A Blackboard Architecture for Control

Barbara Hayes-Roth
Heuristic Programming Project, Stanford University, Stanford, CA 94305, U.S.A.

ABSTRACT

The control problem—which of its potential actions should an AI system perform at each point in the problem-solving process?—is fundamental to all cognitive processes. This paper proposes eight behavioral goals for intelligent control and a 'blackboard control architecture' to achieve them. The architecture distinguishes domain and control problems, knowledge, and solutions. It enables AI systems to operate upon their own knowledge and behavior and to adapt to unanticipated problem-solving situations. The paper shows how OPM, a blackboard control system for multiple-task planning, exploits these capabilities. It also shows how the architecture would replicate the control behavior of HEARSAY-II and HASP. The paper contrasts the blackboard control architecture with three alternatives and shows how it continues an evolutionary progression of control architectures. The paper concludes with a summary of the blackboard control architecture's strengths and weaknesses.

1. The Control Problem

In attempting to solve a domain problem, an AI system performs a series of problem-solving actions. Each action is triggered by data or previously generated solution elements, applies some knowledge source from the problem domain, and generates or modifies a solution element. At each point in the problem-solving process, several such actions may be possible. The control problem is: *which of its potential actions should an AI system perform at each point in the problem-solving process*?

The control problem is fundamental to all cognitive processes and intelligent systems. In solving the control problem, a system decides, either implicitly or explicitly, what problems it will attempt to solve, what knowledge it will bring to bear, and what problem-solving methods and strategies it will apply. It decides how it will evaluate alternative problem solutions, how it will know when specific problems are solved, and under what circumstances it will interrupt its attention to selected problems or sub-problems. Thus, *in solving the control problem, a system determines its own cognitive behavior.*

Artificial Intelligence **26** (1985) 251–321
0004-3702/85/$3.30

Despite increasing sophistication in problem-solving knowledge and heuristics, most AI systems employ relatively simple control programs. In contrast, people do not rely upon pre-determined control programs to guide all of their problem-solving efforts. Instead, they draw upon a repertoire of control knowledge that includes proven control programs and heuristics for dynamically constructing, modifying, and executing control programs during efforts to solve particular domain problems.

This adaptability in the control of one's own problem-solving behavior is the hallmark of human intelligence. Because of it, people do not simply solve a problem. They often know something about how they solve the problem, how they have solved similar problems in the past, why they perform one problem-solving action rather than another, what problem-solving actions they are likely to perform in the future, and so forth. They use this knowledge to adapt their behavior to the demands of the problem-solving situation, to explain their behavior, to cope with new problems, to improve their approaches to familiar problems, and to transfer problem-solving knowledge to other people and to computers. Truly intelligent AI systems must do no less.

Accordingly, this paper suggests that AI systems should approach the control problem as a real-time planning problem. It operationalizes *intelligent* control problem-solving as achievement of (at least) the following behavioral goals:

(1) Make explicit control decisions that solve the control problem.

(2) Decide what actions to perform by reconciling independent decisions about what actions are desirable and what actions are feasible.

(3) Adopt variable grain-size control heuristics.

(4) Adopt control heuristics that focus on whatever action attributes are useful in the current problem-solving situation.

(5) Adopt, retain, and discard individual control heuristics in response to dynamic problem-solving situations.

(6) Decide how to integrate multiple control heuristics of varying importance.

(7) Dynamically plan strategic sequences of actions.

(8) Reason about the relative priorities of domain and control actions.

Section 2 discusses and justifies each of these goals.

A 'blackboard control architecture' is proposed to achieve the goals. It has several important features, some of them previously suggested by other researchers. First, the architecture explicitly represents domain and control problems, knowledge, and solutions [2, 4, 5, 9, 11, 15, 16, 18, 20, 21, 24, 26]. Second, the architecture integrates domain and control problem-solving in a single basic control loop. Third, the architecture articulates, interprets, and modifies representations of its own knowledge and behavior [5, 8, 23, 27]. Fourth, the architecture adapts its problem-solving knowledge, its application of that knowledge, and its basic control loop to dynamic problem-solving situations. Finally, the architecture incorporates these features in a uniform

blackboard architecture in which domain and control problem-solving behaviors are characteristically incremental and opportunistic [1, 3, 7, 13]. Section 3 describes the blackboard control architecture. Section 4 shows how it achieves each of the behavioral goals enumerated above. Appendix A provides more detailed illustration with problem-solving traces from OPM [14], a blackboard control system for multiple-task planning.

Section 5 contrasts the blackboard control architecture with three alternatives: the basic blackboard architecture with a sophisticated scheduler, the basic blackboard architecture with solution-based focusing, and meta-level architecture. Although each of these alternatives has valuable features, none achieves all of the behavioral goals enumerated above. In addition, Appendix B shows how the blackboard control architecture could replicate the control behaviors of HEARSAY-II's sophisticated scheduler [6] and HASP's solution-based focusing [22].

Despite the blackboard control architecture's behavioral innovations, it does not depart radically from previous architectures. The second part of Section 5 shows how the blackboard control architecture continues an evolutionary progression of control architectures.

Section 6 summarizes the blackboard control architecture's strengths and weaknesses.

2. The Semantics of Control

What kinds of control behaviors should an AI system exhibit?

This section sets forth eight behavioral goals for an AI architecture. Each goal rests upon assumptions about effective problem-solving and the nature of intelligence. Thus, the goals are claimed to be necessary (but not sufficient) behaviors for an intelligent system. Although some readers may disagree with this analysis, at least it exposes the goals and assumptions underlying the blackboard control architecture to examination and criticism. It also provides criteria against which to evaluate the architecture and its alternatives.

2.1. An illustrative problem domain: multiple-task planning

The behavioral goals are illustrated with examples from a hypothetical intelligent planner that solves the following problem:

> You have just finished working out at the health club (see Fig. 1). It is 11:00 and you can plan the rest of your day as you like. However, you must pick up your car from the Maple Street parking garage by 5:30 and then head home. You would also like to see a movie today, if possible. Show times at both movie theaters are 1:00, 3:00, and 5:00. Both movies are on your 'must see' list, but go to whichever one most conveniently fits into your plan. Your

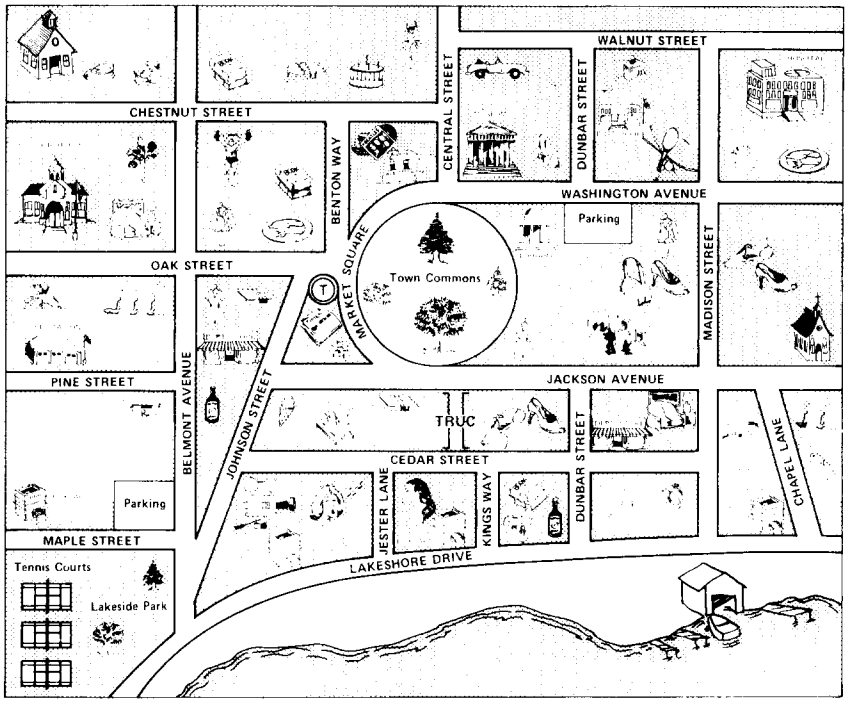

FIG. 1. Town map used for multiple-task planning.

other errands are as follows:
- Pick up medicine for your dog at the vet.
- Buy a fan belt for your refrigerator at the appliance store.
- Tour two of the three luxury apartments.
- Meet a friend for lunch at the restaurant of your choice.
- Buy a toy for your dog at the pet store.
- Pick up your watch at the watch repair.
- Special-order a book at the book store.
- Buy fresh vegetables at the grocery.
- Buy a gardening magazine at the newsstand.
- Go to the florist to send flowers to a friend in the hospital.

Develop a plan that:

(a) specifies which tasks to perform, the ordering of tasks, and the routes to travel between successive tasks;

(b) accomplishes as many tasks as possible, accomplishes high-priority tasks rather than low-priority tasks, honors time constraints, and provides efficient routes.

2.2. Behavioral goals for intelligent control

(1) *Make explicit control decisions that solve the control problem*

To solve a problem, a system performs a subset of its potential problem-solving actions in a particular order. Intelligent problem-solving entails explicitly solving the control problem—*deciding* which actions to perform and when to perform them—as well as performing selected actions. For example, given several possible next actions, the hypothetical planner explicitly decides to perform the action that locates the vet on the town map.

(2) *Decide what actions to perform by reconciling independent decisions about what actions are desirable and what actions are feasible.*

Intelligent control reasoning recognizes both the desirability and the feasibility of potential actions. An intelligent system decides what kinds of actions it should perform, thereby establishing its operative control heuristics. It decides what actions it can perform, thereby identifying its currently executable knowledge sources. The system reconciles these two kinds of decisions in order to decide what actions to perform. For example, the planner might decide that it should perform actions involving high-priority tasks. It might also decide that, in the current problem-solving situation, it can execute several knowledge sources, including one whose action would locate the vet, a high-priority task. Reconciling its task-priority heuristic and its executable knowledge sources, the planner decides to locate the vet.

Although an intelligent system does not allow either the desirability or the feasibility of potential actions to dominate control problem-solving, it drives the problem-solving process in either direction under appropriate circumstances.

Suppose the system decides that it should perform actions having a particular attribute, but discovers that it cannot perform any such actions. In other words, the specified actions are desirable, but not feasible. In that situation, an intelligent system performs actions that *enable* it to perform the desired actions. For example, the planner might decide that it should perform actions that generate abstract partial plans, but discovers that it cannot perform any such actions. In that situation, the planner examines its knowledge sources that produce abstract plans, determines that it needs task-location information in order to apply them, and adopts a control heuristic favoring actions that generate task-location information.

Conversely, suppose the system has decided that it can perform several pending actions that share an 'interesting' attribute, but notices that it has not yet decided that it should perform such actions. In other words, the specified actions are feasible, but not explicitly desirable. In that situation, an intelligent system performs actions that *incline* it to perform the feasible actions. For example, the planner might decide that it can perform several actions sharing the attribute 'triggered by time constraints', which is interesting because the

problem instructions include a requirement to honor time constraints. However, it also might notice that it has not yet decided that it should perform such actions. In that situation, the planner adopts a control heuristic favoring actions triggered by time constraints.

(3) *Adopt variable grain-size control heuristics.*

In some problem-solving situations, the most useful control heuristics prescribe classes of actions, while in others, the most useful heuristics prescribe specific actions. Similarly, in some situations, the most useful heuristics prescribe actions to be performed at any time during a relatively long problem-solving time interval, while in others, the most useful heuristics target actions for specific problem-solving 'cycles'. An intelligent system adopts control heuristics whose grain-size is appropriate for the problem-solving situation.

For example, the planner might generate two credible partial plans, PP-NW and PP-SE (see Fig. 2). In that situation, any actions that extend either partial plan or connect the two plans would advance the problem-solving process. Therefore, the planner adopts a relatively large-grain control heuristic favoring actions whose triggering information includes an unconnected end-point of either of the plans or whose solution context is the spatial-temporal region

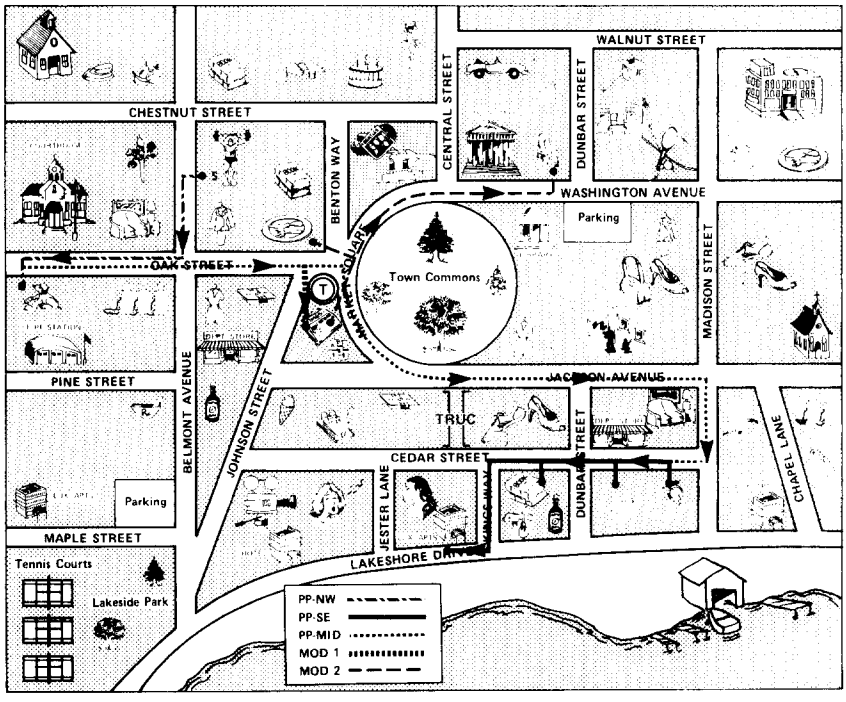

FIG. 2. Illustration of partial plans.

between the two plans. The heuristic prescribes performance of such actions during the problem-solving time interval beginning immediately and continuing until the two partial plans are connected.

Once the planner has connected the two partial plans, for example with PP-MID (see Fig. 2), the only remaining problem-solving objective is to connect the integrated plan with its termination point. In that situation, the planner adopts a smaller-grain control heuristic favoring actions whose knowledge source is Extend-Route and solution interval is '5:15 Lakeshore Apartments, 5:30 Maple Street Parking Garage'. The heuristic prescribes performance of those actions on the few remaining problem-solving cycles.

(4) *Adopt control heuristics that focus on whatever action attributes are useful in the current problem-solving situation.*

Attributes of a problem-solving action's triggering information, knowledge source, or solution context may affect its utility in different problem-solving situations. An intelligent system adopts control heuristics that focus on whatever attributes are useful in the current problem-solving situation. For example, if a planning problem specifies tasks that vary widely in priority, the planning system adopts a heuristic favoring actions triggered by high-priority tasks. If problem-solving time is at a premium, it adopts a heuristic favoring actions whose knowledge sources are efficient and reliable. If the planning system has already generated two credible partial plans, it adopts a heuristic favoring actions whose solution contexts represent the interval between those plans.

(5) *Adopt, retain, and discard individual control heuristics in response to dynamic problem-solving situations.*

Different control heuristics are useful in different problem-solving situations and the problem-solving situation can change repeatedly during the problem-solving process. An intelligent system adopts, retains, and discards control heuristics in response to dynamic problem-solving situations. For example, the planner might determine early in the problem-solving process that the requested tasks vary widely in priority and that the allowable plan execution time is insufficient for performing all of them. It immediately adopts a heuristic favoring actions triggered by high-priority tasks. Ordinarily, it would retain that heuristic for the duration of the problem-solving process. However, if the planner subsequently develops a partial plan that includes all of the high-priority tasks but does not exhaust the allowable execution time, it abandons the heuristic. For another example, sometime in the middle of the problem-solving process, the planner might generate the partial plans, PP-NW and PP-SE (see Fig. 2). At that point, it adopts a heuristic favoring actions that connect those plans. However, once the planner has completely connected PP-NW and PP-SE (for example, with PP-MID in Fig. 2), that heuristic is no longer useful and the planner abandons it.

(6) *Decide how to integrate multiple control heuristics of varying importance.*

Several operative control heuristics (those the system has adopted and not yet abandoned) may bear simultaneously, but differentially, on a particular problem-solving situation. An intelligent system integrates all operative heuristics in deciding which specific action to perform next. For example, the planner might decide that: (a) it is very important to generate abstract partial plans until there are at least three high-credibility partial plans; (b) it is always moderately important to perform actions whose knowledge sources are reliable; and (c) it is always moderately important to perform actions whose triggering information is credible. Until decision (a)'s criterion is met, the planner uses all three heuristics to decide which actions to perform. After (a)'s criterion is met, the planner abandons it and uses only the remaining two heuristics.

On a given problem-solving cycle, several potential actions may have different ratings against different subsets of the operative control heuristics. An intelligent system decides how to integrate each action's ratings and by what rule to choose a single action to perform next. For example, early in the problem-solving process, the planner might decide simply to choose the potential action that has the highest sum of ratings, weighted by importance, against all operative control heuristics. Later, as problem-solving time becomes more precious, it might decide to identify the subset of its potential actions that rate high on knowledge source speed and reliability and to choose from that set the single action that has the highest weighted sum of ratings against the remaining control heuristics.

