierarchies in a computational architecture.

Recent trends in AI research have sparked a renewed interest in multi-level architectures. As the field has matured, researchers have attempted to extend techniques which worked in simplified "micro-worlds" to handle more realistic domains. The early techniques generally fail in realistic domains for two reasons. First, the number of objects and potentially relevant relations in the real world is far greater than that in any of the experimental micro-worlds. Since many of the reasoning processes used in early AI systems have exponential complexity, they quickly become impractical as domains become more realistic (and hence larger). Second, the real world is dynamic in a sense not applicable to the earlier micro-worlds. Events in the world continue to occur while an agent is thinking and acting, and an agent's reasoning processes must be capable of accommodating a changing, semi-predictable world. We have chosen the "reactive planning" problem — the problem of planning in complex, dynamic worlds — as the basis for our study precisely because it forces us to confront directly the difficult, interacting problems of complexity and dynamism. Multi-level architectures, if employed properly, may provide solutions for both of these problems; the modularity of partitioned systems mitigates against combinatorial complexity, and hierarchical organization is a powerful tool for obtaining real-time behavior (as witnessed by the successes in hierarchical real-time robotic control; for example, see [1] , [6], and [14]).

---

[2]We shall treat "hierarchically partitioned", "layered", and "multi-level" as synonymous expressions.

[3]The use of hierarchical taxonomies dates at least to Aristotle, and hierarchical representation techniques are ubiquitous in AI.

Significant preliminary work on the design of multi-level architectures has already been done. In particular, researchers involved in the development of (or influenced by) the HEARSAY-II speech understanding system have been extending HEARSAY-II's layered "blackboard" architecture in several ways, allowing for parallelism, alternative models of module interconnectivity, etc. The motivations of several of the groups involved in the enhancement of blackboard architectures are remarkably similar to our own. Our concern, however, is not so much with the design, but rather with the proper use of multi-level architectures. We discuss the benefits of a principled partitioning of knowledge into levels, and elucidate structures of knowledge that may be productively employed toward this end. While our implementation (first described in [54]) can be seen as modification of any of several advanced blackboard system architectures, our contribution lies in the system of representation used within this architecture, and in the class of problems that our system can solve.

The following is an outline of the remainder of this paper:

Sections 2 and 3 describe the reactive planning problem in general and in the specific domain in which we have chosen to work. Section 4 briefly describes the correct behavior of our target system in a particular problem scenario. Section 5 details our multi-level blackboard architecture and discusses the characteristics which a set of knowledge levels ought to possess in order to make the most of that architecture. Section 6 provides an analysis of the structure of events and actions which yields a specific set of knowledge levels appropriate for implementation within our architecture. In sections 7 and 8 we discuss the use of our event representation within our multi-level architecture, and briefly report on the behavior of the resulting system in the problem scenario

from section 4. Section 9 describes the directions in which this project is currently moving. We describe three well-defined shortcomings of the current system, and our plans to overcome them. Related work by other researchers is mentioned when appropriate throughout this report.

2. Reactive Planning

Intelligent agents must be capable of responding to events in the world even while they are engaged in high-level symbolic reasoning. Accordingly, much of the recent research in AI Planning Systems has focused on techniques for planning in domains for which the assumption of a static world, an assumption that was relied upon in early planning work, does not hold. Questions concerning the nature of the coupling between reasoning, perception, and external events have begun to take center stage.

The earliest research in *dynamic* or *reactive* planning sought to graft facilities for execution monitoring and error detection onto traditional, static-world planners (for example, see [25]). Error analysis and intelligent replanning techniques have been investigated within this framework [35][44], and the resulting planners are more powerful than their static-world counterparts. It has been protested, however, that this approach underestimates the complexity of the world; the protestors argue that intelligent behavior is mainly the result of dynamic interaction with the environment, and that deliberative planning is rarely if ever necessary. (For a good summary of this debate, see [55].) Indeed, some proponents of this view have built systems which interact with the world in seemingly complex ways, but which do not represent the world in the symbolic terms necessary for deliberation [8]. This paper describes a planner which, like some of its contemporaries, attempts to have it both ways (see also [3][5][20][23] and several papers in [31]). Our view is that planners must continually monitor and react to the world in an event-driven manner, but that they must also be capable of goal-directed reasoning as performed by traditional systems. The planning and reaction mechanisms should be smoothly integrated. The term "reactive planner" has been used for systems which meet these requirements.

In our system, reactive planning is achieved through the use of a multiplicity of reasoning/reacting processes which run in parallel (although the parallelism is simulated). This allows the creation of high-level reasoning processes which may be informed or interrupted by lower-level sensory processes. Several control, communication and representation problems are introduced, however, by this computational architecture. For example, which processes should be allowed to communicate with which others? How is competition for processing time to be arbitrated? How can the large amounts of knowledge, ranging from sensory reports to high-level

goals and plan-steps, be managed efficiently?

It is our contention that the answers to these questions lie in the hierarchical decomposition of planning and reacting knowledge (see [60] for a similar view). In order for such a decomposition to be effective, it must be based upon an understanding of the structure of the underlying planning knowledge, and upon an understanding of operational requirements of multi-level representational systems.

3. The HomeBot Domain

The difficulties of reactive planning appear only in complex domains. When a domain is small enough and well enough contained, one can anticipate nearly all eventualities; in this case even exponential searches may perform reasonably well. We have therefore chosen to work in a large and complex domain, which is nonetheless familiar enough to us as experimenters that we can easily introspect about human behavior within it. We feel that our HomeBot domain is somewhat more suitable for reactive planning research than others which have been proposed because of its size and complexity (compare, e.g. with [38]), and because of the structural richness of the real-world events which can occur (compare with [48]).

HomeBot is a simple one-armed, one-eyed robot which lives in a one bedroom apartment, and which is expected to perform household tasks such as cleaning and protecting the resident humans from dangerous conditions. The apartment consists of several rooms (bedroom, kitchen, living room, bathroom and closets), and is quantized into over 70,000 spatial locations. A large collection of objects (furniture, pieces of laundry, etc.) may be present in the apartment at any given time. In addition, there are several external occurrences which may be caused to transpire at any given time. For example, the doorbell may ring, an object can be moved, one of HomeBot's sensors might fail, etc. The layout of HomeBot's domain is shown in Figure 1. The representation of the apartment, the position of objects within it, etc., is entirely distinct from HomeBot's representation of the world. HomeBot knows only that which it can infer from the data it receives through its sensors.

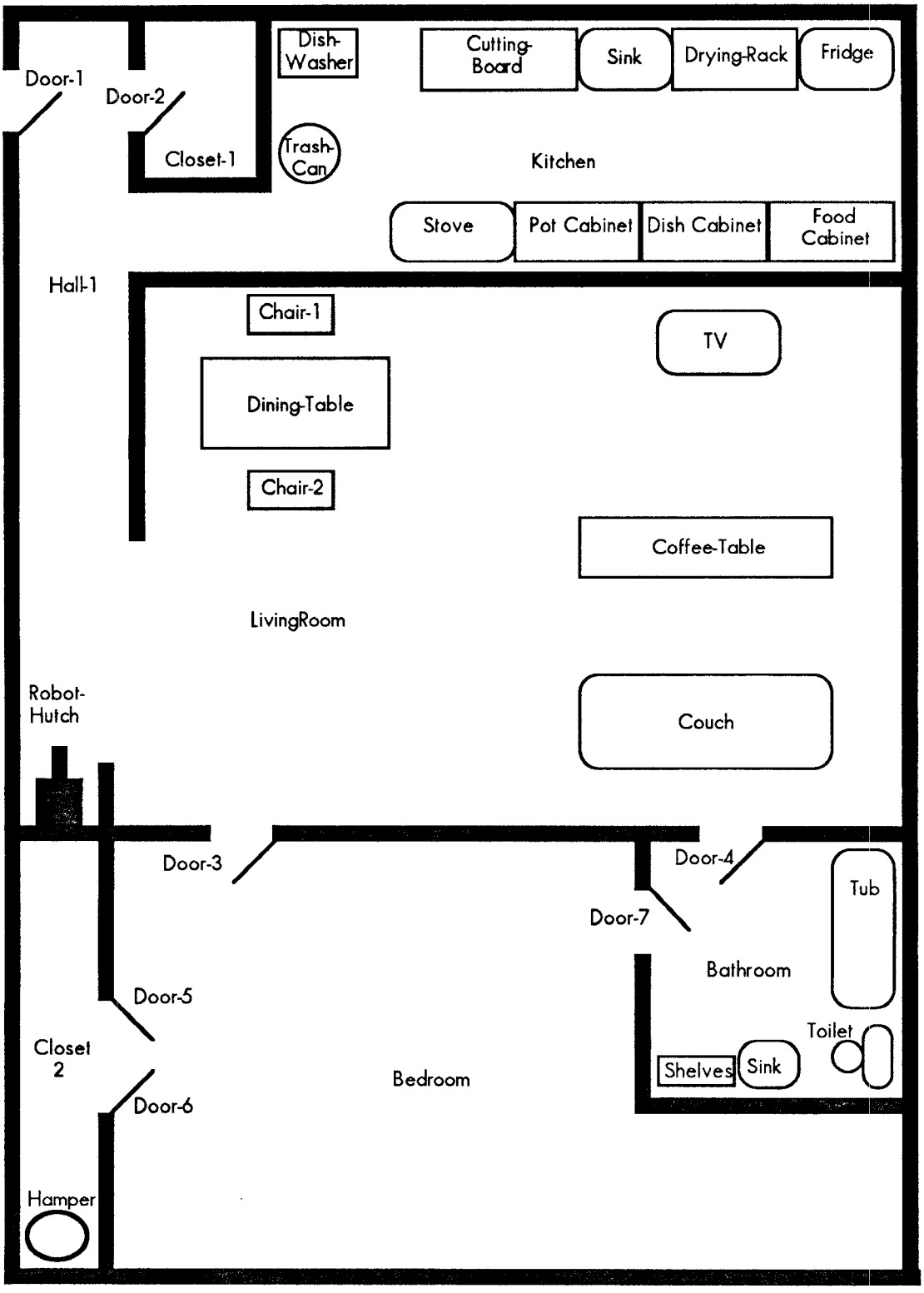


Figure 1: HomeBot's Domain

4. The Desired Behavior: The Banana Peel Problem

In our basic test scenario, HomeBot wanders around the apartment noticing objects that are out of place and putting them away. We can introduce simple complications such as a ringing doorbell (answering the doorbell is a high priority task that interrupts clean-up activity) in order to test the system's reactivity. Consider, however, what ought to happen when HomeBot, in the midst of some clean-up chore, notices a banana peel on the hallway floor. Assuming that there is no pre-wired response for such a situation (which there certainly should not be), HomeBot will have to do a fair amount of reasoning in order to determine that the peel is a safety hazard for passing humans. Meanwhile, the chore in progress must continue, since nothing would ever be accomplished if the robot merely stood still and contemplated the possible problems which might be caused by every object seen. The proper behavior is for HomeBot to continue working while reasoning about in the banana peel on the floor. Sensory-motor and navigational problems should be handled by the lowest levels of the hierarchy, leaving the higher levels free to consider the effects slippery objects being on the floor, the relative priorities of cleaning up and of removing safety hazards, the expectations about when humans will be present in the hallway, etc. Once such the proper inferences are made, HomeBot should suspend work on the current chore, and plan for the removal of the safety hazard. After disposing of the banana peel, HomeBot should resume the previous task, preferably with as little replanning as possible.