(7) *Dynamically plan strategic sequences of actions.*

Strategic sequences of actions sometimes provide problem-solving power not achievable through independently chosen actions. An intelligent system plans strategic sequences of actions under appropriate circumstances. For example, the successive-refinement strategy reduces the combinatorics of search for multiple-task planning problems that specify a large number of tasks. The hypothetical planner uses that strategy to plan a sequence of: (a) actions that generate abstract partial plans; followed by (b) actions that refine, extend, and connect the most credible partial plans.

Strategic plans are complex control heuristics. Rather than prescribing a single kind of action, they prescribe a sequence of different kinds of actions. Therefore, an intelligent system deals with strategic plans just as it deals with simpler heuristics. It adopts variable grain-size strategic plans that focus on whatever action attributes are useful in the current problem-solving situation. It adopts, retains, and abandons strategic plans in response to dynamic problem-solving situations. It integrates strategic plans with its other control heuristics in deciding which specific action to perform next.

Dynamic planning also entails a readiness to interrupt, resume, or re-plan

performance of a strategic sequence of actions in response to dynamic problem-solving situations, as discussed below.

Some actions that do not conform to an adopted plan have other desirable attributes. An intelligent system interrupts its execution of strategic plans to perform such actions. For example, the planner might: (a) adopt the successive-refinement strategy; (b) begin performing a sequence of actions that, if completed, would generate PP-MID, connecting PP-NW with PP-SE (see Fig. 2); and (c) complete actions plotting the route segments beginning at the vet and ending at the intersection of Oak Street and Johnson Street. At this point, the planner can perform either of two actions, one that would continue the strategic action sequence and one that would interrupt it. The strategic action would plot a route segment from the intersection of Oak Street and Johnson Street to the next intersection on the way to PP-SE. The non-strategic action would modify PP-NW to include the newsstand, a task previously excluded from consideration for being low-priority, but now worth incorporating in the plan because it is very convenient to the plotted route. In this situation, the planner interrupts its strategic route-plotting actions to perform the action that incorporates the newsstand in the plan (MOD1 in Fig. 2).

Some interruptions do not affect a strategic plan's utility. On completing non-interfering interruptions, an intelligent system resumes plan execution. For example, inclusion of the newsstand makes only a minor modification to PP-NW. The planner subsequently resumes its strategic route-plotting actions to complete PP-MID.

On the other hand, some interruptions obviate pending strategic actions. On completing interfering interruptions, an intelligent system adapts its execution of the strategic plan to the new problem-solving situation. For example, following incorporation of the newsstand, the planner might perform additional interrupting actions, further extending PP-NW to include lunch at the Oak Street Restaurant followed by a stop at the Washington Avenue Pet Store (MOD2 in Fig. 2). Given this substantial extension of PP-NW, a route from the vet to the watch repair no longer makes sense. Therefore, the planner does not resume its efforts to plot that route. Instead, it 'backs up' in its strategic plan, first plotting routes within the extended PP-NW and then plotting a route connecting it to PP-SE—this time a route from the pet store to the watch repair.

(8) *Reason about the relative priorities of domain and control actions.*

Just as an intelligent system performs domain actions to generate solution elements for a domain problem, it performs control actions to generate solution elements for the control problem, namely control decisions. Because there is no *a priori* reason to presume that either type of action is more productive on any particular problem-solving cycle, the system reasons about the relative priorities of domain and control actions just as it reasons about

other control issues. For example, the planner might have two feasible actions: (a) a domain action that would generate an abstract plan encompassing three of the requested tasks: the health club, the florist, and the vet; and (b) a control action that would adopt a heuristic favoring actions whose knowledge sources are reliable. If, for example, the planner has an operative heuristic favoring actions that generate abstract partial plans, it performs the domain action first. Alternatively, if it has an operative heuristic favoring control actions, it performs the control action first. If the planner has both heuristics, it performs the action favored by the heuristic it considers more important.

3. The Blackboard Control Architecture

The blackboard control architecture extends and elaborates the standard blackboard architecture [7] in order to achieve the behavioral goals set forth in Section 2. Section 3.1 below summarizes the basic assumptions of the standard architecture and the blackboard control architecture's extension of them. Sections 3.2–3.4 examine the blackboard control architecture in more detail: its approach to domain problem-solving, its approach to control problem-solving, and its basic scheduling mechanism. These sections use the multiple-task planning domain introduced in Section 2 for illustration. Because both control and multiple-task planning are planning domains, the discussion distinguishes the 'control plan' and the 'domain plan' where necessary. Section 3.5 summarizes the blackboard control architecture's important features: explicit representation of domain and control problems, knowledge, and solutions; integration of domain and control problem-solving in a single basic control loop; interpretation and modification of a system's own knowledge and behavior; adaptation of knowledge and behavior to dynamic problem-solving situations; and uniform treatment of domain and control problem-solving as incremental, opportunistic processes. The blackboard control architecture is implemented as BB1 [14], a domain-independent system-building environment.

3.1. From the blackboard architecture to the blackboard control architecture

The blackboard architecture is a problem-solving framework developed for the HEARSAY-II speech-understanding system [7], used in a variety of other problem domains [11, 15, 21, 23, 26, 30], and abstracted in several system-building environments [8, 14, 25]. It treats problem-solving as an incremental, opportunistic process of assembling a satisfactory configuration of solution elements. For example, it treats multiple-task planning as a process of assembling a satisfactory configuration of decisions about what tasks to perform, when to perform them, and so forth. It entails three basic assumptions:

(1) *All solution elements generated during problem-solving are recorded in a structured, global database called the blackboard.*

The blackboard structure organizes solution elements along two axes, solu-

tion intervals and levels of abstraction. Different solution intervals represent different regions of the solution on some problem-specific dimension(s). For example, for the multiple-task planning domain, solution intervals represent plan execution time intervals. Different levels of abstraction represent the solution in different amounts of detail. For example, for the multiple-task planning domain, one level of abstraction represents a planned sequence of tasks, while a lower level also elaborates the details of performing individual tasks and the routes for travelling between successive tasks.

(2) *Solution elements are generated and recorded on the blackboard by independent processes called knowledge sources.*

Knowledge sources have a condition-action format. The condition describes situations in which the knowledge source can contribute to the problem-solving process. Ordinarily, it requires a particular configuration of solution elements on the blackboard. The action specifies the knowledge source's behavior. Ordinarily, it entails the creation or modification of solution elements on the blackboard. Only knowledge sources whose conditions are satisfied can perform their actions.

Although implementations vary, in most blackboard systems knowledge-source activity is event-driven. Each change to the blackboard constitutes an event that, in the presence of specific other information on the blackboard, can trigger (satisfy the condition of) one or more knowledge sources. Each such triggering produces a unique knowledge-source activation record (KSAR). A KSAR is similar to an item on a task agenda. It represents a unique triggering of a particular knowledge source by a particular blackboard event. When a KSAR is chosen by the scheduling mechanism discussed below, its knowledge source's action executes in the context of its triggering information, typically producing new blackboard events.

Knowledge sources are independent in that they do not invoke one another and ordinarily have no knowledge of each other's expertise, behavior, or existence. They are cooperative in that they contribute solution elements to a shared problem. The blackboard architecture achieves simultaneous independence and cooperation among knowledge sources by permitting them to influence one another's problem-solving behavior only indirectly, by anonymously responding to and modifying information recorded on the blackboard.

(3) *On each problem-solving cycle, a scheduling mechanism chooses a single KSAR to execute its action.*

Because several KSARs may compete to execute their actions, a scheduling mechanism determines which KSARs execute their actions and in what order. The most common mechanism is a 'sophisticated scheduler' that uses a variety of criteria (e.g., knowledge-source reliability, triggering information credibility, expected value of the knowledge source's action) to choose a KSAR for execution on each problem-solving cycle. The sophisticated scheduler is discussed in more detail in Section 5.

The architectural features summarized above interact to produce a problem-solving style that is characteristically incremental and opportunistic. Executed KSARs apply various inference methods (e.g., goal-driven, data-driven) to generate solution elements, one at a time, in different blackboard locations. They extend the most promising solution elements and eventually merge them with others to form the complete solution. The relative orderliness with which the solution develops depends largely upon the scheduler's behavior. At one extreme, it can follow a rigorous procedure, scheduling a planned sequence of KSARs that monotonically assemble compatible solution elements. At the other extreme, it can apply a variety of conflicting criteria, opportunistically scheduling KSARs that assemble disparate, competing solution elements out of which a complete solution emerges only gradually.

Briefly, the blackboard control architecture extends and elaborates the three standard assumptions as follows:

(1) *The blackboard control architecture defines explicit domain and control blackboards.*

The domain blackboard records solution elements for the current domain problem. Its solution intervals, levels of abstraction, and vocabulary are domain-specific and determined by the designer of an application system. The control blackboard records solution elements for the control problem. These solution elements are decisions about a system's own desirable, feasible, and actual behavior. The control blackboard's solution intervals, levels of abstraction, and vocabulary are domain-independent and defined by the architecture.

(2) *The blackboard control architecture defines explicit domain and control knowledge sources.*

Domain knowledge sources operate primarily on the domain blackboard. They are domain-specific and determined by the application system designer. Control knowledge sources operate primarily on the control blackboard. They articulate, interpret, and modify representations of the system's own knowledge and behavior. Some control knowledge sources are domain-specific; others are domain-independent. All knowledge sources are represented as data structures that are, themselves, available for interpretation and modification.

(3) *The blackboard control architecture defines a simple, adaptive scheduling mechanism to manage both domain and control KSARs.*

Three 'basic' control knowledge sources iterate a three-step problem-solving cycle: (1) enumerate pending KSARs; (2) choose one KSAR; (3) execute the action of the chosen KSAR. The basic control knowledge sources have no specific control knowledge, but simply adapt to dynamic solution state information recorded on the domain and control blackboards. Because they manage both domain and control KSARs, the basic control knowledge sources indirectly influence their own behavior.

3.2. Domain problem-solving

The blackboard control architecture solves domain problems with the same methods used in standard blackboard systems. Domain knowledge sources respond to, generate, and modify solution elements on a domain blackboard, under the control of a scheduling mechanism. This section presents an illustrative domain blackboard and knowledge sources. The architecture does not specifically include these components. It defines them to be domain-specific and determined by the application system designer, requiring only that they satisfy the architectural constraints given above. In the present case, the domain blackboard and knowledge sources were derived from thinking-aloud protocols produced by people while working on multiple-task planning problems [15]. They illustrate the kind of blackboard structure and knowledge sources assumed by the architecture and provide a context in which to discuss its other components.

3.2.1. *A domain blackboard for multiple-task planning*

Solution elements for the multiple-task planning problem are decisions about which tasks to perform, in what order to perform tasks, and by what routes to travel between successive task locations. Each decision is represented as a data structure with attributes and values. Because these attributes are incidental to the multiple-task planning domain and not intrinsic to the blackboard control architecture, this discussion ignores them and gives only the gist of domain decisions.

A related set of decisions on the domain blackboard constitutes a partial domain plan. For example, the partial plan in Fig. 3 specifies:

> Perform a set of tasks including the health club, the pharmacy, the vet, etc. First perform tasks in the northwest part of town, in this order: health club, florist, vet. . . . Go from the florist to the vet by travelling south on Belmont to Oak and west on Oak to the vet. . . .

As the example illustrates, partial plans may encompass variable-size segments of an evolving complete plan. At any point in the problem-solving process, the domain blackboard may contain several complementary or alternative partial plans. The multiple-task planning problem is solved when the system has generated and merged one or more partial plans to form a complete and satisfactory plan at the lowest level of the domain blackboard.

Different solution intervals on the multiple-task planning blackboard represent different temporal intervals in the plan execution sequence. Decisions toward the left side of the blackboard designate tasks or travel planned to occur early in the plan execution sequence, while those on the right side designate tasks or travel planned to occur later in the sequence.

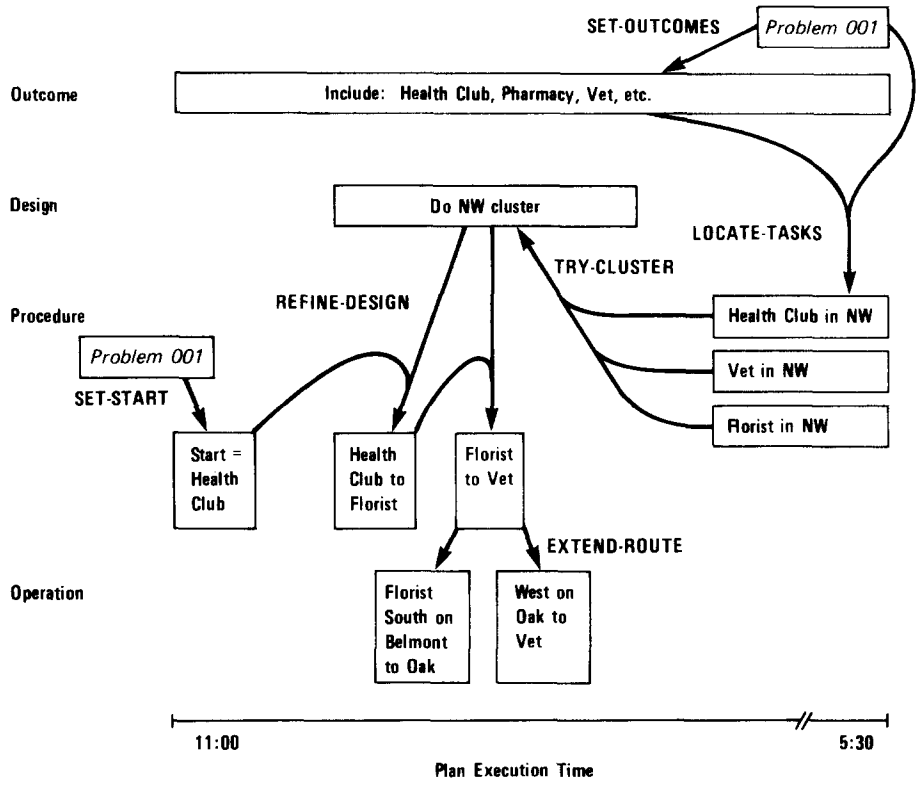

FIG. 3. Partial multiple-task plan.

Outcome Level
Definition: Tasks included in or excluded from the plan
Example: Include: health club, pharmacy, vet, etc.

Design Level
Definition: General temporal layout of the plan
Example: Do tasks in the northwest cluster first

Procedure Level
Definition: Sequence of individual tasks
Example: Do the vet after the florist

Operation Level
Definition: Details of task performance and travel between tasks
Example: Go from the florist south on Belmont Ave. to Oak St.

FIG. 4. Levels of abstraction for the multiple-task planning blackboard.

The multiple-task planning blackboard has four levels of abstraction. Fig. 4 defines and provides an example decision for each level.

3.2.2. *Domain knowledge sources for multiple-task planning*

Domain knowledge sources solve the multiple-task planning problem by creating and modifying decisions on the domain blackboard. The blackboard control architecture represents all knowledge sources as data structures with the attributes defined in Table 1. Knowledge sources reside in a system's 'permanent' memory, as opposed to the 'working' memory embodied in blackboard structures.

For example, Fig. 5 describes the knowledge source Refine-Design. As indicated by its Problem-Domain, Refine-Design is useful for multiple-task planning problems. As indicated by its Description, it refines a Design decision as a series of Procedure decisions. Refine-Design's Trigger indicates that it is potentially useful in situations in which a blackboard event creates a new Procedure decision that sequences two tasks. Its Pre-Condition indicates that it cannot actually execute its Action unless: (a) there is a decision about the general Design of the plan that constrains extension of the triggering Procedure; and (b) there are decisions establishing the locations and orientations (directions and distances) of unplanned tasks with respect to the second task in the triggering Procedure. Refine-Design's Condition-Vars specify that the values of P and D that satisfy its Condition predicates should be bound and used to perform its Action. Its Scheduling-Vars provide values (or functions for computing values) for the variables: Trigger-Weight, Action-Level, Action-Interval, KS-Efficiency, KS-Credibility, and KS-Importance, to be recorded in any KSAR generated for Refine-Design and available for use in scheduling. Finally, its Action, when executed, creates a new Procedure decision that extends the triggering Procedure in accordance with the specified Design. Because Refine-Design can be triggered by the events it creates, ordinarily it is

TABLE 1. Attributes of knowledge sources

Attribute	Definition
Name	Identifying label
Problem-Domain	Domain(s) of application
Description	Characteristic behavior
Condition	Situation of interest: Trigger and Pre-Condition
Trigger	Event-based predicates for triggering
Pre-Condition	State-based predicates required for execution
Condition-Vars	Specification of variables used in Condition
Scheduling-Vars	Specification of variables to be used in scheduling
Action	Program of blackboard changes

```
Name = 'Refine-Design'
Problem-Domain = (Multiple-Task Planning)
Description = 'Refines a Design decision as a series of
              Procedure decisions'
Condition =
  Trigger =
    An Event creates a new Procedure, P
    P's Type is 'sequence'
  Pre-Condition =
    There is a Design, D, whose specification
      encompasses P's Task2
    For each unplanned task, there is a Procedure whose Type
      is 'location' and Orientation-List includes P's Task2
Condition-Vars = (P, D)
Scheduling-Vars =
  Trigger-Weight = P's Weight
  Action-Level = 'Procedure'
  Action-Interval = >P
  KS-Efficiency = 0.5
  KS-Credibility = 0.9
  KS-Importance = 0.9
Action =
  Let Candidates be the set of unplanned tasks whose Orientation
    to P's Task2 satisfies D
  Let Next-Task be the member of Candidates that is closest to
    P's Task2
  Create a Procedure whose Type is 'sequence'
  Let its Task1 be P's Task2
  Let its Task2 be Next-Task
```

FIG. 5. Multiple-task planning knowledge source: Refine-Design.