5. Architecture

5.1 A Multi-Level Parallel Blackboard System

The basic architectural model upon which our work is based is that of a multi-level blackboard system. "Blackboard System" refers to any system based on the model pioneered in the development of the Hearsay-II speech understanding system (see [17] or [18]), in which a multiplicity of independent procedural components[4] communicate solely via a shared data area called a "blackboard." Even in the earliest blackboard systems, provisions were made for blackboards which could be partitioned by "level of abstraction," and for systems in which procedural components were allowed to execute in parallel. More recently, several researchers have been exploring techniques for actually realizing the promise of parallelizability inherent in the

[4]These procedural components are often referred to as "Knowledge Sources" or "KSs."

blackboard model [4][9][11][33]. Others have been exploring the requirements of real-time AI systems, and the modifications of the blackboard model which may be necessary for their satisfaction [15][19][32][42][49]. We have incorporated many of the suggestions provided in these papers into our architecture, which is called APE (for Abstraction Partitioned Evaluator).

The APE architecture consists of a number of data/processing *levels* arranged as shown in figure 2. Each level can communicate only with those levels immediately above and below itself in the linear hierarchy. Although there would be little difficulty in extending the model to handle a branching, partially ordered hierarchy, we have not found it necessary to make such a generalization. Only the lowest level has direct access to sensors and effectors. The levels are permitted to run in parallel and to communicate asynchronously.

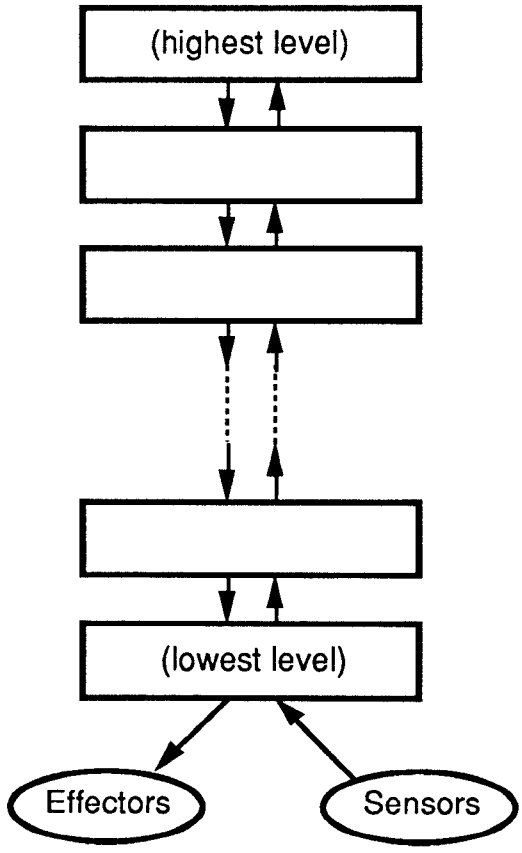


Figure 2. A coarse view of the APE architecture.

Figure 3 shows a single level of the system in greater detail. Each level contains both procedural and declarative components. The declarative knowledge at each level resides in a blackboard structure accessible to all procedures at that level. This shared knowledge forms a representation of the current state of the world as seen by the given level. Borrowing terminology from [28], we call this state-of-the-world-at-a-given-level a "State of Affairs" or SOA. The communication channels to adjacent levels allow for the transmission of two kinds of data: *Commands* are goal-achievement requests which may only be passed from higher to lower levels, and *World Updates* are modification patterns for the SOA at a given level. World Updates are propagated only from lower levels to higher levels.

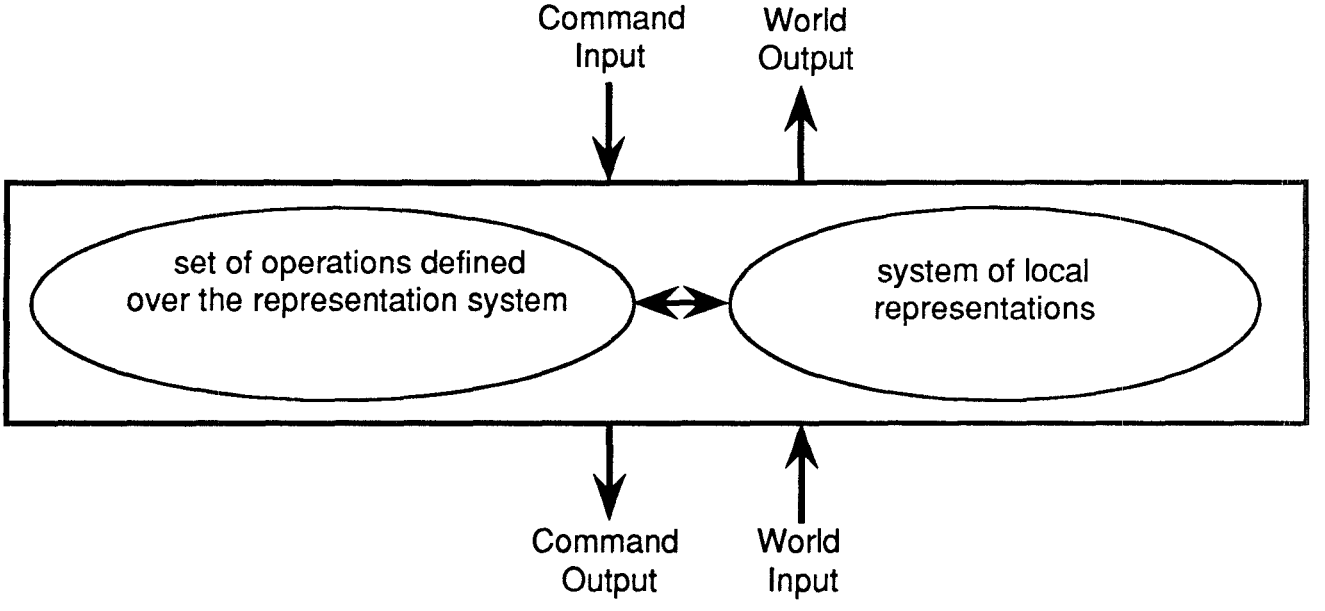


Figure 3. One level of the APE architecture.

Figure 4 adds additional detail to our picture of the individual levels of the APE architecture. The problem-solving (planning) procedures are located within the "Replanner" unit. Although these procedures should possess certain general properties (such as interruptability at some level of temporal granularity — see below) the APE architecture makes no stipulations about the reasoning strategies or micro-architectures within the planners. In fact, it is possible to use radically different planning methodologies at each level. For example, one might wish to use a connectionist planning system at the lowest level, logic programming techniques at intermediate levels, and a traditional nonlinear planner such as NONLIN at the highest levels. As long as all of the planners can interact with the blackboard mechanism using a uniform protocol, such a scheme would be in perfect accord with the APE architectural philosophy. For simplicity, our current implementation uses the

same reasoning mechanism (based on an extension of NOAH-style operators) at each level; however, there is nothing to prevent us from changing the procedures at any given level if this becomes desirable.

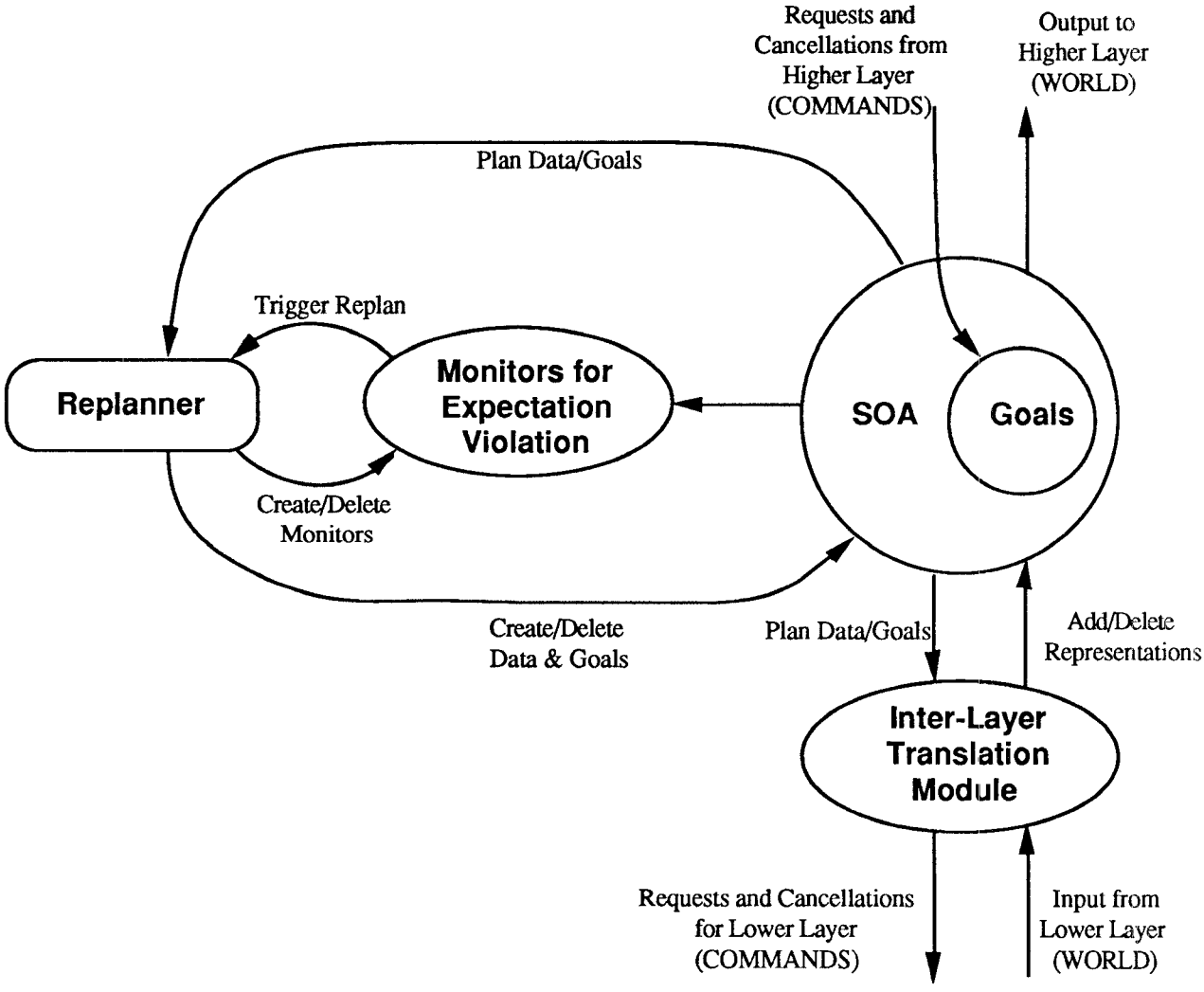


Figure 4. A detailed look at one level of the APE architecture.

Although figure 4 shows a unit containing "Monitors for Expectation Violation," it is not actually necessary that each level contains an explicit "monitor" data type. The unit is shown in the diagram in order to make it clear that the procedural components at each level have a responsibility to watch for changes in the SOA at regular time intervals and to be capable of responding appropriately. In our current implementation we *do* use explicit "monitors" to trigger interrupts to planning procedures. However, any functionally similar mechanism would be equally acceptable.

Since it is possible that the systems of representation employed in adjacent levels will differ considerably[5], it is necessary to provide some facility for translating Commands and World Updates from system to system. In our current implementation the Operators (see below) which are responsible for problem solving also handle inter-level communication and translation tasks. However, we view these translation processes as distinct from the problem-solving tasks of the planners, and we will separate the translation and problem-solving procedures in future versions of the system (see Future Work).

5.2 The Planner and Operators

As mentioned above, the APE architecture does not dictate the type of problem-solving mechanism to be used at each level. We have developed a planning system that uses complex operators, as well as specifications for procedural decomposition (similar to the specifications used in NOAH [51]), which has been used at all levels in our system. In this section we briefly describe this planning mechanism.

We have modified the operator formalism of traditional planners in several ways in order to make it suitable for our purposes. As in most modern planning systems, we allow the specification of information about the hierarchical decomposition of an operator into sub-goals. This information is specified in a procedural formalism (the **steps** of an operator, see below) which has been designed with our multi-level, reactive architecture in mind. Additional modifications, detailed below, allow the planning algorithm to cope with execution failures and with unexpected changes in the world. With the given modifications our operators resemble the *Knowledge Areas* of [22] and the *Reactive Action Packages* of [20] in several ways.

The role of operator "preconditions" is played in our system by a list of **filter** conditions, in conjunction with a partially ordered set of computational **steps**. Filter conditions are preconditions which cannot be achieved by the planner; if a filter condition is not true in a given SOA, then the operator to which it belongs will not even be considered. Such a filter mechanism may, depending on other characteristics of the implementation, have a dramatic impact on the system's efficiency.

The **steps** of an operator express the *achievable* preconditions, and, more generally, the *method*[6], of an operator in a procedural formalism. It is possible to express constraints on the

[5]Even if similar (or even identical) reasoning mechanisms and data formats are used at all levels, it is likely that at least the representational lexicon will differ from level to level.

[6]The **steps** specify *how* to do that which the operator was designed to do. In early operators such as those

order of achievement of APE operator steps, although the steps need only be partially ordered. Within a step it is possible to set and to access local variables, and to query and to modify the blackboard. Simple conditional and looping constructs are also provided (as in [16]). These enhancements help greatly in the specification of operators which achieve complex goals. Similar procedural mechanisms have been used in other planners (for example the SOUP code of [51] and the "plots" of [62]).

The **steps** of an operator also provide the limit to the granularity of potential parallelism in the APE architecture. Recall that the levels of an APE system may all run in parallel, communicating asynchronously. It is also permissible for all of the active operators on each level to run in parallel, communicating with the blackboard asynchronously. In addition, the steps of each operator can all run in parallel unless there is an ordering constraint in the operator which prohibits this. However, each step is an atomic operation; it is guaranteed that a step will not be interrupted. The specifications which are permitted as steps of an operator make up a small, simple parallel programming language. (We call this language Stepeese and we provide a description of its syntax in the Appendix.)

APE is designed to function in complex, unpredictable domains; hence it is natural to assume that APE operators will sometimes fail. Traditional planners provide no facility for describing the effects of operator failure. In APE, we provide two sets of add and delete lists; one to be used when an operator succeeds, and one to be used when an operator fails. We can use the failure lists to assist in recovering from unexpected occurrences, and in some cases to facilitate "meta-reasoning" about the efficacy of certain operators[7].

Whereas previous planners would instantiate operators on the basis of a match between pending goals and the add/delete lists, APE operators use a separate list, called **expect**, for this purpose. This is because an operator may be instantiated in order to bring about some condition which will be added to the knowledge base not by the given operator, but by a sub-goaling operator at a lower level of the hierarchy. For example, suppose that we have a high level operator called **put-away-object** which effects the overall behavior of getting an object to its proper place. Further, suppose that we have a lower level operator called **move-object** which handles the subtask of getting the object to its destination once that destination has been determined. The add list of the **put-away-object** operator contains a representation which indicates that the given object

used in STRIPS this information is to some extent in the preconditions, to some extent in the name of the operator (which is often a robotic control command) and to some extent absent.

[7]We discuss meta-reasoning more fully in the section "A Digression on Reflection," below.

is in its proper place — something like **(in-proper-place ?obj)**. A specific instance of this operator, instantiated in order to put away the object **dirty-sock-23**, adds **(in-proper-place dirty-sock-23)** to the blackboard upon successful completion. There are many additional representations which we could reasonably expect to be present after completion of **put-away-object**. If the proper place of **dirty-sock-23** is in the **hamper**, then we would expect **(in dirty-sock-23 hamper)** to be on the blackboard after the sock has been put away. However, there are reasons for making the **move-object** operator, and not the **put-away-object** operator, responsible for this latter addition. For example, we want **(in dirty-sock-23 hamper)** to be on the blackboard even in cases in which the **move-object** operation succeeds but the **put-away-object** fails. (Suppose that the robot cannot close the hamper lid, and that closing the lid is necessary for **put-away-object** to succeed.) The **"in"** form should still be included in the **put-away-object** operator since we can use **put-away-object** in order to change the location of an object, even when it is superfluous that the object is being moved to its *proper* place. For this reason, the **"in"** form should be put in the **expect** list, but not in the add list, of **put-away-object**.

Two additional components have been added to APE operators. The **monitors** component allows for the specification of simple if-then rules which run "continuously"[8] and which can trigger success, failure, suspension, or resetting of the current operator. Each monitor has an **antecedent** slot which contains an arbitrary LISP form which may include calls to routines which look for patterns on the blackboard. If the antecedent returns **true** then the **consequent** (which must be **succeed**, **fail**, **suspend**, or **replan**) is executed. The **consumes** component contains a list of resources (either computational or physical) required by the operator. (A similar mechanism is used in [62].) The consumes component is used in conjunction with goal priorities in order to suspend operators with conflicting resource requirements.

A synopsis of the components of an APE operator is provided in figure 5.

[8]Since parallelism in APE is currently simulated, monitors are run every time the given operator is given a time-slice for executing a step. One can also specify that a given monitor only be run every *n* time-slices allocated for the given operator.

OPERATOR
:name <a symbol>
:level <a level, e.g., CAUSAL>
:filters <predicates which must be true for instantiation>
:monitors <knowledge-base pattern-match requests which run constantly, and which trigger success, failure, or local variable re-assignment>
:steps <numbered computational steps, which may include knowledge-base pattern-match requests, goals for posting, and control constructs>
:step-seq <ordering constraints on the steps>
:succeed-add <forms to be added to the knowledge-base upon success>
:succeed-delete <forms to be deleted from the knowledge-base upon success>
:fail-add <forms to be added to the knowledge-base upon failure>
:fail-delete <forms to be deleted from the knowledge-base upon failure>
:expect <forms which are expected to be true upon success — used in picking operators for instantiation>
:consumes <a list of resources consumed by the execution of the operator>

Figure 5. The format of an APE operator.

The planning algorithm used in our current implementation is an "execute immediately" algorithm in the tradition of NASL [43]. In other words, primitive actions (sensor and motor commands) are executed as soon as their operators get processing time. While the full planning algorithm used in our current system is fairly complex, the overall control structure can be described simply as follows:

Continuously and in parallel:
- Instantiate any operators with expect conditions matching a goal.
- Suspend lower priority operators with conflicting resource requirements.
- Resume suspended operators if conflicts are gone.
- Run all monitors for all instantiated operators.
- For all instantiated operators execute all steps that are enabled according to the partial order constraints. Many of these steps may post "reduced" goals to the blackboard.
- For all operators which succeed or fail, perform appropriate additions and deletions to blackboard.
- Abort all operators working on goals already achieved.

The pseudocode for a typical operator from our implementation is shown in figure 6.

```
OPERATOR:
:name INFER-ON
:level SPATIAL
:filters (ISA !OBJECT MOVABLE-OBJECT)
:monitors (IF (MOVED !OBJECT) THEN REPLAN)
:steps   1:  ACHIEVE (KNOWN (COORDINATES !OBJECT ?OBJ-COORDS))
         2:  ACHIEVE (KNOWN (COORDINATES !SURFACE ?SURF-COORDS))
         3:  IF (JUST-ABOVE ?OBJ-COORDS ?SURF-COORDS)
             THEN SUCCEED
             ELSE FAIL
:step-seq STEP 1 BEFORE STEP 3, STEP 2 BEFORE STEP 3
:succeed-add (KNOWN (ON !OBJECT !SURFACE))
:succeed-delete (CLEAR-TOP !SURFACE)
:fail-add (NOT (KNOWN (ON !OBJECT !SURFACE))), (POSSIBLE (CLEAR-TOP !SURFACE))
:fail-delete (KNOWN (ON !OBJECT !SURFACE))
:expect (KNOWN (ON !OBJECT !SURFACE)), (KNOWN (COORDINATES !OBJECT ?OBJ-COORDS))
        (KNOWN (COORDINATES !SURFACE ?SURF-COORDS))
:consumes NIL
```

Figure 6. A pseudocoded APE operator.

APE's current planner is a powerful reasoning system which provides facilities for integrated deliberation and reaction. The operator/monitor formalism is still less elegant than we would like, however, and the coding of functional operators is currently a rather delicate craft. The refinement of this formalism is one of our research priorities (see Future Work, below).