TABLE 2. Attributes of knowledge source activation records (KSARs)

Attribute	Definition
Name	Identifying number
KS	Name of the triggered knowledge source
Triggering-Cycle	Cycle on which the knowledge source was triggered
Triggering-Event	Event that triggered the knowledge source
Triggering-Decision	Decision where the Triggering-Event occurred
Pre-Condition-Values	Pre-Conditions and current bindings
Condition-Values	Bindings of Condition-Vars to be used in Action
Scheduling-Values	Bindings of Scheduling-Vars to be used in scheduling
Ratings	Ratings on operative heuristics
Priority	Expected (numerical) value of action

triggered repeatedly to refine a single Design as a connected series of Procedures.

The blackboard control architecture represents KSARs for triggered domain knowledge sources as data structures with the attributes in Table 2. It records them at the To-Do-Set level of the control blackboard (discussed below) to represent domain actions the problem-solving system can perform. For example, when triggered by event E-40 on cycle 31, Refine-Design generates KSAR-103, shown in Fig. 6.

KSAR-103 indicates that the knowledge source Refine-Design is triggered on cycle 31 by event E-40 at decision Procedure5. Its Pre-Condition-Values indicate that its first Pre-Condition is satisfied by Design1, but that its second Pre-Condition is not yet satisfied, implying that KSAR-103 cannot actually execute its Action yet. Its Condition-Values indicate that KSAR-103 should use Procedure5 and Design1 to execute its Action. Its Scheduling-Values indicate that the system's confidence in KSAR-103's triggering information is 0.8, that its Action would produce a new event at the Procedure level in the solution interval beginning at 11:20, that Refine-Design's efficiency is 0.5, that it produces useful decisions 90% of the time, that it operates on the domain blackboard, and that it would produce a very important decision. (The Ratings and Priority attributes of KSARs are discussed in Section 3.4.1 below.)

3.2.3. *An example of multiple-task planning*

Fig. 3 shows how several multiple-task planning knowledge sources might interact to produce a partial plan incrementally:

(1) Set-Outcomes, triggered by a Problem decision on the control blackboard (discussed below), creates an Outcome decision to include the health club, the pharmacy, the vet, etc., in the plan.

KSAR-103
KS = 'Refine-Design'
Triggering-Cycle = 31
Triggering-Event = E-40
Triggering-Decision = Procedure5
Pre-Condition-Values =
  ((There is a Design, *D*, whose specification encompasses *P*'s
    Task2) ('Design-001's specification encompasses 'florist'))
  ((For each unplanned task, there is a Procedure whose Type is location
    and Orientation-List includes 'vet'))
Condition-Values = ((*P* Procedure5) (*D* Design1))
Scheduling-Values = ((Trigger-Weight 0.8) (Action-Level 'Procedure')
  (Action-Interval > 11:20) (KS-Efficiency 0.5) (KS-Credibility 0.9)
  (Action-Blackboard 'Domain') (KS-Importance 0.9))
Ratings = Nil
Priority = 0

FIG. 6. An example KSAR.

(2) Set-Start, also triggered by the Problem decision, creates a Procedure decision to start at the health club.

(3) Locate-Tasks, also triggered by the Problem decision, begins creating Procedure decisions locating each task.

(4) Notice-Cluster, triggered by creation of three Procedure decisions locating the health club, the vet, and the florist in the northwest part of town, creates a Design decision to perform those tasks first.

(5) Refine-Design, triggered previously by creation of the Procedure decision to start at the health club, now has its Pre-Conditions satisfied by creation of the Design decision and by the Procedure decisions locating tasks. It creates a Procedure decision to go from the health club to the florist.

(6) Refine-Design, triggered by the Procedure decision to go from the health club to the florist, and constrained by the Design decision, creates a Procedure decision to go from florist to the vet.

(7) Extend-Route, triggered by the Procedure decision to go from the florist to the vet, creates an Operation decision to travel from the florist, south on Belmont, to Oak.

(8) Extend-Route, triggered by the Operation decision to travel from the florist to Oak, creates an Operation decision to travel west on Oak to the vet.

This example abstracts a cooperative interaction among selected KSARs out of a larger problem-solving context. In practice, other KSARs would compete with those discussed above, possibly assisting in the development of the partial plan in Fig. 3, possibly interrupting or even terminating its development. Section 4 discusses some of these other behavior patterns. Appendix A presents program traces from OPM, a blackboard system for multiple-task planning, to illustrate some of these other behavior patterns in detail.

3.3. Control problem-solving

The blackboard control architecture solves the control problem with the same methods it uses for domain problems. Control knowledge sources respond to, generate, and modify solution elements on a control blackboard, under the control of a scheduling mechanism. They dynamically compose a prescriptive control plan out of modular control heuristics. A simple scheduler adapts to the current control plan to select potential actions for execution. The architecture defines specific solution intervals and levels of abstraction for a domain-independent control blackboard. It defines the attributes and values of decisions that occur at each level. It entails several generic control knowledge sources, but permits additional domain-specific control knowledge sources.

3.3.1. *The control blackboard*

Solution elements for the control problem are decisions about what actions are desirable, feasible, and actually performed at each point in the problem-solving

process. Each control decision is represented as a data structure with the attributes defined in Table 3. These attributes are instantiated for decisions at each level of abstraction below.

A related set of decisions on the control blackboard constitutes a partial control plan. For example, the partial plan in Fig. 7 specifies:

> Solve Problem *P*, etc. Prefer KSARs whose Actions occur at successive domain levels. Begin with KSARs whose Actions occur at the Outcome level.... Also prefer KSARs triggered on recent problem-solving cycles.... On cycle 005, these KSARs are feasible: KSAR-005, KSAR-006, KSAR-007. On cycle 005, perform KSAR-007....

As the example illustrates, partial control plans may encompass variable-size segments of an evolving control plan. At any point in the problem-solving process, the control blackboard may contain several complementary or competing partial plans. As discussed below, the control problem is solved when the Problem's Criterion is met and its Status is changed to 'solved'.

Different solution intervals on the control blackboard represent different problem-solving time intervals in terms of problem-solving cycles. Control decisions toward the left side of the control blackboard refer to problem-solving actions planned for early problem-solving cycles, while those on the right side refer to actions planned for later problem-solving cycles.

The control blackboard's levels of abstraction represent different categories of control decisions. Decisions at the *Problem*, *Strategy*, *Focus*, and *Policy* levels describe desirable actions, thereby determining which of the system's control heuristics operate during particular problem-solving time intervals. As explained below, heuristics at these different levels differ in scope. Decisions at the To-Do-Set level describe feasible actions, identifying all KSARs eligible for

TABLE 3. Basic attributes of control decisions

Attribute	Definition
Name	Identifying level and number
Goal	Prescribed action (predicate or function of KSAR attributes)
Criterion	Expiration condition (predicate)
Weight	Goal importance (0–1)
Rationale	Reason for Goal
Creator	KSAR that created the decision
Source	Triggering decision (or Input)
Type	Role in control plan (e.g., Strategic, Solution-Based)
Status	Function in control plan (e.g., Operative, Inoperative)
First-Cycle	First operative cycle
Last-Cycle	Last operative cycle

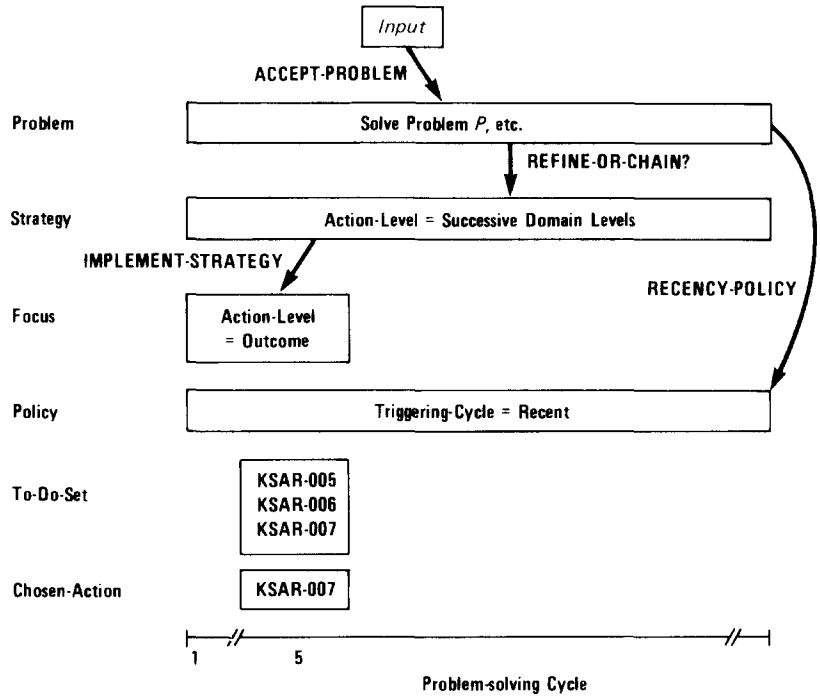

FIG. 7. Partial control plan.

execution on each problem-solving cycle. Decisions at the Chosen-Action level describe actions scheduled for execution on each problem-solving cycle. In general, the architecture uses heuristics at the Problem, Strategy, Focus, and Policy levels to evaluate pending KSARs at the To-Do-Set level in order to schedule KSARs for execution at the Chosen-Action level.

Fig. 8 defines and gives examples of decisions at each level of the control blackboard. Each decision in Fig. 8 corresponds to one of the decisions in the partial control plan in Fig. 7. For brevity, the example decisions in Fig. 8 include only Goal and Criterion attributes. The following sections discuss all the attributes of each decision (see Table 3) in detail.

3.3.1.1. *Problem decisions*

A single Problem decision represents the problem a system has decided to solve and guides the entire problem-solving episode. Creating a Problem decision on the blackboard initiates problem-solving by triggering at least one domain or control knowledge source. The Problem's attributes may subsequently trigger or constrain the behavior of other knowledge sources. Changing the Problem's Status to 'solved' terminates problem-solving activities.

```
Problem
  Definition: Problem the system has decided to solve
  Example:
    Goal = (KS Problem-Domain = multiple-task-planning)
    Criterion =
      ((complete solution at lowest domain Level)
       (all Requested-Tasks planned)
       (all Constraints met)
       (route efficient))

Strategy
  Definition: General sequential plan for solving the problem
  Example:
    Goal = (Action-Level = successive domain Levels)
    Criterion = complete solution spanning all domain Levels

Focus
  Definition: Local (temporary) problem-solving goals
  Example:
    Goal = (Action-Level = Outcome)
    Criterion = complete solution at Outcome Level

Policy
  Definition: Global (permanent) scheduling criteria
  Example:
    Goal = (Triggering-Cycle = recent)

To-Do-Set
  Definition: Sets of pending KSARs
  Example:
    Goal = ((Triggered-List = (KSAR-008, KSAR-009))
            (Invocable-List = (KSAR-005, KSAR-006, KSAR-007)))

Chosen-Action
  Definition: KSARs chosen to execute
  Example:
    Goal = KSAR-007
```

FIG. 8. Levels of abstraction for the control blackboard.

A Problem's Goal prescribes actions whose knowledge sources apply to its domain, for example 'KS Problem-Domain = multiple-task-planning'. Its Criterion characterizes an acceptable solution. OPM's Problem Criterion includes four requirements: 'complete solution at lowest domain Level, all Requested-Tasks planned, all Constraints met, planned route efficient'. For some domains, including multiple-task planning, solving the Problem entails modifying or elaborating the initial Criterion or Constraints. For example, if there is not enough time to perform all Requested-Tasks, the Criterion predicate 'all Requested-Tasks planned' can be changed to 'all important Requested-Tasks planned'.

The Criterion in this example refers to several attributes that occur only at the Problem level: Levels of abstraction for the problem domain ('Outcome, Design, Procedure, Operation'), the Problem's Requested-Tasks ('health club, vet, etc.'), and the Problem's Constraints ('(start 1100), (finish 1730), etc.'). Two other level-specific attributes indicate that the Problem-Domain is 'multiple-task planning' and that the available Problem-Solving-Time is '10 minutes''.

The Problem's Weight indicates the importance of solving the Problem, ordinarily 1, the highest possible value. Its Rationale is either task-specific and provided by the system designer, for example 'to demonstrate planning under uncertainty', or the default 'to cooperate with the system designer'.

Ordinarily, a Problem's Source is 'input', its Type is 'requested by system designer', and its Creator is 'KSAR-001', the first KSAR created and executed. Its Status is 'operative' until some knowledge source determines that its Criterion has been met and changes its Status to 'inoperative'. Assuming the system can solve only one problem at a time, the Problem's First-Cycle is 1. The knowledge source that changes the Problem's Status to 'inoperative' also records the current cycle number as its Last-Cycle.

The discussion above assumes that a system can solve only problems posed by the system designer (or other user). If a system posed problems to itself, the Source might be 'KS 555', the Type might be 'self-imposed', and the Rationale might be 'experiment with a new Strategy knowledge source'.

3.3.1.2. *Strategy decisions*

Strategy decisions establish general sequential plans for problem-solving. Ideally, a single Strategy decision created early in the problem-solving process guides the remainder of the process. However, Strategy decisions may occur at any point in the process and span arbitrary intervals of the remaining problem-solving time. Under some circumstances, conflicting or complementary Strategies may replace one another or operate simultaneously.

Functionally, a Strategy provides parameters for a generic control knowledge source, Implement-Strategy, that generates more specific decisions implementing the Strategy at the lower Focus level.[1] As discussed below, Focus decisions, but not Strategy decisions, are used to rate and schedule KSARs. Thus, Strategy decisions do not directly influence scheduling decisions; they influence them indirectly through the Focus decisions that implement them.

A Strategy's Goal describes a prescribed sequence of actions as a predicate on one or more KSAR attribute-description pairs. The attributes must be legal KSAR attributes. The descriptions must entail sequences of legal values.

[1] For complex strategic plans, the architecture permits additional levels to intervene between Strategy and Focus levels. For example, representing HEARSAY-II's Strategy on the control blackboard requires an intervening Tactic level (see Appendix B). In that case, Strategy decisions function as generators for Tactic decisions, which function as generators for Focus decisions. Thus, any intermediate levels expand the aggregation hierarchy defined for Strategy and Focus levels.

For example, 'Action-level = successive domain Levels' prescribes a sequence of actions that add or modify solution elements at successive Levels of the domain blackboard. A Strategy's Criterion characterizes the desired result of applying the Strategy, for example 'complete solution spanning all domain Levels'. Its Weight (0–1) indicates the expected utility of applying the Strategy, for example 0.9. Its Rationale is the reason for recommending the Strategy, such as 'insufficient time to perform all Requested-Tasks'.

Four attributes specific to the Strategy level provide parameters for implementing the Strategy's Goal as a series of more specific Goals at the Focus level. Consider implementation of the Goal 'Action-Level = successive domain Levels'. The Strategy's Focus1 describes the first Focus Goal it prescribes as a predicate or function on one or more KSAR attribute-description pairs. The attributes must be legal KSAR attributes. The descriptions must evaluate to legal values. The Focus1 'Action-Level = highest domain level' evaluates to 'Action-Level = Outcome'. The Strategy's Refocus-Criterion is a predicate that, when true, indicates that the Strategy prescribes a new Focus, for example 'complete and satisfactory solution in Focus'. The Strategy's Refocus-Increment is a function that, when applied to the current Focus Goal, generates the Goal of the next prescribed Focus. Applying the Refocus-Increment 'level below' to the Goal 'Action-Level = Outcome' generates the Goal 'Action-Level = Design'. The Strategy's FocusN describes the last Focus Goal it prescribes, for example 'Action-Level = lowest domain level', which, for multiple-task planning, evaluates to 'Action-Level = Operation'. Thus, the Strategy Goal 'Action-Level = successive domains Levels' prescribes a series of Focus decisions whose Goals are 'Action-Level = Outcome', 'Action-Level = Design', Action-Level = Procedure', and 'Action-Level = Operation'. It further prescribes that each Focus should remain operative until a complete and satisfactory solution exists at the level currently in the strategic Focus.

Ordinarily, a Strategy's Source is 'Problem1' and its Type is 'problem appropriate', indicating that it is a known Strategy for handling the current Problem. However, a Strategy might be induced from experience or transferred from another problem domain. In that case, its Source would be the triggering information, its Type would be 'induced' or 'transferred', and its Rationale might be 'apparent solution improvement' or 'useful for analogous problem domain: spatial planning'.

A Strategy's Creator is the KSAR that created it, for example 'KSAR-002'. A Strategy's Status is 'operative' until some knowledge source determines either that its Criterion has been met or that it is no longer productive and changes its Status to 'inoperative'. Its First-Cycle is the cycle after it is created and its Last-Cycle is the cycle on which it becomes inoperative.

3.3.1.3. *Focus decisions*

Focus decisions establish local problem-solving objectives to execute KSARs

with particular attributes and values. Sometimes a sequence of Focus decisions implements a previously created Strategy decision. Other Focus decisions operate independently of one another and of prior Strategy decisions. Focus decisions are explicitly temporary and operate during restricted problem-solving time intervals. Several complementary or competing Focus decisions may operate simultaneously.