5.3 Multiple Levels of Abstraction

The previous sections describe a computational architecture designed to operate in a planning domain in which the planning knowledge has been partitioned into levels. However, we have not yet discussed the principles by which such a partitioning can be accomplished, or the specific partitioning scheme which we have found to be useful.

The division of planning knowledge into manageable segments could be accomplished in many ways. For example, some of the early work on *scripts* can be seen as exploiting a partitioning of knowledge by the *situations* in which the knowledge might be useful [52]. In more recent work, Lansky's GEM model used physical location information to structure event-related knowledge and to guide the decomposition of planning tasks [40]. Lansky has also discussed the use of subdivisions of planning knowledge based on functional decomposition [41].

We are exploring the use of *level of abstraction* as the criterion by which to partition planning knowledge (both declarative and procedural). System decomposition by "level of abstraction" is not, by itself, a new idea. Hearsay-II, a piece of work mentioned earlier in this paper for its multi-level blackboard architecture, used multiple levels of abstraction which corresponded to "separable domains of reasoning" in the speech-understanding domain [18]. Other, more recent research has applied the idea of abstraction-partitioned representation directly to the problems of reactive planning [30][34][29]. However, one of the issues which has remained obscure is what people mean by "abstraction" when they refer to "levels of abstraction." The phrase has been used in various ways; "levels of description," "levels of competence," "levels of specificity," and similar expressions have been used, and are not always clearly explained. Within planning, the concept of abstraction has played many roles, from the early work on ABSTRIPS [50] to recent discussions such as [36]. The term "abstraction" seems to be, within Computer Science as a whole, so ambiguous as to be nearly devoid of meaning. The term "level" (or "layer") is similarly problematic. We wish to resolve this muddle by spelling out very clearly the connotations of "abstraction" and of "level" which are appropriate for the design of abstraction-partitioned reactive planners.

What characteristics should a level-based partitioning meet in order to make effective use of a multi-level architecture like APE? The first requirement, dictated by the APE architecture (and by most other multi-level architectures as well) is one of *information composition*. Since the higher levels can access the world only through the mediation of lower levels, the higher level representations must be *composable* from the lower level representations. When details of the lower level representations are omitted from the higher level composition[9], then "information composition" denotes a common use of the word "abstraction."

Another obviously desirable characteristic is that the *temporal granularity* of the reasoning processes should be finer as one moves down in the hierarchy [30]. Since only the lowest level has direct access to sensors and effectors, it is imperative that the lowest level processes be capable of responding quickly. The slowest procedures should be at the highest levels, since protracted uninterruptable computations at lower levels would probably diminish reactivity.

Since the procedures at higher levels will be given longer amounts of time for computation, it is reasonable to stipulate that this extra time be put to good use in providing *competences* not present at lower levels. As one moves up in the hierarchy the time/competence trade-off should

[9]For example, suppose that **(dinner-time)** is composed from **(on table1 dinner32)** and **(time 6:30PM)**.

increasingly be decided in favor of competence. It will further aid in the development of partitioned systems if the new competences provided by a given level form a coherent, natural class. Finally, if the parallelism in a multi-level architecture is to be fully exploited, it is necessary that the partitioning of knowledge allows for simultaneous reasoning at all levels of abstraction.

Is there a way to divide up planning knowledge into a set of partitions which has all of these characteristics? We believe that it is, and we summarize this view in the following hypothesis:

The Abstraction Separability Hypothesis (ASH): Planning knowledge (both declarative and procedural) can be neatly partitioned into a set of *abstraction levels* which meet the following criteria:

1: Information Composition — Each level constructs its representation of the world from the representations of lower levels, and only the lowest level is in direct contact with the real world (through sensors and effectors).

2: Temporal Granularity Stratification — Each level represents a less dynamic world model than is represented at lower levels; that is, lower levels must respond quickly, and higher levels may be more sluggish (allowing more time for deliberation).

3: Competence Stratification — Each level provides a well defined set of reasoning services not available at lower levels.

4: Level Parallelism — Processing can occur at all levels simultaneously and asynchronously.

Note that criterion 1 implies that each level has access to representations not available at lower levels, but not the converse; information composition does not necessarily imply information hiding. In hypothesizing that planning knowledge can be "neatly" partitioned into levels as described above, we mean that a system which partitions knowledge in this way will exhibit several desirable characteristics, including reactivity, parallelizability, and tractability. Of course, we have so far said nothing about any *particular* set of levels; we have only described the set of properties which such a set should have. Before moving on to the details, however, we will digress briefly to distinguish the kind of levels that we have described from *meta-levels,* which have been used in other AI systems, including reactive planners.

5.4 A Digression on Reflection

"Multi-level" is sometimes used in AI to mean "meta-level." Meta-reasoning, also called reflection, refers to a system's reasoning about *itself.* While meta-reasoning is a fascinating topic, with many possible applications and a host of interesting problems (for example, see [46]), it is not currently one of our central concerns. However, meta-reasoning *has* been discussed as a tool for

solving many of the same kinds of reactive planning problems as those that are addressed in this paper. It is possible for a system to reason about *the time that the system's own reasoning is taking,* and to make reasoning/reaction trade-offs on that basis (for example, see [37] or [61]). Further, some researchers have proposed multi-level architectures, similar to our's in many respects, in which each level reasons *about* the lower levels in the hierarchy [26][27][38][53].

There is little doubt that a meta-reasoning component could be of significant utility to a reactive planning system. For example, Kraus et. al. discuss a scenario in which Nell is tied to the railroad tracks as a train approaches [37]. (This scenario originated in [43]). Dudley's job is to rescue Nell, and if he is to be successful he must obviously complete all of his planning and execution tasks before the train runs her down. We agree that the only way that a system can perform reliably in such circumstances is for it to account explicitly for its own reasoning time, and for it to choose between reasoning and execution options on that basis.

There is nothing in the APE architecture which *prevents* its use in such circumstances as a *meta*-level architecture. Higher level representations are composed out of lower level representations, and they might also in some cases be *about* these lower-level representations. In addition, the procedural components on a given level are free to post declarative representations of their algorithms to the blackboard, thereby providing grist for the mills of higher-level meta-reasoning.

Nonetheless, we have decided to leave consideration of meta-level reasoning issues on the back burner for two reasons. First, our motivation for the use of a multi-level architecture springs from the complexity of representations of the world, and from the vast amounts of planning knowledge required for solving real-world problems. These are problems for *any* reactive planning system, whether or not it contains meta-reasoning capabilities, and whether or not it can solve the Nell and Dudley problem mentioned above. Since we feel that *some* significant reactive functionality can be achieved with little or no meta-reasoning, we wish to put aside the difficulties of meta-reasoning[10] at least for the time being. Second, we feel that a good reactive architecture obviates, to some extent, the need for meta-reasoning in the first place. If the lowest level and fastest reasoning mechanisms cause Dudley to run immediately to Nell's aid, and if the higher level reasoning mechanisms run in parallel and interrupt the rescue only if time permits, then Nell will probably survive. Of course, in certain circumstances this will be insufficient. For example, suppose that Dudley, using our non-reflective "run first" architecture, notices at the last moment

[10]Which we feel are considerable, despite recent advances.

that the rope holding Nell to the tracks is electrified, and that he must get rubber gloves (which are at the station, some distance from the tracks) before untying her. Due to the time he lost in running to the tracks before deliberating on the possibilities, he will probably fail in his mission. However, it is possible that Nell would have been saved by the following chain of meta-reasoning:

Time=0: I have about 60 seconds to save Nell before that train runs her down.
Time=1: It will take about 30 seconds to run and untie her, so I now have about 29 seconds to pursue other options.
Time=2: It will take about 10 seconds to consider if I should take anything from the train station with me.
Time=12: I should take the rubber gloves, since the villain Snidely has just taken an electronics course, and since getting the gloves will take only 10 seconds.
Time=22: I now have the gloves and only 8 seconds to spare, so I'll run and untie her.
Time=52: "Oh Dudley, you're my hero!" says Nell.

In the normal, non-electrified rope case, however, Dudley can succeed without recourse to meta-reasoning. Hence while meta-reasoning is necessary in certain reactive planning scenarios, it is reasonable to assume that rudimentary reactive capabilities are achievable without it. The representations at all levels of our system will generally be representations *of the world* at different levels of abstraction, *not* representations of the system itself.

6. Event Representation

6.1 Towards a Specific Set of Levels

The Abstraction Separability Hypothesis says nothing about any particular choice of levels; it says only that *some* level-based partitioning exists that meets criteria 1 through 4. The question of how such a partitioning is to be found is intriguing in its own right. One approach would be empirical; we could collect a large set of representations from our domain and try to divide them into appropriate classes by inspection. We have opted for an alternative approach, based on *a priori* structural properties of the knowledge to be represented.

The knowledge in a reactive planning system is primarily knowledge about *events* in the world and about *actions* that an agent can perform. The representation of events and actions is a topic with a considerable history both in philosophy [13] and more recently in AI (see [47] and several papers in [21]). We have developed a theory of event structure which is based upon the

"component" account of Thalberg [57], and which also borrows from the pioneering work of Goldman [24]. In the next several sections we will sketch this theory and its impact on the representation of knowledge in multi-level reactive planning systems.

6.2 The Problem of Action Representation

Consider the action of HomeBot putting the banana peel in the trash in the scenario which we discussed previously. All of the following statements about this action are true and informative:

S1: HomeBot protected a human.
HomeBot prevented a human from falling.
HomeBot prevented a human from slipping on the banana peel.
HomeBot moved the peel before the expected arrival time of humans.
HomeBot put the banana peel in the trash can.
HomeBot let go of the banana peel.

The prospect of representing all of this knowledge raises several questions. For example, philosophers interested in the theory of action have been concerned with determining, in cases such as this, how many actions are described by such a list of statements. Is there one action, described in many ways, or are there several distinct actions? If there are several actions, what are they and in what ways are they related? This is known as the problem of *action individuation,* which has been the subject of considerable controversy (see [13] for a brief summary)

Such questions raise a number of issues that will turn out to be important in the design of intelligent, reactive systems. We are not particularly interested in the counting of actions *per se*; it doesn't really matter for our purposes if one thinks of the above statements as describing 1, 6, or 1000 of HomeBot's actions. What does matter is that these statements describe a rich knowledge structure. We wish to formulate a representation system capable of capturing this knowledge structure in a natural, useful way.

6.3 Events, Actions and Parts

A first stab at representing the statements in S1 might run as follows:

R1: a1: **(protected HomeBot human)**
a2: **(prevented HomeBot (event (fall human)))**
a3: **(prevented HomeBot (event (slip human)))**
a4: **(before a5 (expected-time (event (return-home human))))**
a5: **(moved HomeBot banana-peel trash-can (time-interval t1 t2))**
a6: **(ungrasped HomeBot (time t2))**

There are several problems with this representation, but the principal difficulty is that the representations are not tied together in the proper way. For example, there is nothing to indicate that the human was protected *by* preventing the fall, or that the slip was prevented *by* moving the banana peel.[11] One solution, proposed in theory by Goldman [24] and used in the AI work of Allen [2] and Pollack [47] is to formalize the "by" locution (with a predicate that has been called GEN for historical reasons[12]) and to add assertions such as the following:

g1: **(GEN a2 a1)**
g2: **(GEN a5 a3)**

GEN is an asymmetric, irreflexive, transitive relation between events (including actions) that has been formalized and discussed extensively by Pollack [47]. While the GEN relation does allow the representation of event-structure which was previously unavailable to AI systems, we argue that it is an unnecessary addition to our representational ontology. The philosophical literature contains several objections to Goldman's "generation" theory, many of which need not concern us here. However, some of Goldman's detractors have proposed a rival account, the "component" account, which we consider to be better suited to our representational endeavors. Costa, commenting on the motivations for the for the component account, notes that "The notion of [generation] is, I am afraid, very obscure... One would like to be able to cash it out in terms of some already familiar metaphysical relations, but Goldman offers nothing very helpful here."[10]

The component account, championed primarily by Thalberg [57] and Thomson [58], replaces the obscure notion of generation with the very familiar "component" relation, a species of the "part/whole" relation. It remains to be seen if this move is compatible with all of the uses to which the GEN relation has been put[13], but we submit that the more recognizable territory of part/whole hierarchies is the proper framework within which to further study the interrelation of

[11]In the spirit of Davidson and others [11], we could claim that a1-a6 are actually all descriptions the same action, but we would then be left with the similar problem of how these descriptions ought to be interrelated.

[12]Goldman defined a relation between events called "level-generation." Subsequent commentators have dropped the "level" prefix and have sometimes abbreviated "generates" as "GEN." I will henceforth use "GEN" to describe the relation, even when discussing Goldman's views.

actions and events. Within this framework, our representation becomes:

```
a1:   (protected human)
a2:   (prevented (event (fall human)))
a3:   (prevented (event (slip human)))
a4:   (before a5 (expected-time (event (return-home human))))
a5:   (moved banana-peel trash-can (time-interval t1 t2))
a6:   (ungrasped HomeBot (time t2))
c1:   (components a1 (a2 a4))
c2:   (components a2 (a3))
c3:   (components a3 (a5))
c4:   (components a5 (a6))
```

This representation has the benefits of the GEN-based approach with at least two advantages: the novel and confusing concept of generation is no longer present, and inference techniques developed for use with part-whole relations can now be applied to event and action knowledge in a straightforward manner.

6.4 Levels of Organization

In his original exposition of GEN, Goldman distinguished four sub-species of the relation: causal generation, conventional generation, simple generation, and augmentation generation.

Similarly, Thalberg, in his component account, distinguishes several classes of potential components of a given event, including: "purely relational" consequences, causal consequences, and conventional consequences.

These analyses begin to reveal a structural property of events (or of sets of events, depending on how we decide questions of event individuation) which will turn out to be very useful in the design of intelligent, reactive systems. We express this property, and assert its applicability, with the following hypothesis:

---

[13]Note that the component relation is (like the generation relation) asymmetric, irreflexive and transitive [62]. We are currently preparing a more detailed formal comparison of GEN and the component relation.

The Knowledge Strata Hypothesis for Events (KSH):

1: The representations of the set of event-components of any given event[14] can be conveniently partitioned into the same small set of distinct classes.

2: These classes can be ordered {C0, C1, C2...,Cn}, such that for all i, the representations in class Ci are composed only of representations from the set of classes Cj with $j \leq i$.

Condition 1 derives from the insights in the analyses of both Thalberg and Goldman, although in Goldman's account it is the relations between events that are partitioned, rather than events themselves. Condition 2, which can be viewed as asserting that higher-level representations are "composed of" or "defined in terms of" lower level representations, is not stated explicitly in the previous literature, but it appears to follow from the set of event-classes that the literature leads us to posit. Within the context of a component account of events, condition 2 states that the classes form levels of a part/whole hierarchy, such that any given event is composed only of events at the same or lower levels.

The classes of consequences offered by Thalberg and the types of the GEN relation specified by Goldman have certain similarities. Both accounts make use of a category based on the existence of "conventions." In addition, both accounts have a special place for knowledge based in causality. Thalberg's "purely relational" consequences and Goldman's "simple generation" are also very close; Goldman says that "In simple generation the existence of certain circumstances, conjoined with the performance of A, ensures that the agent has performed A'."

The similarities between the theories of Goldman and Thalberg break down upon closer inspection (they are, after all, *rival* theories), and the affinities of these theories to the account we will now offer are not much deeper. Hence, although the particular decomposition of event-knowledge that we advocate was strongly influenced by these prior analyses, we will not spend too much more time on comparisons.

We view event-knowledge as partitioned into five, hierarchically related knowledge domains: (0) sensory-motor, (1) spatial, (2) temporal, (3) causal, and (4) conventional. We also refer to these domains as levels, where the conventional level is the highest and the sensory-motor level is the lowest.

Since knowledge is represented by use of symbolic predications such as those in the examples above, we can partition event-related knowledge by partitioning the set of predicates. Each predicate in our system is "defined" at some level in the hierarchy, and is available only at that

level and (possibly) at higher levels.

The sensory-motor level represents events as simple sensory reports and as operators for effector manipulation (such as **(ungrasp (time 3:45-PM))**). The only "reasoning" at this level is analogous to reflex arcs in animals. The spatial level contains structures which organize sensory data with respect to spatial relations (for example, **(moved banana-peel trashcan (time 3:43-PM 3:45-PM))**). Operators for complex spatial reasoning (such as path planning) reside here. The temporal level augments spatial representations with temporal relations, allowing for reasoning about deadlines and temporal projection. (This is where we would put **(before (event e5) (expected-time (event (return-home human))))**.) The causal level contains representations which embody the agent's conception of the causal structure of the world (including causal rules and causally deduced facts such as **(prevented (event (fall human)))** and **(prevented (event (slip human)))**). The conventional level contains knowledge about facts that are true by convention; for example, that a certain hand gesture is a signal for a turn, or that a dirty sock "belongs" in the dirty clothes hamper. We also put **(protected HomeBot human)** at the conventional level, since in our system the concept of "danger", and hence also that of "protection", is defined by convention.

In summary, then, here is the set of levels that emerges from our analysis of event structure:

Level 0: **Sensory/Motor** — Contains sensory reports and operators for effector manipulation. The only "reasoning" at this level is analogous to reflex arcs in animals.

Level 1: **Spatial** — Contains structures which organize sensory data with respect to spatial relations. Operators for complex spatial reasoning (such as path planning) reside here.

Level 2: **Temporal** — Augments spatial representations with temporal relations, allowing for reasoning about deadlines and temporal projection.

Level 3: **Causal** — Contains representations which embody the agent's conception of the causal structure of the world.

Level 4: **Conventional** — Contains knowledge about facts that are true by convention; for example, that a certain hand gesture is a signal for a turn, or that a dirty sock "belongs" in the dirty clothes hamper.

[14]Or "the set of descriptions of a given event" or "the set of events GEN-related to a given event," etc.

The following is a list of some of the major predicates defined at each level:

conventional: **protected, improper**
causal: **causes, prevented**
temporal: **before, after, during**
spatial: **above, below, in, on, moved**
sensory-motor: **sense, control**

7. Architecture + Representation: Putting it Together

The Knowledge Strata Hypothesis is a philosophically motivated thesis about the representation of events. The Abstraction Separability Hypothesis is in implementationally motivated thesis about knowledge representation for reactive planners. Our five-level representation scheme was developed in order to satisfy the conditions of both.

Our system may be thought of as maintaining five different models (or "states") of the world, one for each level. Facts about the world are generally distributed across multiple levels. For example, the fact that **dirty-sock-23** is not where it ought to be includes a representation at the conventional level **(out-of-place dirty-sock-23)**, several representations at the spatial level (concerning the current location of the sock, the locations that would be acceptable, and the relations between the two), and sensory/motor representations about the sighting of the sock, etc. Descriptions of actions and events are similarly distributed. For example, the conventional level **put-away-object** contains spatial level components (such as **compute-path** and **go-to-place**), sensory-motor level components (such as **move-forward** and **grasp**) and so forth. The resulting part/whole action hierarchy can be used in very much the same ways as the ad hoc task decomposition networks were used in previous planning research. However, since the hierarchy is stratified into well-motivated partitions, we can also take advantage of the benefits offered by multi-level architectures.

8. HomeBot In Action

Our HomeBot system, which has been built on the basis of the principles discussed above, has reached a moderately functional stage of development and is capable of solving the "Banana Peel Problem" mentioned earlier in this paper. A detailed description of its behavior in this scenario

would be very long (and somewhat tedious); we will, however, attempt to convey the highlights of the system's operation in a brief and summary fashion.

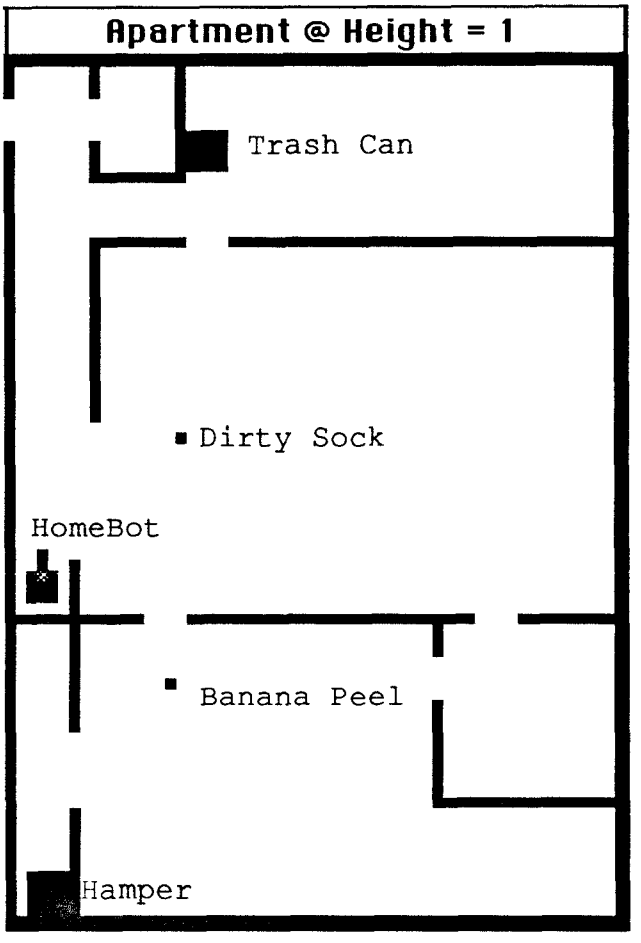


Figure 7. A snapshot of the initial state in the banana peel scenario (irrelevant objects have been suppressed for clarity).