Focus decisions are used to rate KSARs. As a consequence, they can influence scheduling decisions. However, a given Focus decision actually influences scheduling decisions only if: (a) the current To-Do-Set contains KSARs with the attributes and values it prescribes; and (b) the current integration and scheduling rules (discussed in Section 3.4 below) incorporate the Focus decision.

A Focus decision's Goal is a predicate or function on one or more KSAR attribute-value pairs.[2] For example, the Goal 'Action-Level = Outcome' prescribes execution of KSARs whose Actions occur at the Outcome Level. Its Criterion is a predicate that, when true, indicates that the Goal is no longer useful, for example 'complete solution at the Outcome Level'. Its Weight indicates the expected utility of the prescribed actions.

A Focus decision's Rationale, Source, and Type depend upon its etiology. If the Focus implements a previously generated Strategy, its Rationale describes its role within the Strategy, for example 'implements Strategy's Focus1', its Source is the Strategy, for example 'Strategy1', and its Type is 'strategic'. If it simply prescribes feasible problem-solving actions—that is, an identified subset of those appearing in recent To-Do-Sets—its Rationale is 'is feasible'. Its Source is the set of To-Do-Sets, for example 'To-Do-Set15, To-Do-Set25', and its Type is 'inferred'. If it prescribes actions in a blackboard region where progress would be especially useful, its Rationale might be 'unsatisfactory partial solution' or 'contiguous to a credible partial solution', its Source is the blackboard region, for example 'domain interval 10', and its Type is 'solution-based'.

A Focus decision's Creator is the KSAR that created it, for example 'KSAR-003'. Its Status is 'operative' until some control knowledge source notices that its Criterion has been met and changes its Status to 'inoperative'. Its First-Cycle and Last-Cycle are the first and last cycles on which it is operative.

3.3.1.4. *Policy decisions*

Policy decisions establish global scheduling criteria favoring KSARs with particular attributes and values. In contrast to Focus decisions, Policy decisions ordinarily remain operative from the time they are created until the end of the

[2] Focus Goals can be predicates whose true/false values indicate satisfaction/non-satisfaction of the Goal or functions whose numerical values indicate degree of satisfaction. An example of a Goal function appears in the discussion of Policy decisions (Section 3.3.1.4).

problem-solving episode. They carry no internal expiration Criteria. The only exception occurs when some knowledge source independently determines that a Policy should become inoperative. Ordinarily, several Policy decisions operate simulataneously.

Like Focus decisions, Policy decisions are used to rate KSARs and they can influence scheduling decisions. Again, however, a given Policy decision actually influences scheduling decisions only if: (a) the current To-Do-Set contains KSARs with the attributes and values it prescribes; and (b) the current integration and scheduling rules incorporate the Policy decision.

A Policy decision's Goal is a predicate or function on one or more KSAR attribute-value pairs. For example, the Goal 'Triggering-Cycle = recent' prescribes execution of KSARs triggered on the last few problem-solving cycles.[3] A Policy's Weight is the expected value of its prescribed actions.

A Policy's Rationale, Source, and Type depend upon its etiology. If a Policy decision establishes scheduling criteria known to be useful for the current Problem, its Rationale might be 'useful for multiple-task planning problems', its Source is 'Problem1', and its Type is 'Problem-appropriate'. If a Policy establishes scheduling criteria in response to a particular problem-solving state, its Rationale characterizes the state, for example 'problem-solving time is running out', its Source is 'problem-solving state on cycle c', and its Type is 'solution-based'.

A Policy's Creator is the KSAR that created it. Its Status remains 'operative' until either the problem-solving process terminates or a domain-specific control knowledge source changes its Status to 'inoperative'. Its First-Cycle and Last-Cycle are the first and last cycles on which it is operative.

3.3.1.5. *To-Do-Set decisions*

To-Do-Set decisions identify all pending KSARs on each problem-solving cycle. They record a cycle-by-cycle history of all problem-solving actions the system can perform. Because knowledge source Triggers are event-based while Pre-Conditions are state-based and possibly transient, a To-Do-Set's Goal distinguishes a Triggered-List and an Invocable-List of pending KSARs. Update-To-Do-Set, a basic control knowledge source defined in the Section 3.4, places newly triggered KSARs on the Triggered-List. When their Pre-Conditions become true (after an arbitrary problem-solving time interval), it moves them to the Invocable-List. If any of their Pre-Conditions subsequently become false, it moves them back to the Triggered-List. KSARs can move back and forth between the two lists. However, only KSARs on the Invocable-List can be scheduled for execution. A To-Do-Set's Only-Cycle is the only problem-solving cycle on which it is operative.

[3] A more sophisticated form of the same Goal might, for example, specify the function '(Prefer-Recent Triggering-Cycle)' to return the numerical values 100, 90, 50, 0 for Triggering-Cycle = the current cycle, the preceding cycle, 2–5 cycles ago, 6 or more cycles ago.

3.3.1.6. *Chosen-Action decisions*

Chosen-Action decisions identify KSARs scheduled to execute on each problem-solving cycle. Thus, they record a cycle-by-cycle history of all problem-solving actions the system actually does perform. A Chosen-Action's Goal is the scheduled KSAR, for example 'KSAR-005'. Its Rationale lists the Focus and Policy decisions that lead to its selection, for example 'Focus1, Policy1'. Its Consequences list the blackboard events produced by executing the KSAR's Action. Its Status is 'scheduled' at the time it is created and changes to 'executed' after it executes. Its Only-Cycle is the only cycle on which it is operative.

3.3.2. *Higher-level control knowledge sources*

Control knowledge sources solve the control problem by creating, modifying, and interpreting decisions on the control blackboard. Higher-level control knowledge sources, discussed in this section, determine what kinds of problem-solving actions are desirable. Thus, they determine which of the system's control heuristics operate during particular problem-solving time intervals. Three basic control knowledge sources, discussed in Section 3.4 below, identify feasible actions and, based on the operative control heuristics, determine which of the feasible actions is executed on each problem-solving cycle.

Higher-level control knowledge sources are represented as data structures with the attributes defined in Table 1. A few of them (for example, the knowledge source Implement-Strategy mentioned below) are generic; most are domain-specific.

Figure 9 shows the illustrative control knowledge source, Refine-or-Chain? As indicated by its Problem-Domain, Refine-or-Chain? is a domain-specific control knowledge source appropriate for multiple-task planning problems. However, notice that Refine-or-Chain?'s other attributes are expressed in domain-independent terms. Thus, its knowledge is, in principle, general over multiple domains and its Problem-Domain potentially expandable to include other domains.

Refine-or-Chain? decides which of two problem-solving strategies the system should use. Its Trigger specifies the creation of a new Problem decision. It has no additional Pre-Condition. Its Condition-Vars bind the value of *P*, the triggering Problem decision, for use in carrying out its Action. Its Scheduling-Vars bind the values of several potentially useful scheduling variables. Its Action examines the triggering Problem's attributes to evaluate the feasibility of performing all of the Requested-Tasks in the available plan execution time. If performing all of the Requested-Tasks is not feasible, Refine-or-Chain? changes the Problem's Criterion 'all Requested-Tasks planned' to 'all important Requested-Tasks planned' and creates the successive-refinement Strategy decision discussed above. It gives the Strategy a high Weight, 0.9, and records its Rationale, 'insufficient time for all Requested-Tasks'. On the other hand, if performing all of the Requested-Tasks is clearly feasible, Refine-or-

```
Name = 'Refine-or-Chain?'
Problem-Domain = (Multiple-Task Planning)
Description = 'Creates a successive-refinement strategy if plan execution time is inadequate;
              otherwise creates a temporal-chaining strategy'
Condition =
  Trigger =
    An Event creates a new Problem, P
  Pre-Condition = T
Condition-Vars = (P)
Scheduling-Vars =
  Trigger-Weight = P's Weight
  Action-Blackboard = 'Control'
  Action-Level = 'Strategy'
  KS-Efficiency = 0.7
  KS-Credibility = 0.8
  KS-Importance = 1
Action =
  Create a Strategy
  If P's Execution-time not ≫
       time required for all of P's Requested-Tasks
    Then change P's Criterion 'all Requested-Tasks planned'
         to 'all important Requested-Tasks planned'
      Let the new Strategy's Goal be
        'Action-Level = successive domain Levels'
      Let its Criterion be
        'complete plan spanning all domain Levels'
      Let its Weight be 0.9
      Let its Rationale be
        'insufficient time for all requested tasks'
      Let its Focus1 be 'highest domain Level'
      Let its Refocus-Criterion be
        'complete solution in focus'
      Let its Refocus-Increment be 'Level below'
      Let its FocusN be 'lowest domain Level'
    Otherwise let its Goal be
         'Action-Level = successive time intervals'
      Let its Criterion be
        'complete plan at lowest domain Level'
      Let its Weight be 0.7
      Let its Rationale be
        'more than sufficient time for all requested tasks'
      Let its Focus1 be
        'task interval that begins at P's start'
      Let its Refocus-Criterion be
        'complete solution in Focus at the lowest domain Level'
      Let its Refocus-Increment be
        'first-empty-time-unit-after'
      Let its FocusN be
        'task interval that ends at P's finish'
  Let its Source be 'P'
  Let its Type be 'problem appropriate'
```

FIG. 9. Control knowledge source: Refine-or-Chain?

Chain? creates a simpler temporal-chaining Strategy decision: Generate a plan by incorporating tasks in the order in which they will be executed. Because this is not an especially powerful strategy, Refine-or-Chain? gives it an intermediate Weight, 0.7, and records its Rationale, 'more than sufficient time for all Requested-Tasks'. In either case, Refine-or-Chain? records the Strategy's Source, 'Problem1', and its Type, 'Problem-appropriate'. When triggered, Refine-or-Chain? generates a KSAR with the attributes defined in Table 2.

3.3.3. *An example of control planning*

Fig. 7 shows how four illustrative control knowledge sources might interact to develop a partial control plan incrementally:

(1) Accept-Problem, triggered by input requesting solution of a problem, *P*, creates a corresponding Problem decision.

(2) Refine-or-Chain?, triggered by creation of the Problem decision, determines that it is not feasible to perform all of the Requested-Tasks in the available plan execution time. It changes the Problem decision's Criterion 'all Requested-Tasks planned' to 'all important Requested-Tasks planned' and creates a Strategy decision representing the successive-refinement Strategy.

(3) Implement-Strategy, triggered by creation of the successive-refine Strategy, creates a Focus decision that implements the Strategy's Focus1, favoring actions at the Outcome Level.

(4) Recency-Policy, triggered by creation of the Problem decision, creates a Policy decision favoring actions triggered by recent blackboard events.

This example abstracts a simple, cooperative interaction among selected control KSARs out of a larger problem-solving context. In practice, other KSARs would compete with those discussed above, possibly assisting in the development of the partial control plan in Fig. 7, possibly interrupting or even terminating its development. Section 4 discusses some of these other behavior patterns. Appendix A presents program traces from OPM to illustrate some of these other behavioral patterns in detail.

3.4. Scheduling mechanism: basic control knowledge sources

The blackboard control architecture's scheduling mechanism comprises three domain-independent, basic control knowledge sources: Update-To-Do-Set, Choose-KSAR, and Interpret-KSAR. These knowledge sources operate at the To-Do-Set and Chosen-KSAR levels of the control blackboard, determining and documenting all of a system's feasible and actual problem-solving actions.

Because these three knowledge sources execute the basic control loop that runs a problem-solving system, they do not generate KSARs that compete with other domain and control KSARs for scheduling priority. Instead, they are invoked directly whenever their Conditions are satisfied. The specification of 'new' or 'changed' decisions in the Conditions of these knowledge sources

(discussed below) insures that each one will be invoked exactly once for any Condition-satisfying change to the blackboard. The three knowledge sources are coordinated so that Update-To-Do-Set's Action produces a blackboard modification that satisfies Choose-KSAR's Condition, Choose-KSAR's Action produces a blackboard modification that satisfies Interpret-KSAR's Condition, and Interpret-KSAR's Action produces a blackboard modification that satisfies Update-To-Do-Set's Condition. The three basic control knowledge sources are described below.

3.4.1. *Basic control knowledge source: Update-To-Do-Set*

Update-To-Do-Set (see Fig. 10) is invoked when the Status of the most recently created Chosen-Action changes to 'executed'. Its Action increments the problem-solving cycle and creates a new To-Do-Set, as follow.[4] Update-To-Do-Set

```
Name = 'Update-To-Do-Set'
Problem-Domain = (Any)
Description = 'Maintains To-Do-Sets of pending Actions'
Condition =
  The Status of the most recent Chosen-Action, K,
    changes to 'executed'
Condition-Vars = K
Action =
  Increment the current cycle by 1
  Create a To-Do-Set
  Let its Only-Cycle be the current cycle
  Let its Goal be the Goal of the most recent prior To-Do-Set
  Remove K from the new Goal's Invocable-List
  If K has Consequences
    Then for each knowledge source, KS, whose
        Problem-Domain = Problem1's Problem-Domain or 'Any'
      For each Event (one of K's Consequences)
        If KS's Trigger is true
          Then create a KSAR and add it to the Triggered-List
    For each KSAR on the Triggered-List
      For each KSAR Pre-Condition that is true
        Bind the variables in the Pre-Condition
      If all of the Pre-Conditions are true
        Then move the KSAR from the Triggered-List
          to the Invocable-List
  For each KSAR on the Triggered-List or the Invocable-List
    Compute and record the KSAR's Ratings against all operative
      Focus and Policy Goals
    Compute and record the KSAR's Priority by applying the
      Integration-Rule in Policy1 to the KSAR's Ratings
```

FIG. 10. Basic control knowledge source: Update-To-Do-Set.

[4] In practice, creating a new To-Do-Set on each cycle is not necessary for the architecture to function and it inefficiently consumes space. For this reason, both OPM and BB1 use a version of Update-To-Do-Set that simply updates a single To-Do-Set.

sets the new To-Do-Set's Only-Cycle to the current cycle. It initially sets the To-Do-Set's Goal to the Goal of the most recent prior To-Do-Set, removing the executed KSAR from its Invocable-List. It then examines all domain and control knowledge sources that specify the current Problem's Problem-Domain (or 'Any') and all blackboard events listed as the Chosen-Action's Consequences. For every unique pairing of a knowledge source and an event that satisfies the knowledge source's Trigger, Update-To-Do-Set creates a new KSAR and gives it the attributes defined in Table 2. Update-To-Do-Set adds each new KSAR to the new To-Do-Set's Triggered-List. It then evaluates the Pre-Conditions of KSARs on the Triggered-List and moves those whose Pre-Conditions are true to the Invocable-List. (The other KSARs simply remain on the Triggered-List.) Thus, Update-To-Do-Set evaluates the feasibility of all of the system's own potential actions except those of the basic control knowledge sources.

Update-To-Do-Set also computes and records each KSAR's Ratings against operative Focus and Policy decisions and its Priority. It computes a KSAR's Ratings by evaluating each operative Focus and Policy decision's Goal against the KSAR's Scheduling-Vals. It computes a KSAR's Priority by applying an integration rule (always recorded in Policy1) to the KSAR's Ratings. Because control knowledge sources can change Focus decisions, Policy decisions, and the integration rule, a KSAR's Ratings and Priority can also change from cycle to cycle.[5] The following examples illustrate the computation of Ratings and Priorities for the KSARs in Fig. 11.

Suppose that Policy1 contains the integration rule 'number of positive ratings on operative Focus and Policy decisions' and that the control blackboard contains one operative Policy decision—Policy2, whose Goal is 'KS-Importance $\geq$ 0.8'. KSAR-052's KS-Importance is only 0.7, so its Rating on Policy2 is 0 and its Priority is 0. KSAR-051's KS-Importance is 0.9, so its Rating on Policy2 is 100 and its Priority is 1.

Now suppose a control knowledge source records a new Policy decision—Policy3, whose Goal is 'Action-Blackboard = Control'. KSAR-052 satisfies Policy3, but not Policy2, so its new Priority is 1. KSAR-051 satisfies Policy2, but not Policy3, so its new Priority is 1.

Finally, suppose a control knowledge source changes Policy1's integration rule to 'sum of weighted ratings on operative Focus and Policy decisions'. Assume also that Policy2's Weight is 5 and Policy3's Weight is 10. KSAR-052's new Priority is $0 + (10 * 100) = 1000$. KSAR-051's new Priority is $(5 * 100) + 0 = 500$.

Update-To-Do-Set records each KSAR's Ratings and Priority on the control

[5] For efficiency, the version of Update-To-Do-Set used in OPM and BB1 does not recompute each KSAR's Ratings and Priority on each cycle, but simply updates its Ratings against changed Focus and Policy decisions and, if necessary, updates its Priority.

```
KSAR-052
  KS = 'Implement-Strategy'
  Triggering-Event = E-035
  Triggering-Cycle = 046
  Triggering-Decision = Problem-001
  Condition-Values =
    P = Problem-001
  Scheduling-Values =
    Action-Blackboard = 'Control'
    Action-Level = 'Focus'
    Trigger-Weight = 0.9
    KS-Importance = 0.7
    KS-Efficiency = 0.7
    KS-Credibility = 1.0
KSAR-051
  KS = 'Insert-Outcome'
  Triggering-Event = E-032
  Triggering-Cycle 044
  Triggering-Decision = Outcome-001
  Condition-Value =
    t = 'vet'
  Scheduling-Values =
    Action-Blackboard = 'Domain'
    Action-Level = 'Outcome'
    Trigger-Weight = 0.9
    KS-Importance = 0.9
    KS-Efficiency = 0.7
    KS-Credibility = 0.8
```

FIG. 11. Example To-Do-Set Invocable-List.

blackboard. As discussed below, Choose-KSAR uses this information to make scheduling decisions. In addition, higher-level control knowledge sources can inspect and reason about the implications of KSAR Ratings and Priorities.

3.4.2. *Basic control knowledge source: Choose-KSAR*

Choose-KSAR (see Fig. 12) is triggered by the creation of a new To-Do-Set. Its Action schedules a KSAR according to the current scheduling rule (always recorded in Policy1). For example, suppose the scheduling rule were 'highest-priority KSAR'.[6] Following the examples above, given the integration rule, 'number of positive ratings on operative Focus and Policy decisions', and Policy2, 'KS-Importance ≥ 0.8', KSAR-052's Priority is 0 and KSAR-051's Priority is 100. Choose-KSAR schedules KSAR-051. Given the addition of Policy3, 'Action-Blackboard = Control', both KSARs have Priority 100. Choose-KSAR must schedule one of them arbitrarily. Finally, given the new integration rule, 'sum of weighted ratings on operative Focus and Policy

[6] OPM uses a different, probabilistic scheduling rule, discussed in Appendix A.

```
Name = 'Choose-KSAR'
Description = 'Decides which pending action to perform next'
Problem-Domain = (Any)
Condition =
  There is a new To-Do-Set, A
Conditions-Vars = (A)
Action =
  Choose the KSAR, K, from A's Goal's Invocable-List
    that satisfies the Scheduling-Rule in Policy1
  If any of K's Pre-Conditions is not true
    Then unbind the variables in the Pre-Condition
      Move K from the Invocable-List to the Triggered-List
      Choose another KSAR
  Create a new Chosen-Action
  Let its Goal be K
  Let its Status be 'Scheduled'
  Let its Rationale be the Foci and Policies that favor it
  Let its Only-Cycle be the current cycle
```

FIG. 12. Basic control knowledge source: Choose-KSAR.

decisions', KSAR-052's Priority is 1000 and KSAR-051's Priority is 500. Choose-KSAR schedules KSAR-052.

Since Pre-Conditions specify potentially transient state information, a KSAR's Pre-Conditions may be invalidated by blackboard modifications occurring after Update-To-Do-Set moves it to the Invocable-List. Therefore, after making its choice, Choose-KSAR reevaluates the chosen KSAR's Pre-Conditions. If any of them is no longer true, Choose-KSAR unbinds the Pre-Condition variables, moves the KSAR back to the Triggered-List and chooses another KSAR. When it finds a KSAR whose Pre-Conditions are true, it creates a Chosen-Action whose Goal is the chosen KSAR, Rationale is the list of Foci and Policies that favor it, Status is 'Scheduled', and Only-Cycle is the current problem-solving cycle. Thus, Choose-KSAR schedules all of the system's actions except those of the three basic control knowledge sources.

3.4.3. *Basic control knowledge source: Interpret-KSAR*

Interpret-KSAR (see Fig. 13) is triggered by the creation of a new Chosen-

```
Name = 'Interpret-KSAR'
Description = 'Performs a chosen action'
Problem-Domain = 'Any'
Condition =
  There is a new Chosen-Action, K
Condition-Vars = (K)
Action =
  Interpret and execute K's Goal's Action
  Let K's Consequences be all resulting blackboard events
  Change K's Status to 'executed'
```

FIG. 13. Basic control knowledge source: Interpret-KSAR.

Action. Its Action interprets and executes the Action of the KSAR recorded as the Chosen-Action's Goal. It also records the Chosen-Action's Consequences (resulting blackboard events) and changes its Status to 'Executed'. Thus, it executes all of the system's actions execpt those of the three basic control knowledge sources.

3.4.4. *Character of the basic control loop*

As the example above illustrates, the blackboard control architecture's basic control loop is explicit and modifiable. Choose-KSAR has no *a priori* scheduling priorities for KSARs representing domain or control actions, actions at different levels of abstraction, or actions having any other specific attributes. Any KSAR attributes may or may not be recorded in KSARs for a given knowledge source and may or may not appear in the Goals of operative Foci and Policies on a given problem-solving cycle. Even the rule for integrating ratings against multiple heuristics is recorded (and modifiable) in Policy1. Update-To-Do-Set computes and records each KSAR's Ratings and Priority based on this dynamic state information recorded on the control blackboard. Choose-KSAR adapts its scheduling behavior to the current KSAR Ratings and Priorities and the current scheduling rule, which is itself recorded (and modifiable) in Policy1. Interpret-KSAR executes whatever KSAR Action is scheduled.

3.5. Important features of blackboard control architecture

(1) *The blackboard control architecture explicitly represents domain and control problems, knowledge, and solutions.*

The architecture accommodates domain problems posed by a system user or by the problem-solving system itself. It defines the control problem as a real-time planning problem whose solution determines what domain and control actions the problem-solving system should, can, and does perform at each point in the problem-solving process. The architecture specifies explicit domain and control knowledge sources to solve their respective problems on explicit domain and control blackboards. It accommodates domain blackboard structures specified by an application system designer. It specifies the structure of the control blackboard and a vocabulary for expressing control decisions and the relationships among them. Finally, it permits variable—arbitrarily large—grain-size heuristics at different levels of abstraction, but requires small grain-size KSAR actions for actual execution.

(2) *The blackboard control architecture integrates domain and control problem-solving in a single basic control loop.*

The architecture's three basic control knowledge sources (Update-To-Do-Set, Choose-KSAR, and Interpret-KSAR) manage the triggering, scheduling, and interpreting of all problem-solving actions, including both domain and control actions, in a uniform manner and with no *a priori* control biases.

(3) *The blackboard control architecture articulates, interprets, and modifies representations of its own knowledge and behavior.*

The architecture represents all domain and control knowledge sources, all solution elements generated for a domain problem, and all solution elements generated for the control problem as data structures, available for interpretation and modification. Its control knowledge sources operate directly on these representations.

(4) *The blackboard control architecture adapts its problem-solving knowledge, its application of that knowledge, and its basic control loop to dynamic problem-solving situations.*

The architecture *requires* only the three basic control knowledge sources: Update-To-Do-Set, Choose-KSAR, and Interpret-KSAR. These knowledge sources have no specific control knowledge of their own, but adapt to whatever control decisions are recorded by higher-level control knowledge sources on the control blackboard. Thus, their behavior and the system's overall problem-solving behavior are determined entirely by dynamic interactions among domain and control knowledge sources and the simultaneously evolving solutions to domain and control problems.

(5) *The blackboard control architecture realizes these features in a uniform mechanism.*

The architecture achieves all domain and control problem-solving with condition-action knowledge sources that respond to, generate, and modify solution elements in a global blackboard structure under the management of a single scheduling mechanism. Both domain and control problem-solving are characteristically incremental and opportunistic, entailing fortuitous, synergistic interactions among independently generated solution elements.

4. Behavioral Goals Revisited

This section shows how blackboard control systems (AI systems developed within the blackboard control architecture) achieve the behavioral goals set forth in Section 2. Each goal is illustrated with examples from the behavior of OPM [16], a blackboard system that performs multiple-task planning. Appendix A presents and analyzes detailed traces of OPM's problem-solving behavior. As indicated in the discussion, the current OPM system performs most, but not all of the example behaviors. However, all of the example behaviors could be implemented in BB1 [14], a domain-independent implementation of the blackboard control architecture.

(1) *A blackboard control system makes explicit control decisions that solve the control problem.*

The basic knowledge source Choose-KSAR decides which KSARs execute on each problem-solving cycle. Choose-KSAR bases its decisions on other

control decisions, made by other control knowledge sources, that characterize KSARs the system should and can execute during variable-length problem-solving time intervals. The traces in Appendix A contain many examples of control decision-making.

(2) *A blackboard control system decides what actions to perform by reconciling independent decisions about what actions are desirable and what actions are feasible.*

Higher-level control knowledge sources make Problem, Strategy, Focus, and Policy decisions about actions the system should perform, thereby establishing its operative control heuristics. The basic knowledge source Update-To-Do-Set makes To-Do-Set decisions about actions the system can perform, thereby identifying triggered knowledge sources. The basic knowledge source Choose-KSAR makes Chosen-Action decisions, taking into account prior decisions at all higher levels of the control blackboard, to determine what actions the system does perform. The traces in Appendix A provide many examples of all three kinds of decisions and their interactions.

As implemented, OPM balances the desirability and feasibility of potential actions. However, it could drive the problem-solving process in either direction by introducing appropriate control knowledge sources.

Suppose OPM has created a Strategic Focus on the Design level, but its current To-Do-Set contains no invocable KSARs whose actions occur at that level. In other words, Design level actions are desirable, but not feasible. In that situation, OPM needs to perform actions that enable it to execute Design level KSARs. Specifically, it needs to perform actions whose blackboard modifications satisfy the Triggers and Pre-Conditions of knowledge sources whose Actions occur at the Design level. A control knowledge source, Enable-KS-Focus, could create a new Focus decision favoring such actions. In the present example, Enable-KS-Focus would be triggered by the creation of a highly weighted Focus decision favoring KSARs at the Design level in the context of a To-Do-Set containing no invocable KSARs at the Design level. Enable-KS-Focus would examine OPM's knowledge sources whose Actions occur at the Design level and determine that task-location information is necessary to satisfy their Pre-Conditions. Enable-KS-Focus would then create a new Focus decision favoring KSARs whose Actions generate task-location information. Obviously, this enabling process could regress further.

Alternatively, suppose OPM's recent To-Do-Sets contain several KSARs triggered by time constraints, which is interesting because the Problem's Criterion requires OPM to honor time constraints, but that it has not decided that it should perform such actions. In other words, these actions are interesting and feasible, but not explicitly desirable. In that situation, OPM needs to perform actions that incline it to execute KSARs triggered by time constraints. Specifically, it needs to perform an action that establishes a Focus decision

favoring the desired actions. A control knowledge source, Exploit-Feasibility, could perform this function. In the present example, Exploit-Feasibility would be triggered by the creation of To-Do-Sets containing many KSARs triggered by time constraints (an interesting attribute) in the context of no Focus (or Policy) decision favoring such actions. Exploit-Feasibility would create a new Focus decision favoring KSARs triggered by time constraints. Exploit-Feasibility could be parameterized to assess different degrees of interestingness and create Focus decisions with correspondingly different Weights.

(3) *A blackboard control system adopts variable grain-size control heuristics.*

Control knowledge sources operate at different levels of abstraction, establishing operative control heuristics of different grain-sizes. Problem decisions have the largest grain-size. They prescribe any actions appropriate for the problem domain and they remain operative for the duration of the problem-solving process. For example, OPM's Problem decisions prescribe any actions appropriate for the multiple-task planning domain. Decisions in the Strategy-Focus hierarchy have successively smaller grain-sizes. Strategy decisions prescribe sequences of different kinds of problem-solving actions and they remain operative for relatively long problem-solving time intervals. Focus decisions prescribe particular kinds of actions and they remain operative for relatively short problem-solving time intervals. For example, the successive-refinement Strategy used by OPM prescribes actions at successive domain levels and remains operative until the Problem is solved. One of the Focus decisions that implements that Strategy prescribes actions at the Design level and remains operative only until the Design level plan is complete. In contrast to the Strategy-Focus hierarchy, Policy decisions prescribe particular kinds of actions, but they usually remain operative for long problem-solving time intervals. For example, OPM usually records a Policy favoring recently triggered KSARs early in the problem-solving process and that Policy usually remains operative for the duration of the problem-solving process. To-Do-Set and Chosen-Action decisions have the smallest grain-size, identifying specific KSARs whose actions can be and are performed on specific problem-solving cycles. The OPM trace in Appendix A contains many examples of decisions of different grain-sizes.

(4) *A blackboard control system adopts control heuristics that focus on whatever action attributes are useful in the current problem-solving situation.*

Different control heuristics may refer to attributes of a KSAR's knowledge source, triggering information, or solution context. For example, as discussed in Appendix A, OPM creates Policies favoring KSARs with reliable, important, and efficient knowledge sources. It creates Policies favoring KSARs with recent triggering events. It creates Focus decisions favoring KSARs whose Actions occur at specific blackboard levels and in specific plan intervals.

(5) *A blackboard control system adopts, retains, and discards individual control heuristics in response to dynamic problem-solving situations.*

Because a blackboard control system treats control as a real-time planning problem, it naturally adapts its operative control heuristics to dynamic problem-solving situations. Higher-level control knowledge sources may, in principle, create or modify control heuristics at any point in the problem-solving process—namely, whenever (a) their Conditions are satisfied and (b) Choose-KSAR schedules them. In practice, some knowledge sources may be triggered by events that occur a predictable number of times or at predictable points in the problem-solving process. For example, OPM's control knowledge sources, defined in Appendix A, include some that are triggered exactly once on predictable problem-solving cycles, some that are triggered exactly once on unpredictable problem-solving cycles, and some that can be triggered repeatedly on unpredictable problem-solving cycles.

As illustrated in Appendix A, OPM's current control knowledge sources dynamically establish operative control heuristics and change each one's Status to 'inoperative' when its Criterion is satisfied. In addition, control knowledge sources could modify control heuristics in other situations. For example, a control knowledge source, Reject-Heuristic, could be triggered by situations in which an operative control heuristic influences Choose-KSAR's behavior with no corresponding improvement in the problem solution. Reject-Heuristic would change the ineffective heuristic's status to 'inoperative'.

(6) *A blackboard control system decides how to integrate multiple control heuristics of varying importance.*

Independent control knowledge sources create and assign different Weights to simultaneously operative, conflicting or complementary, heuristics on the control blackboard. The trace in Appendix A contains numerous examples of simultaneously operative control heuristics. The basic control knowledge sources (Update-To-Do-Set, Choose-KSAR, and Interpret-KSAR) integrate simultaneously operative heuristics at the Focus and Policy levels to make scheduling decisions. The 'integration rule' and 'scheduling rule' they use are, themselves, recorded and modified by control knowledge sources on the control blackboard. For example, OPM's knowledge source Planning-Policies records a probabilistic scheduling rule. Under this rule, the probability that Choose-KSAR schedules any particular pending action on a particular cycle is proportional to the action's Weighted ratings on operative Focus and Policy decisions (discussed in Appendix A). By contrast, corresponding knowledge sources for HEARSAY-II and HASP record a deterministic scheduling rule under which Choose-KSAR always schedules the pending action with the highest Priority, determined by the recorded integration rule to be the sum of weighted ratings on operative Focus and Policy decisions (discussed in Appendix B).

(7) *A blackboard control system dynamically plans useful sequences of actions.*

A blackboard control system plans strategic sequences of actions with control knowledge sources that create Strategy decisions and then implement them as prescribed series of Focus decisions. The generic knowledge source Implement-Strategy incrementally implements successive strategic Focus decisions in response to dynamic problem-solving situations. Each Focus decision influences Choose-KSAR's scheduling decisions on subsequent problem-solving cycles. For example, the discussion of OPM's 'Strategic Top-Down Domain Planning' in Appendix A shows in detail how two control knowledge sources, Refine-Or-Chain? and Implement-Strategy, cooperate to create and implement the successive-refinement strategy.

A blackboard control system also interrupts, resumes, and terminates strategic action sequences in response to dynamic problem-solving situations, as discussed below.

A blackboard control system interrupts strategic action sequences in two ways. First, it interrupts its adherence to a strategic Focus when Choose-KSAR, adapting to conflicting Focus and Policy decisions, schedules a KSAR that does not match the strategic Focus. For example, in the discussion of OPM's 'Opportunistic Bottom-Up Domain Planning' in Appendix A, Choose-KSAR interrupts a strategic sequence of actions whose current Focus is at the Operation level to schedule Outcome and Procedure level actions that satisfy an 'Action-Importance' Policy. Second, a blackboard control system interrupts its transition between sucessive strategic Focus decisions when Choose-KSAR schedules competing KSARs in favor of those that update the strategic Focus. For example, Choose-KSAR might not decide to schedule a pending Implement-Strategy KSAR. In either case, OPM interrupts a planned strategic sequence to perform one or more non-strategic actions that have other desirable attributes.

A blackboard control system resumes interrupted strategic sequences whenever Choose-KSAR decides that pending strategic actions are preferable to non-strategic alternatives. The blackboard representation of an evolving control plan supports strategy resumption by preserving strategic context—the original Strategy decision, all previously implemented strategic Focus decisions, and the To-Do-Set of pending KSARs. The discussion of 'Opportunistic Bottom-Up Domain Planning' in Appendix A shows how OPM resumes its strategic Operation level actions following completion of the interrupting Outcome and Procedure level actions mentioned above.