We start the system with HomeBot in its "robot hutch" with the conventional level goal of earning praise, and we place a dirty sock in a spot where it is likely to be found, and a banana peel on the floor on the way to the hamper (see figure 7). The goal of earning praise is decomposed into the conventional level actions of finding and rectifying improper conditions, which in turn further decompose into spatial and sensory/motor level actions involved in seeking an out-of-place object (see figure 8). HomeBot's wanderings soon lead it to the discovery of the dirty sock on the living-room floor. The representation of this dirty sock propagates up the hierarchy to the conventional level where it becomes, by virtue of conventional level knowledge about the proper places for dirty socks, an **out-of-place-object**. Having found an object to be put away, HomeBot

proceeds to decompose the put-away task into its components (see figure 9). Most of these are at the spatial level (path planning) or at the sensory/motor level (moving, grasping, sensing). One subtask, that of discovering where the sock should be dropped such that it will fall into the hamper, is at the causal level.

After HomeBot has successfully piloted its way to the sock, grasped it, planned its path to the hamper, and begun to follow that path, it will notice the presence of the banana peel on the floor. HomeBot will initially continue on its way, but the representation of the banana peel will soon propagate up to the causal level where it will be recognized as a human-threat through a simple chain of causal inferences[15]. A newly instantiated rectify-improper-condition operator will post temporal level goals concerning the expected arrival-time of humans, and concerning the time it will take to complete the sock-moving task. Based on this temporal information, the rectify-improper-condition operator will suspend work on the sock-moving chore and begin work on putting away the banana peel (see figure 10). Work on the goal of putting away the banana peel proceeds similarly to that of putting away the dirty sock. When the banana peel is in the trash can, HomeBot will plan a path back to the dirty sock and then resume the task of putting the sock in the hamper.

[15]Our causal reasoning mechanism is a simple back-chainer which, in this case, reasons roughly as follows: the banana peel is a slippery object and therefore a potential slip-hazard; the potential slip-hazard is on the floor, and it is projected that a human will be on the floor in the future, therefore project a potential human-fall in the future; a human-fall is a human threat, therefore the banana peel on the floor is a human threat.

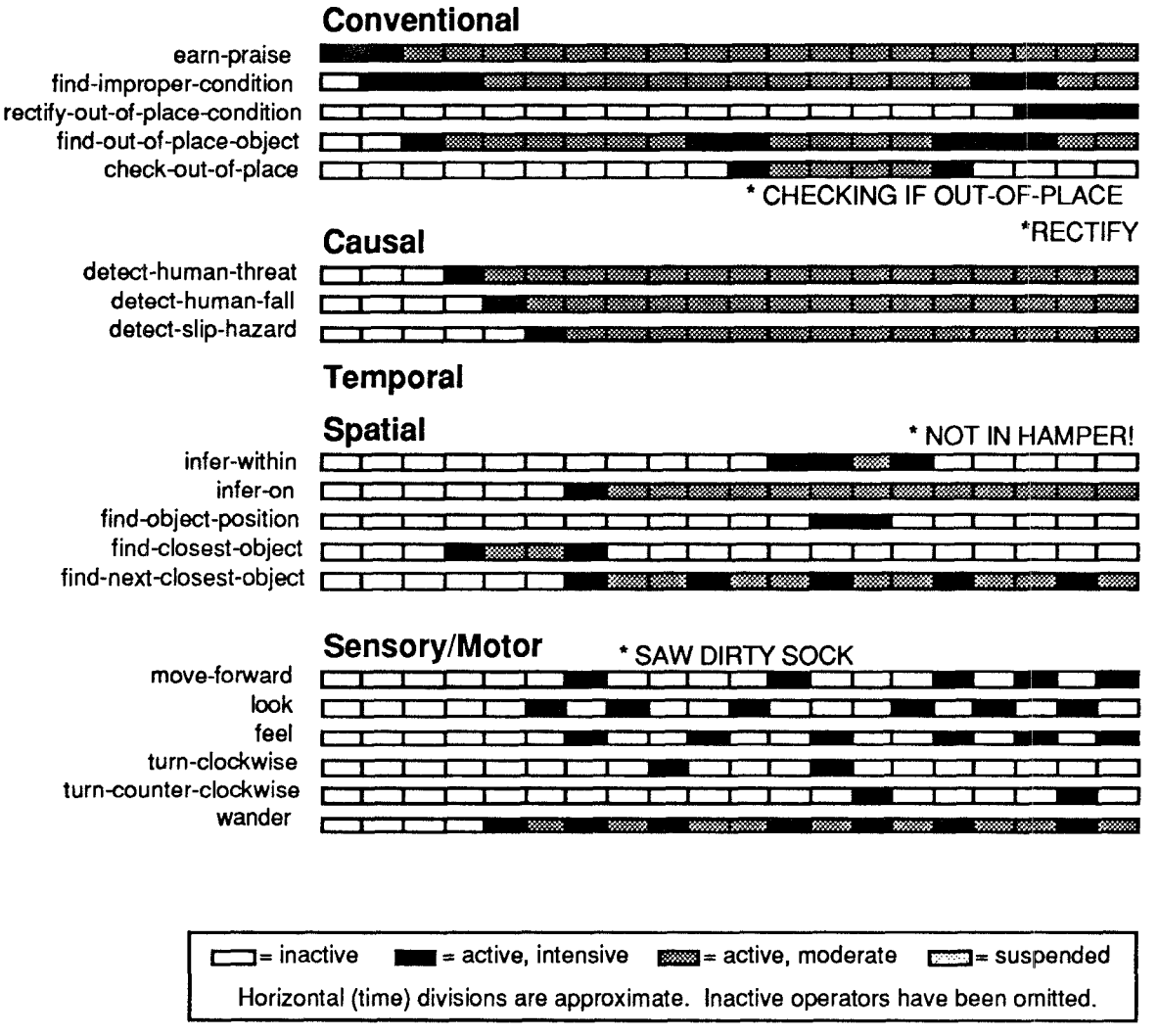


Figure 8. An early stage of processing in the banana peel scenario.

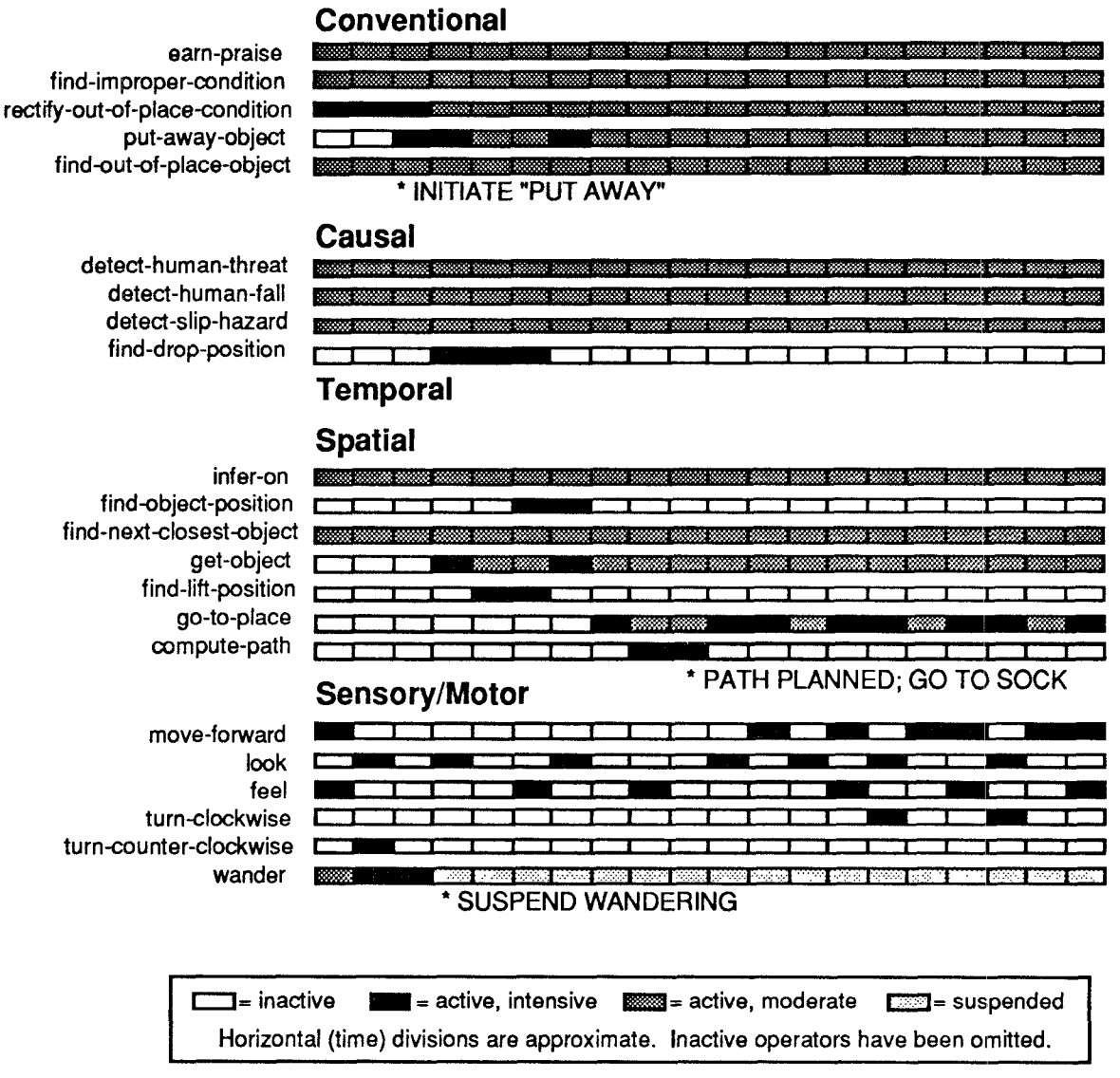


Figure 9. An intermediate stage of processing in the banana peel scenario.