Of course, with Strategy interruption, intervening events might obviate pending strategic actions. A blackboard control system can respond appropriately to this kind of situation. For example, suppose Choose-KSAR interrupts its successive-refinement Strategy just before Implement-Strategy creates the Design level Focus. Suppose also that the interrupting actions

create a complete Design level plan. Subsequently resuming the successive-refinement Strategy, Choose-KSAR schedules the pending Implement-Strategy KSAR, which creates the unnecessary Design Focus. However, the Design Focus's Criterion is satisfied as soon as it is created, immediately re-triggering Implement-Strategy. When Choose-KSAR schedules the new Implement-Strategy KSAR, it declares the Design Focus 'inoperative' and creates the subsequent Procedure level Focus. If it is very important for Strategic Focus decisions to be implemented promptly and accurately, Implement-Strategy can be given a KS-Importance value that is substantially higher than those of other knowledge sources or the integration rule or scheduling rule can explicitly favor it. OPM has a moderately weighted Policy favoring all control knowledge sources.

Sometimes intervening events change the results of previous strategic actions, requiring the system to back up and redo parts of a strategic plan. For example, suppose OPM interrupts strategic actions at the Operation level to perform actions that substantially revise its Outcome level plan. It should not resume the interrupted actions to continue refining the superseded Outcome level plan at the Operation level. It should back up to the Design level and begin refining the revised Outcome level plan. It could achieve this behavior through the actions of additional control knowledge sources that revise the state of the control plan. To illustrate, the knowledge source Back-Up might be triggered by substantial changes to partial solutions that satisfied the Criteria of preceding strategic Focus decisions. In the present example, Back-Up would be triggered by changes to the Outcome level solution because it satisfied the Criterion of the preceding strategic Focus on the Outcome level. Back-Up's action would declare the current strategic Focus 'inoperative' and create an intermediate Strategic Focus corresponding to the earliest strategic Focus whose Criterial partial solution changed. In the present example, Back-Up would declare the Operation level Focus 'inoperative' and create a new version of the Outcome level Focus. Implement-Strategy would be triggered immediately by satisfaction of the Outcome Focus Criterion and generate a new Design level Focus.[7]

Finally, some intervening events obviate the interrupted Strategy itself. For example, OPM might adopt a temporal-chaining strategy and subsequently decide that there is insufficient time to perform all Requested-Tasks. It should abandon that strategy (but preserve the domain plan generated so far) in favor of a new strategy. OPM could achieve this behavior with a new control knowledge source, Reject-Strategy, that notices when an operative Strategy's

[7] Although OPM currently does not include the control knowledge source, Back-Up, its Policy favoring recently triggered KSARs *incidentally* achieves the same objective. Choose-KSAR schedules KSARs that refine a recently created Outcome decision in favor of those that refine a previously created Outcome decision.

Rationale is invalidated and declares both the Strategy and all of the Focus decisions that implement it 'inoperative'. If the To-Do-List contains pending Implement-Strategy KSARs for the abandoned Strategy and Choose-KSAR subsequently schedules them, they will fail Choose-KSAR's final validation of the Pre-Condition requiring that the triggering Strategy be operative (see Section 3.4.2).

Once a control heuristic has been tried and abandoned, the system ordinarily should not try it again without making an explicit decision that the heuristic is, after all, worthwhile. In other words, it should not make the same mistake twice. The control blackboard's preservation of inoperative (formerly operative) heuristics supports this kind of behavior. Any knowledge source that creates new heuristics can make its action contingent upon there being no redundant heuristic, operative or inoperative, on the blackboard. For a more sophisticated approach, the knowledge sources that declare heuristics 'inoperative' could also declare their Post-Mortems, justifying the change in Status. Knowledge sources that considered creating redundant heuristics could use that information in deciding whether or not to do so.

(8) *A blackboard control system reasons about the relative priorities of domain and control actions.*

Event-driven, condition-action knowledge sources operate on a structured blackboard to generate solution elements for both domain and control problems. Reasoning about the relative priorities of different domain actions, different control actions, and domain versus control actions is uniform. Thus, Update-To-Do-Set integrates pending domain and control KSARs in a single To-Do-Set. Choose-KSAR schedules KSARs from the To-Do-Set based on operative Foci and Policies. Because the architecture has no *a priori* biases regarding the relative priorities of domain and control actions, operative Foci and Policies may or may not include heuristics that distinguish domain and control KSARs. Further, all operative heuristics are, themselves, established through the actions of previously executed control KSARs. Finally, Interpret-KSAR executes the actions of all scheduled domain and control KSARs. The OPM trace in Appendix A includes several illustrations of the architecture's uniform treatment of domain and control actions.

5. Relationship to Other Work

The first section below compares the blackboard control architecture with three alternative control architectures. The second section places it in an evolutionary progression of control architectures.

5.1. Comparison with alternative architectures

This section compares the blackboard control architecture with three alternatives: the standard blackboard architecture with a sophisticated scheduler,

the standard blackboard architecture with solution-based focusing, and meta-level architecture. Each of these alternatives shares certain features with the blackboard control architecture and achieves some of the behavioral goals set forth in Section 2. However, as summarized in Table 4, only the blackboard control architecture achieves all of the behavioral goals.

Three caveats are in order.

First, it is important to distinguish between an architecture's theoretical entailments and its computational sufficiency. For example, the blackboard control architecture explicitly entails dynamic control heuristics (goal 5). Thus, the architecture is not only computationally sufficient to achieve goal 5; it does so in a theoretically natural and computationally elegant manner. In contrast, meta-level architecture entails a fixed repertoire of control heuristics. Although some implementations are computationally sufficient to achieve goal 5, they do so in a theoretically anomalous and computationally awkward manner. The discussion below ignores computational sufficiency, focusing instead upon each architecture's theoretical entailments.

Second, other scientists may question the set of behavioral goals used to evaluate alternative architectures. Section 2 presents arguments for the necessity, but not the sufficiency, of these goals for intelligent systems. The alter-

TABLE 4. Achievement of behavioral goals by alternative architectures

	Alternative control architectures and representative systems			
Behavioral goals	Sophisticated scheduler HEARSAY-II	Solution-based focusing HASP	Meta-level architecture MOLGEN	Blackboard control architecture OPM
(1) Make explicit decisions that solve the control problem.	+	+	+	+
(2) Reconcile desirability and feasibility of actions.	–	–	–	+
(3) Adopt variable grain-size control heuristics.	–	–	–	+
(4) Adopt heuristics that focus on useful action attributes.	+	–	+	+
(5) Dynamically adopt, retain, and discard control heuristics.	–	–	–	+
(6) Decide how to integrate multiple control heuristics.	–	–	–	+
(7) Dynamically plan strategic action sequences.	–	–	–	+
(8) Reason about control vs. domain actions.	–	–	–	+

native architectures discussed below may surpass the blackboard control architecture in their achievement of other important goals not considered in this paper.

Third, the discussion below omits important research efforts that address related issues, but do not propose specific AI architectures. Notable examples are Smith's and Rivieres's work on reflection in 3-LISP [28] and Wilensky's work on meta-planning [33].

5.1.1. *The sophisticated scheduler*

Many basic blackboard systems attack the control problem with a sophisticated scheduler [4, 7, 18, 21]. Like the basic control knowledge source Choose-KSAR, the sophisticated scheduler selects one of a set of pending KSARs on each problem-solving cycle. Unlike Choose-KSAR, which has no specific control knowledge and simply adapts to operative control decisions on the control blackboard, the sophisticated scheduler is a complex program embodying all of a system's control knowledge. For example, the HEARSAY-II scheduler [7] incorporates control heuristics favoring KSARs with efficient and reliable knowledge sources, credible triggering information, and credible solution contexts. On each problem-solving cycle, the scheduler chooses the pending KSAR that has the highest combined ratings against all of these heuristics. (Appendix B discusses HEARSAY-II and its control behavior in more detail.)

The sophisticated scheduler is an innovative and powerful mechanism for achieving considerable flexibility and opportunism in complex problem-solving processes. However, as indicated in the first column in Table 4, the sophisticated scheduler achieves only behavioral goals 1 and 4. It fails to achieve goal 2 because it cannot pursue recursive sub-goals of its control heuristics and it cannot generate new heuristics favoring feasible actions. It fails to achieve goal 3 because all of its heuristics prescribe actions of a fixed grain-size, namely individual KSARs. It fails to achieve goal 5 because its scheduling heuristics are implicit and not modifiable. It fails to achieve goal 6 because its integration rule (sum of weighted ratings) and its scheduling rule (highest-priority KSAR) are also implicit and not modifiable. It fails to achieve goal 7 because it makes only instantaneous scheduling decisions, cannot plan sequences of actions, and has no mechanism for maintaining, modifying, and incorporating strategic context over extended problem-solving intervals. It fails to achieve goal 8 because it does not reason about or even perform control actions.

Responding to some of the same behavioral limitations described above, other researchers have introduced improvements to the sophisticated scheduler, as discussed below.

Lesser and Corkill [21] integrated the sophisticated scheduler with a planning component to coordinate sequences of KSARs that satisfy high-priority problem-solving goals. Although they considered implementing the planning component, itself, as a blackboard system, they did not actually do so. Instead, their planning component embodies specific strategic heuristics. By

contrast, the blackboard control architecture implements dynamic planning capabilities within a larger, uniform blackboard architecture for domain and control problem-solving.

Hudlicka and Lesser [18] propose the fault detection/diagnosis (FDD) method to modify deficient strategies during problem-solving. The FDD method detects suspicious or undesirable situations, diagnoses which scheduling parameters produced the undesirable situation, and modifies those parameters to improve problem-solving performance. In the context of the blackboard control architecture, the FDD method defines a class of extremely sophisticated control knowledge sources for analyzing the impact of operative control decision and modifying those decisions when necessary. These knowledge sources would operate along with other control knowledge sources on the control blackboard.

Erman, London, and Fickas [8] developed HEARSAY-III, a uniform blackboard environment with separate domain and control blackboards and a default low-level scheduler that simply schedules any pending KSAR. Although these researchers intended HEARSAY-III to support general control problem-solving, it provides no theory of effective control problem-solving and no specific mechanism for implementing it. The blackboard control architecture is complementary to HEARSAY-III. It provides a theory of effective control behavior (the behavioral goals that motivate it) and a mechanism (its blackboard structure and vocabulary, higher-level control knowledge sources, and three basic control knowledge sources). The blackboard control architecture could be implemented in HEARSAY-III.

Appendix B shows how the blackboard control architecture replicates the behavior of the prototypical sophisticated scheduler used in HEARSAY-II. The architecture explicates important control reasoning done by HEARSAY-II's system builders, but not explicit in the system they built. It produces a control plan that is easy to comprehend, analyze, and compare to other control plans. It illuminates control heuristics that are potentially useful in other problem domains.

5.1.2. *Solution-based focusing*

Many basic blackboard systems attack the control problem with solution-based focusing [25, 26, 30]. Like the sophisticated scheduler, solution-based focusing relies upon a complex program that embodies all of a system's control knowledge. Unlike the sophisticated scheduler (and the blackboard control architecture), solution-based focusing does not identify or select among pending KSARs. Instead, it sequentially selects specific blackboard events and executes knowledge sources triggered by each one. In some implementations, all triggered knowledge sources execute in a pre-determined sequence; in others, the focusing program uses other aspects of the current solution to determine which subset of triggered knowledge sources execute and in what

order. For example, the HASP control program [26] operates as follows:

1. For each currently due clock event, selected in last-in-first-out order, execute the pre-determined sequence of triggered knowledge sources associated with the event's type.
2. For each expected event whose specified blackboard events have occurred, selected in last-in-first-out order, execute the pre-determined sequence of triggered knowledge sources associated with the event's type.
3. For the most recent simple event, execute the pre-determined sequence of triggered knowledge sources associated with the event's type.
4. Repeat 2 and 3 until all expected events and simple events are processed.
5. Repeat 1–4.

(Appendix B discusses HASP and its behavior in more detail.)

Solution-based focusing is an efficient mechanism for achieving a moderate amount of flexibility in complex problem-solving processes. However, as indicated in the second column in Table 4, solution-based focusing achieves only behavioral goal 1. It fails to achieve goal 2 because it cannot pursue recursive sub-goals of its control heuristics and it cannot generate new control heuristics favoring feasible actions. It fails to achieve goal 3 because all of its heuristics prescribe actions of a fixed grain-size, namely pre-specified sets of knowledge sources. It fails to achieve goal 4 because its heuristics prescribe actions according to attributes of their triggering events, but disregard attributes of their knowledge sources and solution contexts. It fails to achieve goal 5 because its operative scheduling heuristics are implicit and not modifiable. It fails to achieve goal 6 because its integration rule (sum of weighted ratings) and its scheduling rule (highest-priority KSAR) are also implicit and not modifiable. It fails to achieve goal 7 because it executes a pre-determined control plan (sometimes with local tuning), rather than dynamically constructing and modifying a control plan in the course of problem-solving. It fails to achieve goal 8 because it does not reason about or even perform control actions.

Appendix B shows how the blackboard control architecture replicates the prototypical solution-based focusing procedure in HASP. The architecture explicates important control reasoning obscured in HASP's control program. It produces a control plan that is easy to comprehend, analyze, and compare to other control plans. It illuminates control heuristics that are potentially useful in other problem domains.

5.1.3. *Meta-level architecture*

Meta-level architecture [2, 3, 5, 6, 10, 19, 20, 29, 32] distinguishes domain actions and meta-level actions, which select among or otherwise operate on

domain actions. Similarly, meta-rules perform meta-level actions. Continuing the progression, meta-meta-level actions operate on meta-level actions, and so forth. In addressing the control problem, meta-level architectures execute all actions at a given meta-level to identify the ideal action to execute at the next lower level. Thus, the basic control loop is: (a) execute all meta-level actions; (b) execute the ideal domain action. With additional meta-levels of knowledge, the loop expands recursively, for example: (a) execute all meta-level actions as follows: (a1) execute all meta-meta-level actions; (a2) execute the ideal meta-level action; (b) execute the ideal domain action.

Because of the use of common terms and the resulting potential for confusion, it is important to distinguish between: (a) meta-rules and decisions versus control knowledge sources and decisions; and (b) meta-level versus levels of the control blackboard.

First, because control knowledge sources make decisions that refer to pending KSARs, all of them operate at one or more meta-levels. Control knowledge sources whose decisions refer exclusively to domain KSARs operate at the lowest meta-level. For example, Identify-Gaps-To-Fill (see Appendix A) makes a meta-level decision to favor domain KSARs whose Action-Interval (or Action-Level) is a selected region on the domain blackboard. However, some control knowledge sources make decisions that refer to control KSARs or to both domain and control KSARs. For example, Planning-Policies (see Appendix A) establishes general scheduling criteria (e.g., KS-Efficiency = high) that refer to all domain and control KSARs. When applied to domain KSARs, its decision operates at the lowest meta-level. When applied to some control KSARs, such as a pending KSAR for Identify-Gaps-To-Fill, its decision operates at the meta-meta-level. When applied to control KSARs whose own effects are at meta-meta or higher levels, its decision operates at still higher meta-levels. Thus, *many control knowledge sources make decisions whose meta-levels are functions of their current applications* (see also [20]). In general, although all control knowledge sources and decisions operate at *some* meta-level(s), there is no fixed mapping of meta-levels onto control knowledge sources.

Second, meta-level and level of the control blackboard are orthogonal. For example, a Focus decision might prescribe domain KSARs whose Action-Level is Outcome, while a Policy decision might prescribe domain KSARs whose Triggering-Decisions are credible. Although both are meta-level decisions, their difference in scope (temporary versus permanent) places them at different blackboard levels. Conversely, decisions whose effects occur at different meta-levels may appear at a single blackboard level. For example, a meta-meta-level decision favoring KSARs that refer to the Action-Intervals of domain KSARs might also appear at the Policy level. In fact, as illustrated above with Planning-Policies and its decisions, a single decision at a single blackboard level may have effects at multiple meta-levels. Thus, for the blackboard control architecture, *meta-level is not a defining theoretical construct, but simply another*

potentially useful attribute of KSARs and the decisions they generate, available for consideration during scheduling.

Meta-level reasoning architectures provide a perspicuous mechanism for explicitly representing control (meta-level) knowledge and its application during problem-solving. However, as indicated in the third column of Table 4, meta-level reasoning achieves only behavioral goals 1 and 4. It fails to achieve goal 2 because it cannot pursue recursive sub-goals of its control heuristics (but see [19]) and it cannot generate new heuristics favoring feasible actions. It fails to achieve goal 3 because all its heuristics prescribe actions of a fixed grain-size, namely individual rules. It fails to achieve goal 5 because it applies its entire repertoire of meta-level rules on every pass through the basic control loop. It fails to achieve goal 6 because, although it can differentially weight rules within a meta-level, its basic control loop embodies a priority system favoring rules at higher meta-levels. It fails to achieve goal 7 because it has no mechanism for maintaining, modifying, and incorporating strategic context over extended problem-solving intervals. It fails to achieve goal 8 because it does not reason about domain versus control actions; its basic control loop simply alternates sets of control (meta-level) actions with single domain actions.

The blackboard control architecture can replicate the control behavior of meta-level systems. For example, NEOMYCIN's meta-level reasoning [12] implements a procedure represented as an aggregation hierarchy of actions planned to occur at sequential stages of the problem-solving process. Its organization mirrors the Strategy-Focus hierarchy on the control blackboard. The blackboard control architecture could incorporate control knowledge sources to establish and implement NEOMYCIN's procedure.

5.2. Evolutionary perspective

Despite its behavioral innovations, the blackboard control architecture is not a radical departure from previous control architectures. Rather, it represents a stage in the evolution of control architectures, each of which instantiates a simple three-step process:

1. Identify the set of permissible next computations.
2. Select the next computation from among the permissible computations.
3. Execute the selected computation.

As discussed below, successive architectures, including the von Neumann architecture, the production system architecture, the basic blackboard architecture with a sophisticated scheduler, and the blackboard control architecture, elaborate on their predecessors by introducing additional knowledge and intelligence into each of the three steps in the control process.

First consider the classic von Neumann architecture:

1. Advance (load, then increment) the instruction counter.
2. Fetch the next instruction.
3. Execute the fetched instruction.

In Step 1, the von Neumann architecture identifies a set of exactly one permissible next computation with a two-part operation, advance the instruction counter: (a) load the current contents of the instruction counter (an address); and (b) increment the instruction counter. In Step 2, it selects the next computation by fetching the contents of the loaded address. In Step 3, it executes the selected computation, a single instruction. These steps define a sequential programming environment with branching whenever an executed instruction modifies the contents of the instruction counter.

Next consider the production system architecture:

1. Form the conflict set.
2. Resolve the conflict and select a rule.
3. Execute the selected rule.

In Step 1, the production system identifies a set of n permissible next computations as the set of productions whose left-hand sides evaluate to true. In order to identify this set, the system uses knowledge of the rules in the system's repertoire and the semantics of simple pattern matching. In Step 2, it selects the next computation by resolving the conflict among rules in the set and selecting one of them. In order to perform this selection, the system uses knowledge of the priorities of individual rules and sometimes additional data in working memory. In Step 3, it executes the selected computation, the right-hand side of a production rule, which ordinarily comprises a program of instructions. These innovations define a pattern-directed programming environment.

Now consider the basic blackboard architecture with a sophisticated scheduler:

1. Update the To-Do-Set.
2. Schedule a Pending KSAR.
3. Execute the Scheduled KSAR.

In Step 1, the sophisticated scheduler identifies a set of n permissible computations as the set of pending knowledge source activations: unique combinations of knowledge sources and triggering conditions. The triggering of knowledge sources can entail arbitrary computations, complex pattern matching, and the binding of contextural variables for incorporation into the resulting KSARs. In Step 2, the scheduler selects the next computation by scheduling one of the pending KSARs. In making these decisions, it uses knowledge of triggering event characteristics, knowledge source characteristics, problem characteristics, scheduling heuristics, and solution state. In Step 3, it executes the next computation, a knowledge source action, which ordinarily has the

computational power of a program of production rules. These innovations permit programs comprising variable-sized chunks of knowledge and increased potential for variability and fine-tuning in the application of knowledge. In particular, they permit multiple, possibly conflicting scheduling criteria and intelligent flexibility in program performance.

Finally, consider the blackboard control architecture. It uses the same three steps as the sophisticated scheduler, but introduces several innovations. In Step 1, it generates an expanded To-Do-Set of permissible computations, including both triggered and invocable domain and control KSARs, and records it on the control blackboard. In Step 2, it adapts to dynamic scheduling criteria, recorded on the control blackboard, in order to select one of the pending KSARs and it records the selected KSAR on the control blackboard. In Step 3, it executes the next computation, which may be either a domain action or a control action whose consequences reflexively influence Step 2, and again it records the consequences on the control blackboard. These innovations provide a flexible environment for explicit, dynamic control planning; a perspicuous, interpretable representation of the actual control plan; and a uniform mechanism for integrating domain and control problem-solving.

6. Strengths and weaknesses of the blackboard control architecture

The blackboard control architecture provides extreme flexibility in system behavior. Blackboard control systems can follow rigorous procedural strategies. They can coordinate implementation of successive strategic phases with problem-specific situations. They can interrupt, resume, or terminate adopted strategies. They can adopt variable subsets of simultaneously applicable strategic and non-strategic heuristics. They can use variable rules for integrating adopted heuristics and choosing among pending problem-solving actions. In sum, blackboard control systems adapt to complex control plans whose operative strategies, heuristics, and integration and scheduling rules can change repeatedly in the course of problem-solving.

Further, the blackboard control architecture places all of the parameters governing system behavior under system control. Blackboard control systems determine what strategies to follow. They determine when to implement successive strategic phases. They determine when to interrupt, resume, or terminate strategies. They determine which heuristics apply in the current problem-solving situation and which ones to adopt. They determine the rule by which to integrate simultaneously applicable heuristics and the rule by which to schedule pending problem-solving actions. In sum, they determine the strategies, heuristics, and integration and scheduling rules that define their own control plans.

Thus, blackboard control systems do not specify 'complete' control pro-

cedures or 'correct' combinations of control heuristics. They do not attempt to enumerate (a necessarily small number of) important problem-solving contingencies. Instead, they dynamically construct and modify situation-specific control plans out of modular control heuristics, adapting their problem-solving behavior to a wide range of unanticipated problem-solving situations.

The blackboard control architecture's greatest weakness is its high overhead, combining computational and storage costs. Given today's hardware and operating systems, if one's goal is to build high-performance application systems, the blackboard control architecture is probably inappropriate. Such systems may benefit from intelligent control mechanisms, but they are bound by severe computational and storage constraints. Because the blackboard control architecture exacerbates, rather than ameliorates these problems, it would ordinarily be too inefficient for a final application system.

On the other hand, the blackboard control architecture offers an extremely flexible and modular environment for experimental development of application systems prior to reimplementation in a high-performance environment. The architecture permits system builders to introduce, modify, and remove control knowledge sources independently. It permits them to exploit previous system development efforts by transferring and combining control knowledge sources from different application systems. If the control knowledge sources configured for an application system entail a relatively simple control plan that is effective for all of the specific problems the application system must solve, the system builder can compile them in a more efficient, run-time control procedure.

Returning to the motivations set forth in the introduction to this paper, the blackboard control architecture provides a framework and mechanism for intelligent control behavior. Blackboard control systems do not simply solve problems. They know something about how they solve problems, why they perform one problem-solving action rather than another, and what problem-solving actions they are likely to perform in the future. They use this knowledge to adapt their behavior to the demands of dynamic problem-solving situations. Research in progress exploits this knowledge to produce intelligent explanation and learning behaviors.

ACKNOWLEDGMENT

This work was supported by a DARPA grant to the Heuristic Programming Project at Stanford University and by an ONR contract to the Rand Corporation. Lee Erman, Frederick Hayes-Roth, Perry Thorndyke, and an anonymous reviewer provided helpful comments on an earlier version of the manuscript. Special thanks to Mark Stefik for editorial assistance and to Edward Feigenbaum for sponsoring the work.

REFERENCES

1. Barnett, J.A. and Erman, L.D., Making control decisions in an expert system is a problem-solving task, Tech. Rept., USC/Information Sciences Institute, 1982.

2. Clancey, W.J., The advantages of abstract control knowledge in expert system design, Tech. Rep. HPP-83-17, Stanford University, Stanford, CA, 1983.
3. Clancey, W.J., Acquiring, representing, and evaluating a competence model of diagnostic strategy, Tech. Rept. HPP-84-2, Stanford University, Stanford, CA, 1984.
4. Corkill, D.D. and Lesser, V.R., A goal-directed Hearsay-II architecture: Unifying data-directed and goal-directed control, in: *Proceedings National Conference on Artificial Intelligence*, Pittsburgh, PA, 1982.
5. Davis, R., Applications of meta level knowledge to the construction, maintenance, and use of large knowledge bases, Tech. Rept. Memo AIM-283, Stanford University, Artificial Intelligence Laboratory, Stanford, CA, 1976.
6. Doyle, J., A truth maintenance system, *Artificial Intelligence* **15** (1980) 179–222.
7. Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R., The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys* **12** (1980) 213–253.
8. Erman, L.D., London, P.E. and Fickas, S.F., The design and an example use of Hearsay-III, in: *Proceedings Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC (1981) 409–415.
9. Genesereth, M.R. and Lenat, D.B., Self-description and -modification in a knowledge representation language, Tech. Rept. HPP-80-10, Stanford University, Stanford, CA, 1980.
10. Genesereth, M.R. and Smith, D.E., Meta-level architecture, Tech. Rept. HPP-81-6, Stanford University, Stanford, CA, 1982.
11. Hanson, A. and Riseman, E., VISIONS: A computer system for interpreting scenes, in: A. Hanson and E. Riseman (Eds.), *Computer Vision Systems* (Academic Press, New York, 1978).
12. Hasling, D.W., Clancey, W.J. and Rennels, G., Strategic explanations for a diagnostic consultation system, Tech. Rept. STAN-CS-83-996, Stanford University, Stanford, CA, 1983.
13. Hayes-Roth, B., Flexibility in executive processes, Tech. Rept. N-1170-ONR, Rand Corporation, Santa Monica, CA, 1980.
14. Hayes-Roth, B., BB1: An architecture for blackboard systems that control, explain, and learn about their own behavior, Tech. Rept. HPP-84-16, Stanford University, Stanford, CA, 1984.
15. Hayes-Roth, B. and Hayes-Roth, F., A cognitive model of planning, *Cognitive Sci.* **3** (1979) 275–310.
16. Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S. and Cammarata, S., Modelling planning as an incremental, opportunistic process, in: *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan (1979) 375–383.
17. Hayes-Roth, F. and Lesser, V.R., Focus of attention in the Hearsay-II speech understanding system, in: *Proceedings Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA (1977) 27–35.
18. Hudlicka, E. and Lesser, V.R., Meta-level control through fault detection and diagnosis, Tech. Rept., University of Massachusetts, Amherst, MA, 1984.
19. Laird, J.E., Universal subgoaling, Tech. Rept., Carnegie-Mellon University, Pittsburgh, PA, 1984.
20. Lenat, D.B., Davis, R., Doyle, J., Genesereth, M., Goldstein, I. and Schrobe, H., Reasoning about reasoning, in: F. Hayes-Roth, D.A. Waterman and D.B. Lenat, (Eds.), *Building Expert Systems* (Addison-Wesley, Reading, MA, 1983).
21. Lesser, V.R. and Corkill, D., Functionally accurate cooperative distributed systems, *IEEE Trans. Systems, Man, Cybernetics* **1** (1981) 81–96.
22. McCarthy, J., Programs with common sense, in: *Proceedings Teddington Conference on the Mechanization of Thought Processes*, 1960.
23. Nagao, M., Matsuyama, T. and Mori, H., Structured analysis of complex photographs, in: *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan (1979) 610–616.
24. Newell, A., Shaw, J.C. and Simon, H.A., Report on a general problem-solving program, in: *Proceedings International Conference on Information Processing*, 1959.

25. Nii, H.P. and Aiello, N., AGE (attempt to generalize): A knowledge-based program for building knowledge-based programs, in: *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Toyko, Japan (1979) 645–655.
26. Nii, H.P., Feigenbaum, E.A., Anton, J.J. and Rockmore, A.J., Signal-to-symbol transformation: HASP/SIAP case study. *AI Magazine* **3** (1982) 23–35.
27. Smith, B., Reflection and semantics in a procedural language, Tech. Rept. MIT-TR-272, Cambridge, MA, 1982.
28. Smith, B. and Rivieres J.D., Interim 3-LISP Reference Manual. Tech. Rept., XEROX PARC, Palo Alto, CA, 1984.
29. Stefik, M., Planning and meta-planning (MOLGEN: Part 2), *Artificial Intelligence* **16** (1981) 141–169.
30. Terry, A., Hierarchical control of production systems, Ph.D. Thesis, University of California, Irvine, CA, 1983.
31. Van Melle, W., A domain-independent production-rule system for consultation programs, in: *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan (1979) 923–925.
32. Weyrauch, R.W., Prolegomena to a theory of mechanized formal reasoning, *Artificial Intelligence* **13** (1980) 133–170.
33. Wilensky, R., *Planning and Understanding* (Addison-Wesley, Reading, MA, 1983).

Appendix A. OPM: A Blackboard Control System for Multiple-Task Planning

OPM [16] is a blackboard control system that solves multiple-task planning problems like the one described in Section 2. This appendix presents excerpts from one of OPM's problem-solving traces to illustrate some of the possible behaviors of a blackboard control system and its achievement of the behavioral goals set forth in section 2.

OPM includes the following higher-level control knowledge sources:

- Accept-Problem is triggered by user input and creates a corresponding Problem decision.
- Modify-Criterion is triggered by a new Problem decision and, if necessary, refines the Problem Criterion. When there is insufficient time to perform all requested tasks, it changes the predicate 'all Requested-Tasks planned' to 'all important Requested-Tasks planned'.
- Planning-Policies is triggered by a new Problem decision. It recommends general scheduling Policies favoring recently triggered KSARs, KSARs with control knowledge sources, and KSARs with efficient, reliable, and important knowledge sources. It also recommends the scheduling rule: let the probability of scheduling a KSAR be proportional to the sum of its weighted ratings on all operative Foci and Policies.
- Refine-or-Chain? is triggered by a new Problem decision. It records either a successive-refinement Strategy decision or a temporal-chaining Strategy decision, as explained in Section 3.
- Implement-Strategy is triggered by either a new Strategy decision or satisfaction of a strategic Focus decision's Criterion. It implements an adopted Strategy as a series of Focus decisions, as explained in Section 3.

- Identify-Gaps-To-Fill is triggered by situations in which some blackboard regions contain fewer or less reliable solution elements than others. It creates Focus decisions favoring actions in those regions.
- Conserve-Resources is triggered when problem-solving time is 75% depleted. It recommends increasing the Weights of Policies favoring KSARs with efficient and reliable knowledge sources.
- Detect-Solution is triggered by the emergence of a plan that satisfies the Problem decision's Criterion. It changes the Problem's Status to 'solved'.

OPM uses these basic control knowledge sources:

- Update-To-Do-Set is invoked by execution of a Chosen-Action. It maintains a single To-Do-Set.
- Choose-KSAR is invoked by updating of the To-Do-Set. It chooses a KSAR from the To-Do-Set's Invocable-List, validates its Pre-Conditions, and records it as a Chosen-Action.
- Interpret-KSAR is invoked by creation of a Chosen-Action, which it executes.

These knowledge sources correspond to control heuristics derived from thinking-aloud protocols produced by people as they performed the multiple-task planning task. They control a version of OPM containing approximately fifty domain knowledge sources. They operate on the control blackboard defined in Fig. 8. In the trace below, they produce the control plan abstracted in the top panel of Fig. A.1.

A.1. Character of the problem-solving process

Fig. A.1 abstracts a protocol of OPM's problem-solving behavior for a single planning problem. Each number in Fig. A.1 represents a decision generated on a particular problem-solving cycle. Thus, the number 1 represents the decision made on cycle 1, the number 2 represents the decision made on cycle 2, and so forth. An ordered pair of numbers represents an ordered series of decisions. Thus, 140–143 represents the series 140, 141, 142, 143, while 88–84 represents the reverse series 88, 87, 86, 85, 84.

Placement of a number reflects three factors. First, placement on the upper or lower blackboard indicates whether the decision was a control or problem decision. Second, vertical placement within a blackboard indicates the decision's level of abstraction. For simplicity, control decisions at To-Do-Set and Chosen-Action levels do not appear in Fig. A.1. However, it should be understood that decisions at these two levels precede each decision that does appear. Third, horizontal span within a blackboard level designates the solution interval encompassed by a decision sequence. For the control blackboard, solution interval refers to the sub-sequence of problem-solving cycles encompassed by a decision. For the domain blackboard, solution interval refers to the sub-sequence of Operation-level plan elements encompassed by a decision.

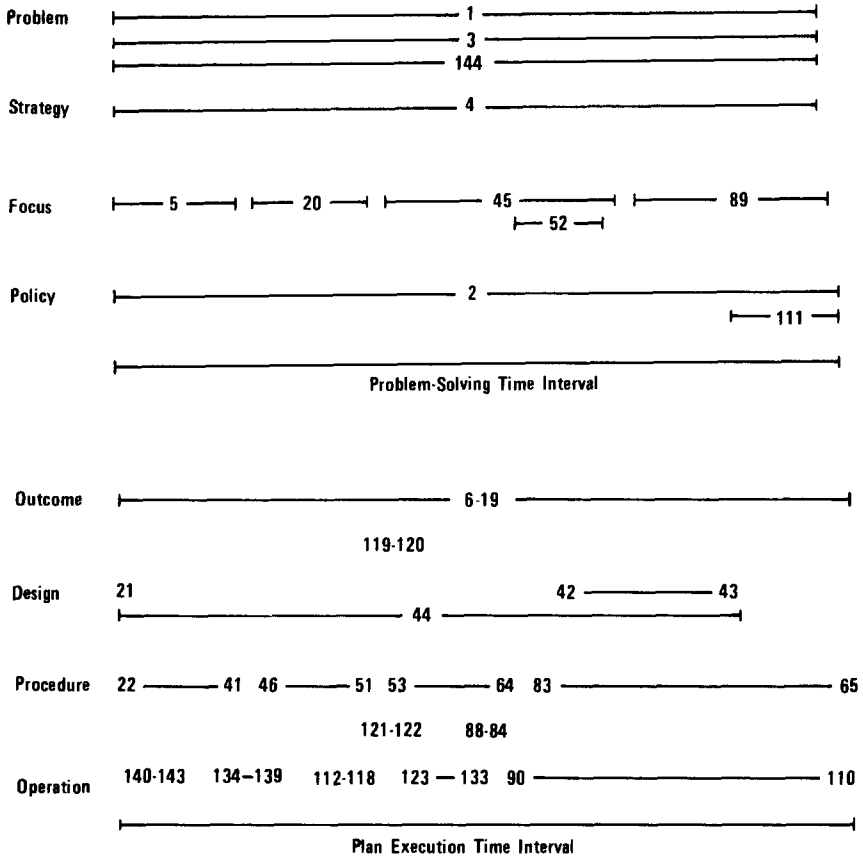

FIG. A.1. Abstracted OPM protocol.