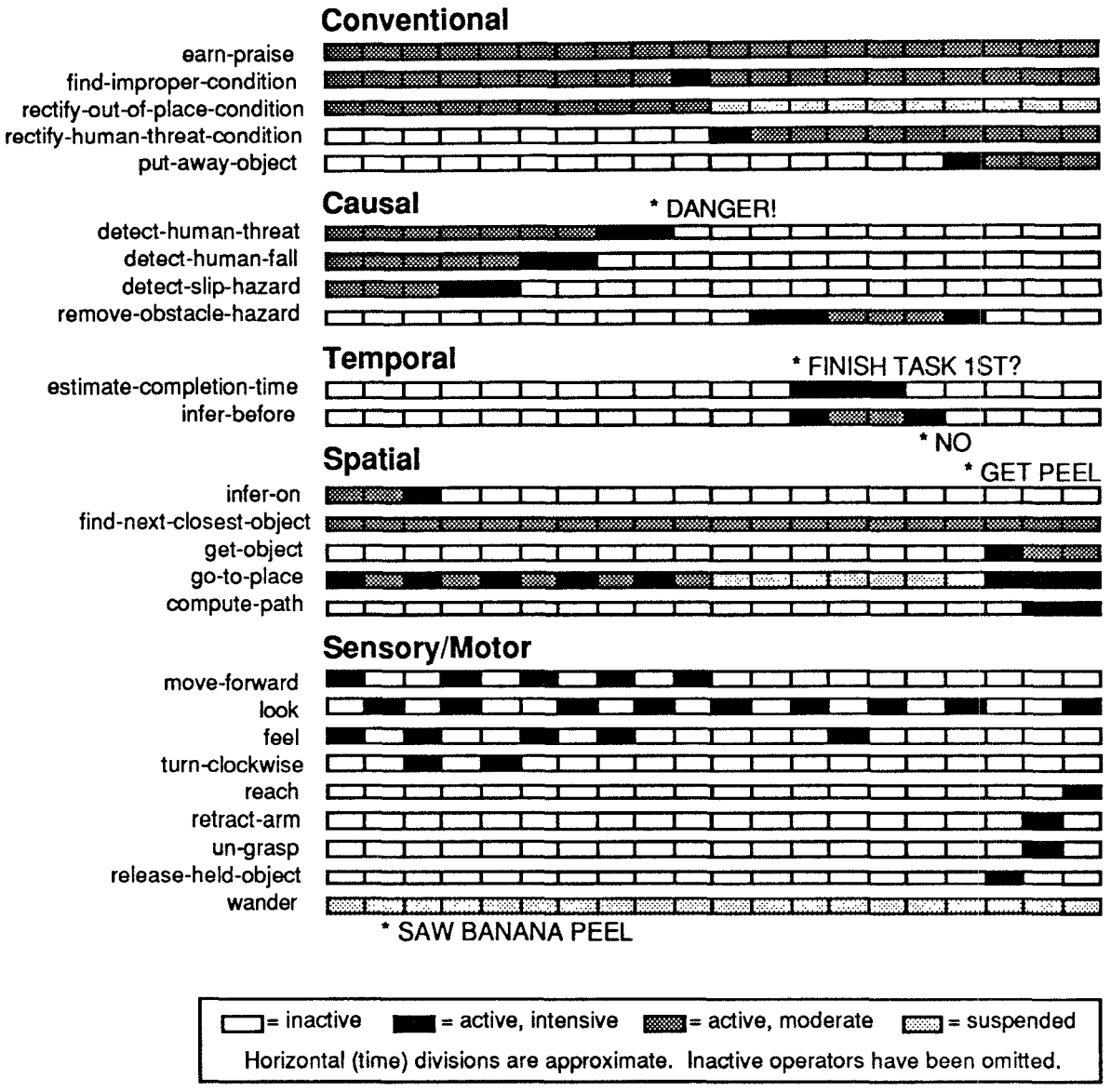


Figure 10. HomeBot decides to remove the hazard.

9. Future Work

The project described in this paper is incomplete in three respects:

1: The event representation work suggests a cleaner distinction between within-level problem-solving procedures and inter-level translation procedures than we have implemented. We propose to reformulate our system to be in stricter accordance with architectural ideas diagrammed in figure 4; that is, we wish to separate the maintenance of the event-representation hierarchy from

the within-level manipulation of event structures. Recall that our event-representation structure forms a part/whole hierarchy. We call the combination of this hierarchy, together with the procedures which propagate (translate) representations from level to level, an *inferential part/whole hierarchy*. It is our contention that inferential part/whole hierarchies are interesting in their own right, and also that the separation of hierarchy maintenance from within-level reasoning will greatly improve the clarity of our model as well as that of systems implemented within it.

2: Some of our representational mechanisms are still awkward and/or unnecessarily complex. In particular, the specifications of operators and monitors should be simpler and more concise. APE operators are similar to those of many other advanced planning systems, and they have thus far proven to be functionally adequate. Nonetheless, the construction of a functional set of operators is still too delicate a craft. With further study of the purposes for which operators are used, we hope to reduce the specification of advanced planning operators to a more elegant formalism. (The formalism of *traditional* planning operators is already rather elegant, but it is also weak.)

3. We wish to make good the promise of parallelizability inherent in the APE architecture. Our current implementation merely *simulates* parallel execution, but we are currently exploring options for real parallel implementation. We have recently obtained a parallel LISP system called *Top Level Common Lisp* which uses a model of parallelism based on "futures objects" [59]. We are planning to our next implementation on top of this system in order to assess the degree of speedup that is actually achievable.

The improvements indicated in items 1 and 2 are mutually supporting. As the translation tasks and the problem-solving tasks become more cleanly distinguished, it should become easier to simplify the specification of operators (which will then be responsible only for problem-solving tasks). Conversely, as the specification for operators is streamlined, the interface between the operators and the blackboard should also be simplified, and hence the functional requirements of the inferential part/whole hierarchy should come into sharper focus. Item 3, the use of real parallelism, also interacts with the redesign of the planning operators insofar as the **steps** of an operator are the limiting factor in the granularity of potential parallelism. In addition, the separation of translation from problem-solving tasks should provide further opportunities for the utilization of real parallelism.

The true test of the improvements which we have described is the ease with which we will be able to construct a reactive planner, and the class of problems that such a planner will be capable

of solving. In order to assess our progress we will run our planner in the HomeBot domain with scenarios which are progressively more complex. In addition to the possible presence of the banana peels and the like (which will not be expected by HomeBot) we will introduce a number of other complications: objects that are moved; a VCR (an old model, without a timer) which must be turned on and off at specified times; a dishwasher buzzer that signals that the dishes have been cleaned and are ready to be put away; a sink that blocks up (requiring that it be unclogged, but more importantly, requiring that the spigot be turned off and that the puddle on the floor be mopped up); and a dog with a tendency to bark loudly and to disturb the neighbors (it must be pacified with milkbones). When HomeBot is capable of functioning reasonably in such scenarios it will be clear that we have succeeded in extending significantly the state of the art in reactive planning.

10. Summary

This paper has examined the application of multi-level architectures to the problem of reactive planning. We have analyzed several criteria by which planning knowledge could be subdivided into levels, and have developed and implemented a computational architecture (APE) based on the most suitable of those criteria. Our analysis included a discussion of the functional requirements for multi-level representation schemes, as well as a discussion of the multi-level structure of event and action knowledge. Our theories have been applied within a large, complex and dynamic domain (HomeBot). Although our current system is capable of solving a large class of reactive planning problems, we have outlined several improvements which will make our method more comprehensible, and our implementation faster and more robust.

Appendix: The Stepeese Language

The specifications which are permitted in the **steps** slot of an APE operator make up a small, simple parallel programming language called Stepeese. The following is a brief description of this language using an extended BNF notation. Nonterminals are in <angle brackets>, terminal symbols are in **bold type**, and optional elements are enclosed within [square brackets].

```
<steps-spec>          ::=  (<step-list>)
<step-list>           ::=  <step> | <step> <step-list>
<step>                ::=  (step-number step-body)
<step-number>         ::=  <integer>
<step-body>           ::=  <achieve-body>|<re-achieve-body>|<post-goal-body>|
                           <if-body>|<do-again-body>|<set-local-body>|
                           <do-world-sim-body>|<delete-all-body>|
                           <succeed-body>|<fail-body>
<achieve-body>        ::=  (achieve <goal-form> [:level <level>])
<re-achieve-body>     ::=  (re-achieve <goal-form> [:level <level>])
<post-goal-body>      ::=  (post-goal <goal-form> [:level <level>])
<if-body>             ::=  (if <boolean-expression> <step-body> [<step-body>])
<do-again-body>       ::=  (do-again <step-number-list>)
<set-local-body>      ::=  (set-local <local-variable> <lisp-expression>)
<do-world-sim-body>   ::=  (do-world-sim <arbitrary motor command sequence>)
<delete-all-body>     ::=  (delete-all <blackbrd-pattern> [:level <level>])
<succeed-body>        ::=  (succeed)
<fail-body>           ::=  (fail)
<goal-form>           ::=  <arbitrary lisp form with local variables>
<blackbrd-pattern>    ::=  <arbitrary lisp form with local variables>
<level>               ::=  sensory-motor | spatial | temporal | causal |
                           conventional
<step-number-list>    ::=  <step-number> | <step-number> <step-number-list>
<local-variable>      ::=  <read-only-ref> | <settable-ref>
<read-only-ref>       ::=  !<lisp-identifier>
<settable-ref>        ::=  ?<lisp-identifier>
```

Notes: The **step-seq** slot of every operator contains a list of step-number pairs. If a pair *(m n)* is in **step-seq** then step *n* cannot execute until step *m* has completed. Otherwise steps *m* and *n* may execute in parallel or in any order. An **achieve** step posts a goal to the blackboard and does not complete until that goal has been achieved or until an appropriate **failed-to-achieve** form appears on the blackboard. Note that other steps may continue to execute while an **achieve** step is waiting. A **re-achieve** step first deletes any blackboard elements that match the goal and then functions as an **achieve** step. A **post-goal** step posts a goal to the blackboard and then completes immediately, without waiting. An **if** step evaluates an arbitrary boolean expression; if the expression returns **true** then the first following step-body is executed. Otherwise the second following step-body (if present) is executed. Nested **if** steps are not currently allowed. A **do-again** step marks as "uncompleted" the steps indicated by the list of step-numbers. These steps may therefore be executed again (if the operator does not terminate before they get the chance), and steps constrained to run after these steps will be suspended. A **set-local** step assigns the value of an arbitrary lisp expression to a local variable. A **do-world-sim** step executes an arbitrary lisp expression; the intent here is to provide the interface to robotic controllers. A **delete-all** step removes from the blackboard all entries that match the given pattern. A **succeed** step terminates execution of the operator and causes the operator's **succeed-add** and **succeed-delete** lists to be processed. A **fail** step terminates execution of the operator and causes the operator's **fail-add** and

fail-delete lists to be processed.

Bibliography

[1] Albus, James S., Barbera, Anthony J., and Nagel, Roger N., "Theory and Practice of Hierarchical Control," in *The Proceedings of the Twenty-Third IEEE Computer Society International Conference — Productivity: An Urgent Priority,* 1981.

[2] Allen, James F. "Towards a General Theory of Action and Time," in *Artificial Intelligence* **23** (1984), pp. 123-154.

[3] Ambros-Ingerson, Jose A., and Steel, Sam, "Integrating Planning, Execution, and Monitoring," in *The Proceedings of AAAI-88,* 1988.

[4] Bisiani, Roberto, and Forin, A. "Parallelization of Blackboard Architectures and the Agora System," in *Blackboard Architectures and Applications,* V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.

[5] Bresina, John, and Drummond, Mark, "Integrating Planning and Reaction: A Preliminary Report," in *Planning in Uncertain, Unpredictable, or Changing Environments,* James Hendler, Editor, University of Maryland Systems Research Center Technical Report SRC TR 90-45, 1990.

[6] Brooks, R. "A Robust Layered Control System for a Mobile Robot," in *IEEE Journal of Robotics and Automation,* Vol. RA-2, No. 1, 1985.

[7] Chapman, David "Planning for Conjunctive Goals," in *Artificial Intelligence* Volume 32, 1987.

[8] Chapman, David, and Agre, Philip "Abstract Reasoning as Emergent from Concrete Activity," in M. Georgeff and A Lansky, Editors, *The 1986 Workshop on Reasoning About Actions and Plans,* Morgan Kaufman, 1986.

[9] Corkill, Daniel D. "Design Alternatives for Parallel and Distributed Blackboard Systems," in *Blackboard Architectures and Applications,* V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.

[10] Costa, Michael J. "Causal Theories of Action," in *Canadian Journal of Philosophy,* Volume 17, Number 4, December 1987, pp. 831-854.

[11] Craig, Iain D. *The Cassandra Acrhitecture: Distributed Control in a Blackboard System,* Ellis Horwood Limited, Publishers, Chinchester, England, 1989.

[12] Davidson, Donald, "The Individuation of Events," in Davidson, Donald, *Essays on Actions and Events,* Oxford University Press, 1980.

[13] Davis, Lawrence, *Theory of Action,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

[14] Dean, Thomas, Basye, Kenneth, Chekaluk, Robert, Seungseok, Hyun, Lejter, Moises, Randazza, Margaret, "Coping with Uncertainty in a Control System for Navigation and Exploration," in *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90,* MIT Press, Cambridge, Mass., 1990.

[15] Dodhiawala, Rajendra T., Sridharan, N. S., and Pickering, Cynthia, "A Real-Time Blackboard Architecture," in *Blackboard Architectures and Applications,* V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.

[16] Drummond, Mark E., "Refining and Extending the Procedural Net," in *The Proceedings of IJCAI-85,* and reprinted in *Readings in Planning,* James Allen, James Hendler, and Austin Tate, Editors, Morgan Kaufmann Publishers, Inc., 1990.

[17] Erman, Lee D., and Lesser, Victor R., "A Multi-level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge," in *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75,* 1975.

[18] Erman, Lee D., Hayes-Roth, Frederick, Lesser, Victor R., and Reddy, Raj D. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," in *Computing Surveys,* Vol. 12, No. 2, June 1980.

[19] Fehling, Michael R., Altman, Art M., and Wilber., B. Michael "The Heuristic Control Virtual Machine: An Implementation of the Schemer Computational Model of Reflective, Real-Time Problem-Solving," in *Blackboard Architectures and Applications,* V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.

[20] Firby, James R. "Adaptive Execution in Complex Dynamic Worlds," Doctoral Dissertation, Department of

Computer Science, Yale University, 1989.
[21] Georgeff, M. and Lansky, A., Editors, *The 1986 Workshop on Reasoning About Actions and Plans*, Morgan Kaufman, 1986.
[22] Georgeff, Michael P., and Lansky, Amy L. "Reactive Reasoning and Planning," AAAI-87.
[23] Georgeff, Michael P., and Ingrand, François Felix "Decision-Making in an Embedded Reasoning System," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.
[24] Goldman, Alvin, I., *A Theory of Human Action*, Princeton University, Princeton, 1970.
[25] Hayes, P.J. "A Representation for Robot Plans," *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75*, 1975.
[26] Hayes-Roth, Barbara, "A Blackboard Architecture for Control," in *Artificial Intelligence* **26**, pp. 251-321, 1985.
[27] Hayes-Roth, Barbara, "Dynamic Control Planning in Intelligent Agents," in *Planning in Uncertain, Unpredictable, or Changing Environments*, James Hendler, Editor, University of Maryland Systems Research Center Technical Report SRC TR 90-45, 1990.
[28] Hendler, James A., and Sanborn, James "A Model of Reaction for Planning in Complex Environments," In *Proceedings of the Knowledge-Based Planning Workshop*, DARPA, Autin, TX, December, 1987.
[29] Hendler, James, and Subrahmanian, V.S., "A Formal Model of Abstraction for Planning", University of Maryland Technical Report UMIACSWW-TR-90-75 and CS-TR-2480, May 1990.
[30] Hendler, James, "Abstraction and Reaction," in *Planning in Uncertain, Unpredictable, or Changing Environments*, James Hendler, Editor, University of Maryland Systems Research Center Technical Report SRC TR 90-45, 1990.
[31] Hendler, James, Editor,*Planning in Uncertain, Unpredictable, or Changing Environments*, University of Maryland Systems Research Center Technical Report SRC TR 90-45, 1990.
[32] Hewett, Micheal, and Hayes-Roth, Barbara "Real-Time I/O in Knowledge-Based Systems," in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.
[33] Jagannathan, Vasudevan, "Realizing the Concurrent Blackboard Model," in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.
[34] Kaelbling, Leslie Pack "An Architecture for Intelligent Reactive Systems," in M. Georgeff and A Lansky, Editors, *The 1986 Workshop on Reasoning About Actions and Plans*, Morgan Kaufman, 1986.
[35] Kambhampati, S. "Flexible Reuse and Modification in Hierarchical Planning: A Validation Sturcture Based Approach," Doctoral dissertation, Department of Computer Science, University of Maryland, 1989.
[36] Korf, R. E. "Planning as Search: A Quantitative Approach," in *Artificial Intelligence* 22(1), 1987.
[37] Kraus, Sarit, Nirkhe, Madhura, and Perlis, Donald "Toward Fully Deadline-Coupled Planning," University of Maryland Technical Report, 1990.
[38] Kuokka, Daniel R., "The Deliberative Integration of Planning, Execution, and Learning," Carnegie Mellon University School of Computer Science. Technical Report CMU-CS-90-135, May 1990.
[39] Laird, John E., and Rosenbloom, Paul S., "Integrating Execution, Planning, and Learning in Soar for External Environments," in *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90*, MIT Press, Cambridge, Mass., 1990.
[40] Lansky, Amy L. "A Representation of Parallel Activity Based on Events, Structure, and Causality," in M. Georgeff and A Lansky, Editors, *The 1986 Workshop on Reasoning About Actions and Plans*, Morgan Kaufman, 1986.
[41] Lansky, Amy L., "Localized Representation and Planning," in *Readings in Planning*, James Allen, James Hendler, and Austin Tate, Editors, Morgan Kaufmann Publishers, Inc., 1990.
[42] Lesser, Victor R., Pavlin, Jasmina, and Durfee, Edmund H., "Approximate Processing in Real-Time Problem Solving," in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.
[43] McDermott, D., "Planning and Acting," in *Cognitive Science*, 2:71-109, 1978.
[44] Morgenstern, L. "Replanning," *Proceedings DARPA Knowledge-Based Planning Workshop*, Austin, TX, 1987.
[45] Nilsson, Nils J. *Principles of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1980.
[46] Perlis, Donald "Meta in Logic," in *Meta-Level Architectures and Reflection*, P. Maes and D. Nardi, Editors, Elsevier Science Publishers B.V. (North-Holland), 1988.

[47] Pollack, Martha E., "Inferring Domain Plans in Question-Answering," SRI Technical Note #403, SRI International, Menlo Park, CA, 1986.
[48] Pollack, Martha E., and Ringuette, Marc,"Introducing the Tileworld: Experimentally Evaluating Agent Architectures," in *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90,* MIT Press, Cambridge, Mass., 1990.
[49] Raulefs, Peter "Toward a Blackboard Architecture for Real-Time Interactions with Dynamic Systems," in *Blackboard Architectures and Applications,* V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, Editors, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, New York, 1989.
[50] Sacerdoti, E. D. "Planning in a Hierarchy of Abstraction Spaces," in *Artificial Intelligence,* 5(2), 1974.
[51] Sacerdoti, Earl D. "The Nonlinear Nature of Plans," in *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75,* 1975.
[52] Schank, Roger C., *Dynamic Memory,* Cambridge University Press, 1982.
[53] Schoppers, Marcel, and Linden, Ted, "The Dimensions of Knowledge Based Control Systems and the Significance of Metalevels," in *Planning in Uncertain, Unpredictable, or Changing Environments,* James Hendler, Editor, University of Maryland Systems Research Center Technical Report SRC TR 90-45, 1990.
[54] Spector, Lee, and Hendler, James A., "An Abstraction-Partitioned Model for Reactive Planning," in *Proceedings of the Fifth Rocky Mountain Conference on Artificial Intelligence (RMCAI-90),* New Mexico State University, Las Cruces, New Mexico, 1990.
[55] Swartout, William, (Editor) "DARPA Santa Cruz Workshop on Planning," in *AI Magazine,* Vol. 9, No. 2, Summer 1988.
[56] Tate, Austin "Project Planning Using a Hierarchic Non-Linear Planner," Department of Artificial Intelligence Research Report No. 25, University of Edinburgh, Edinburgh, August 1976.
[57] Thalberg, Irving, *Perception, Emotion & Action,*Yale University Press, New Haven, 1977.
[58] Thomson, Judith Jarvis, *Acts and Other Events,* Cornell University Press, Ithica, New York, 1977.
[59] *Top Level Common Lisp Reference Manual,* Top Level, Inc., Amherst, Massachusetts, 1990.
[60] Washington, Richard, and Hayes-Roth, Barbara, "Abstraction Planning in Real-Time," in *Planning in Uncertain, Unpredictable, or Changing Environments,* James Hendler, Editor, University of Maryland Systems Research Center Technical Report SRC TR 90-45, 1990.
[61] Wilensky, Robert, *Planning and Understanding,* Addison-Wesley Publishing Company, 1983.
[62] Wilkins, David E. *Practical Planning: Extending the Classical AI Planning Paradigm,* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.
[63] Winston, Morton E., Chaffin, Roger, and Herrmann, Douglas, "A Taxonomy of Part-Whole Relations," in *Cognitive Science* **11**, 1987, pp. 417-444.