Decisions on the control blackboard reflect OPM's orderly generation of a simple, well-defined control plan for this problem:

(1) Accept-Problem, triggered by input of the user's problem, creates decision 1, which describes the problem to be solved and remains operative for the entire problem-solving process.

(2) Planning-Policies, triggered by creation of the Problem, creates decision 2, which establishes general scheduling Policies that apply throughout the problem-solving process.

(3) Modify-Criterion, also triggered by creation of the Problem, creates decision 3, which changes the original Criterion predicate, 'all Requested-Tasks planned' to 'all important Requested-Tasks planned'.

(4) Refine-or-Chain?, also triggered by creation of the Problem, creates decision 4, the successive-refinement Strategy that remains opeative for the duration of the problem-solving process.

(5) Implement-Strategy, triggered by creation of the Strategy, creates decision 5, an Outcome level Focus.

(6) Domain KSARs execute and develop a complete Outcome level plan.

(7) Implement-Strategy, triggered by completeion of the Outcome level plan, creates decision 20, the Design level Focus.

(8) Domain KSARs execute, some of which develop a complete Design level plan.

(9) Implement-Strategy, triggered by completion of the Design level plan, creates decision 45, the Procedure level Focus.

(10) Domain KSARs execute.

(11) Identify-Gaps-To-Fill, triggered by a solution gap in the interval 2:00–4:00, creates decision 52, a Focus on that interval.

(12) Domain KSARs execute, some of which develop a complete Procedure level plan.

(13) Implement-Strategy, triggered by completion of the Procedure level plan, creates decision 89, the Operation level Focus.

(14) Domain KSARs execute.

(15) Conserve-Resources, triggered by depletion of 75% of the allowable problem-solving time, creates decision 111, which increases the Weights on Policies favoring KSARs with efficient and reliable knowledge sources to 0.9.

(16) Domain KSARs execute, some of which complete the Operation level plan.

(17) Detect-Solution, triggered by its determination that the current plan on the domain blackboard meets the Problem's Criterion, creates decision 144, changing the Problem's Status to 'solved'.

The simplicity of the control plan and the orderliness with which it is generated reflect the small number of control knowledge sources in the system and their excellent fit to this particular problem.

By contrast, decisions on the domain blackboard show the highly opportunistic generation of solution elements that is characteristic of blackboard systems. Decisions generated on successive problem-solving cycles may be at higher or lower levels of abstraction than their predecessors. They may encompass larger or smaller, overlapping or non-overlapping, earlier or later plan intervals. However, let us consider the generation of plan elements separately for the two blackboard dimensions.

A.2. Planning at different levels of abstraction

Fig. A.2 ignores solution interval and plots the level of abstraction for domain decisions as a function of cycle in the problem-solving process. Again, an ordered pair of numbers represents an ordered sequence of decisions. Each 'F' represents a control decision to focus on KSARs at the designated level of abstraction.

As Fig. A.2 shows, domain decisions conform roughly to the series of Focus

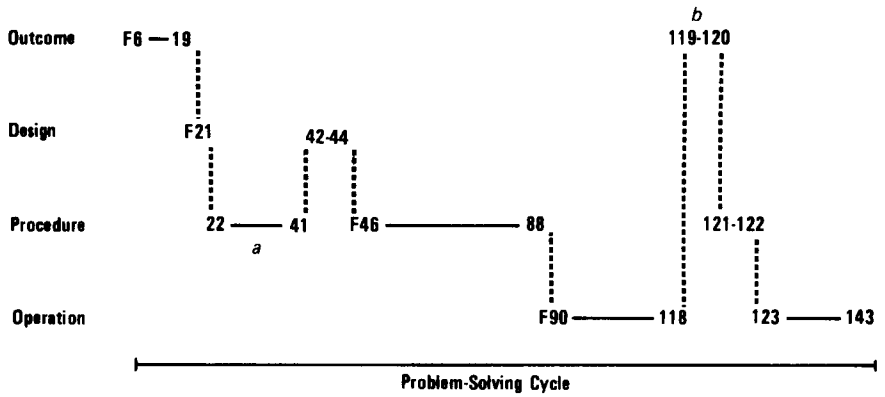

FIG. A.2. Strategic opportunism over levels of abstraction.

decisions prescribed by the successive-refinement Strategy decision. An initial series of Outcome decisions leads to series of Design decisions, Procedure decisions, and Operation decisions. This progression reflects Choose-KSAR's adaptation to heavily weighted strategic Focus decisions. It uses less heavily weighted, non-strategic Policy and Focus decisions mainly to discriminate among alternative KSARs in the current strategic Focus. However, there are two deviations from the top-down pattern, marked as '*a*' and '*b*' in Fig. A.2. First, a preliminary series of Procedure decisions intervenes early in the Design series. Second, pairs of Outcome and Procedure decisions intervene in the Operation series. These deviations reflect Choose-KSAR's potential to adapt to non-strategic Policies and Foci in favor of even a heavily weighted strategic Focus under the probabilistic scheduling rule. The following discussion illustrates the control decisions underlying both the strategic top-down progression and the opportunistic deviations illustrated in Fig. A.2.

A.2.1. *Strategic top-down domain planning*

First, let us examine the control decisions underlying the top-down progression from Design to Procedure levels.

In the discussion below, To-Do-Sets are condensed for brevity and stylized for comprehensibility. To illustrate, the first KSAR in the To-Do-Set for Cycle 43, below, is 'Try-Cluster E-042 Domain Design'. It represents the triggering of knowledge source 'Try-Cluster' by event 42. The next two words indicate that the KSAR's Action-Blackboard is 'Domain' and its Action-Level is 'Design'. Some KSARs in subsequent To-Do-Sets also indicate that their KS-Importance is 'Important'. The To-Do-Set lists KSARs in reverse chronological order, with the most recently triggered KSARs at the top. Other KSAR attributes (e.g., Trigger-Weight, KS-Efficiency, KS-Credibility) are omitted because they do not influence the behavior illustrated in these examples. Each KSAR is referred to

by combining its name and triggering event number. Thus, the first KSAR on this To-Do-Set is Try-Cluster-042.

The example begins on problem-solving cycle 43. On previous cycles, OPM adopted a highly Weighted (0.9) successive-refinement Strategy and the following Policies and Weights: Trigger-Recency = high, 0.9; Trigger-Weight = high, 0.5; KS-Efficiency = high, 0.3; KS-Credibility = high, 0.5; Action-Blackboard = 'Control', 0.9; and KS-Importance = high, 0.9. OPM has completed a satisfactory Outcome level plan. Its current Strategic Focus (Weight = 0.9) is on KSARs whose actions occur at the Design level. It has begun making decisions at that level.

```
Cycle 43:
Focus = Design

To-Do-Set =
  Try-Cluster          E-042  Domain  Design
  Notice-Proximity     E-041  Domain  Procedure
  Try-Beeline          E-034  Domain  Operation
  Try-Artery           E-034  Domain  Operation
  Set-Start-Or-End     E-033  Domain  Procedure
  Try-Cluster          E-011  Domain  Design
  Try-Loop             E-011  Domain  Design
  Try-Sweep            E-011  Domain  Design
  Try-Forward          E-011  Domain  Procedure

Chosen-Action = Try-Cluster-042

Event 43—Design Decision:
Organize the plan around the cluster of
  errands in the southeast part of town.
```

On cycle 43, the To-Do-Set contains nine domain KSARs. Try-Cluster-042 is the most recent and its Action is in the Design Focus. None of the other KSARs is both recent and in Focus. None has any other highly weighted scheduling attributes. Therefore, under OPM's probabilistic scheduling rule, Choose-KSAR is most likely to schedule Try-Cluster-042 and, as indicated below the To-Do-Set, it does. Choose-KSAR creates a new Design decision to organize the plan around a previously identified cluster of errands.

```
Cycle 44:
Focus = Design

To-Do-Set =
  Use-Cluster-C        E-043  Domain  Design
  Notice-Proximity     E-041  Domain  Procedure
  Try-Beeline          E-034  Domain  Procedure
  Try-Artery           E-034  Domain  Procedure
  Set-Start-Or-End     E-033  Domain  Procedure
  Try-Cluster          E-011  Domain  Design
  Try-Loop             E-011  Domain  Design
  Try-Sweep            E-011  Domain  Design
  Try-Forward          E-011  Domain  Procedure
```

Chosen-Action = Use-Cluster-C-043

Event 44—Design Decision:
Travel from the florist to the southeast
cluster and then to the final destination,
doing errands on the way.

The new To-Do-Set on cycle 44 contains a new KSAR, Use-Cluster-C-043. It also contains the unexecuted KSARs remaining from the previous To-Do-Set. Again, because Use-Cluster-C-043 is the only KSAR that is both very recent and in Focus, Choose-KSAR schedules it. Choose-KSAR creates a new Design decision encompassing the entire plan.

Cycle 45:
Focus = Design

To-Do-Set =

Implement-Strategy	E-044	Control	Focus
Try-Forward	E-044	Domain	Procedure
Notice-Proximity	E-041	Domain	Procedure
Try-Beeline	E-034	Domain	Operation
Try-Artery	E-034	Domain	Operation
Set-Start-Or-End	E-033	Domain	Procedure
Try-Cluster	E-011	Domain	Design
Try-Loop	E-011	Domain	Design
Try-Sweep	E-011	Domain	Design
Try-Forward	E-011	Domain	Procedure

Chosen-Action = Implement-Strategy-044

Event 45—Focus Decision:
Prefer KSARs that operate at the Procedure level.

On cycle 45, the To-Do-Set contains two new KSARs, Implement-Strategy-044 and Try-Forward-044, along with all of the previously unexecuted KSARs. Implement-Strategy is a control knowledge source, triggered by completion of the Design level plan, whose action is at the Focus level. It is also one of the two most recently triggered knowledge sources. Choose-KSAR schedules it. Implement-Strategy changes the Status of the Design level Focus to 'in-operative' and creates a new Procedure level Focus.

Cycle 46:
Focus = Procedure

To-Do-Set =

Try-Forward	E-044	Domain	Procedure
Notice-Proximity	E-041	Domain	Procedure
Try-Beeline	E-034	Domain	Operation
Try-Artery	E-034	Domain	Operation
Set-Start-Or-End	E-033	Domain	Procedure
Try-Cluster	E-011	Domain	Design
Try-Loop	E-011	Domain	Design
Try-Sweep	E-011	Domain	Design
Try-Forward	E-011	Domain	Procedure

```
Chosen-Action = Notice-Proximity-041

Event 46—Procedure Decision:
The movie is on Cedar Street, which is too far
  from the florist and in the wrong direction.
```

On cycle 46, there are no new KSARs. However, notice the changes in which KSARs are in Focus. Where Try-Cluster-011, Try-Loop-011, and Try-Sweep-011 were in the Design Focus on cycle 45, they are not in the new Procedure Focus on cycle 46. Conversely, the previously unfocused KSARs Try-Forward-044, Notice-Proximity-041, Set-Start-Or-End-033, and Try-Forward-011 are in the new Procedure Focus.

This cycle also illustrates the impact of Choose-KSAR's probabilistic scheduling rule. Because KSAR Try-Forward-044 is the most recent KSAR and in Focus, Choose-KSAR is most likely to schedule it, but in this case it does not. Instead, it schedules Notice-Proximity-041, which is also in Focus, but slightly less recent and, therefore, slightly less likely to be scheduled. Notice-Proximity-041 produces the first decision in the newly Focused Procedure level series.

In summary, this excerpt of OPM's behavior shows how it incrementally and dynamically implements the planned sequence of problem-solving activities prescribed by the successive-refinement Strategy: (a) Choose-KSAR adapts to an operative Strategic Focus, scheduling domain actions at the Design level; (b) the scheduled domain actions produce a Design level solution, which triggers Implement-Strategy; (c) Choose-KSAR, adapting to an operative Policy favoring control knowledge sources, schedules Implement-Strategy, which changes the Strategic Focus to the Procedure level; and (d) Choose-KSAR adapts to the new Strategic Focus, scheduling actions at the Procedure level. The result is top-down development of the domain plan.

A.2.2. *Opportunistic bottom-up domain planning*

Now let us examine the control decisions underlying the opportunistic deviation marked '*b*' in Fig. A.2. The example begins on cycle 118. OPM has developed a complete plan at Outcome, Design, and Procedure levels. Its current Strategic Focus is on KSARs whose Action-Level is 'Operation'. OPM has begun making decisions at that level and is in the process of plotting a route from the vet to the watch repair store.

```
Cycle 118:
Focus = Operation

To-Do-Set =
  Try-Artery       E-117  Domain  Operation
  Extend-Route     E-117  Domain  Operation
[Rest of To-Do-Set]

Scheduled-Action = Extend-Route-117

Event 118—Operation Decision:
Continue east on Oak Street toward the
  watch repair store
```

On cycle 118, the To-Do-Set contains two new KSARs, Try-Artery-117 and Extend-Route-117, both in the Operation Focus. Choose-KSAR is equally likely to schedule either of the two KSARs. (Other KSARs on the To-Do-Set have lower scheduling probabilities and are omitted for brevity.) It schedules Extend-Route-117, which creates the decision to continue east on Oak Street toward the watch repair store.

```
Cycle 119:
Focus = Operation

To-Do-Set =
  Notice-Other-Tasks  E-118  Domain  Outcome    Important
  Extend-Route        E-118  Domain  Operation
  Try-Artery          E-117  Domain  Operation
[Rest of To-Do-Set]

Scheduled-Action = Notice-Other-Tasks-118

Event 119—Outcome Decision:
The newsstand is very convenient to the
  route from the vet to the watch repair
```

On cycle 119, there are two new KSARs. Extend-Rule-118 is in the Operation Focus, while Notice-Other-Tasks-118 is not. However, Notice-Other-Tasks is a very important knowledge source. It notices when tasks previously excluded from the plan turn out to be very convenient, given the detailed specification of the plan. Thus, Notice-Other-Tasks is a bottom-up knowledge source that generates Outcome decisions on the basis of prior Operation decisions. Because the Operation level Focus and the KS-Importance Policy have equal Weights, Choose-KSAR is equally likely to schedule either KSAR. It schedules Notice-Other-Tasks-118, producing the observation that the newsstand, a previously excluded task, is convenient to the planned route from the vet to the watch repair.

```
Cycle 120:
Focus = Operation

To-Do-Set =
  Insert-New-Outcome  E-119  Domain  Outcome    Important
  Extend-Route        E-118  Domain  Operation
  Try-Artery          E-117  Domain  Operation
[Rest of To-Do-Set]

Scheduled-Action = Insert-New-Outcome-119

Event 120—Outcome Decision:
Include the newsstand in the plan
```

On cycle 120, the To-Do-Set contains a new KSAR, Insert-New-Outcome-119. Insert-New-Outcome is also an important knowledge source that incorporates a convenient task in the Outcome level plan. Because it is also the

most recent KSAR in the To-Do-Set, Choose-KSAR schedules it, adding the newsstand to the Outcome level plan.

```
Cycle 121:
Focus = Operation

To-Do-Set =
   Patch-Procedure   E-120  Domain  Procedure  Important
   Refine-Design     E-120  Domain  Procedure  Important
   Extend-Route      E-118  Domain  Operation
   Try-Artery        E-117  Domain  Operation
[Rest of To-Do-Set]

Scheduled-Action = Patch-Procedure-120

Event 121—Procedure Decision:
Go from the vet to the newsstand
```

On cycle 121, there are two new KSARs, Patch-Procedure-120 and Refine-Design-120. Neither is in the Operation level Focus, but because they concern insertion of the new Outcome level task in a previously specified Procedure, they are both important. Choose-KSAR schedules Patch-Procedure-120, producing the Procedure level decision to do the newly planned task, the newsstand, after the vet.

```
Cycle 122:
Focus = Outcome

To-Do-Set =
   Terminate-Route   E-121  Domain  Operation
   Refine-Design     E-120  Domain  Procedure  Important
   Extend-Route      E-118  Domain  Operation
   Try-Artery        E-117  Domain  Operation
[Rest of To-Do-Set]

Scheduled-Action = Refine-Design-120

Event 122—Procedure Decision:
Go from the newsstand to the watch repair
```

On cycle 122, there is one new KSAR, Terminate-Route-121, which is in the Operation Focus. Being most recent and in Focus, Terminate-Route-121 has the highest probability of being scheduled. However, because Refine-Design-120 is important and relatively recent, it also has a relatively high probability of being scheduled. Again demonstrating the effects of its probabilistic scheduling rule, Choose-KSAR schedules Refine-Design-120, even though it does not have the highest scheduling probability. It produces the Procedure level decision to go to the watch repair after the newly planned task, the vet.

```
Cycle 123:
Focus = Operation

To-Do-Set =
  Try-Beeline       E-122  Domain  Operation
  Try-Artery        E-122  Domain  Operation
  Terminate-Route   E-121  Domain  Operation
  Extend-Route      E-118  Domain  Operation
  Try-Artery        E-117  Domain  Operation
[Rest of To-Do-Set]

Scheduled-Action = Terminate-Route-121

Event 123—Operation Decision:
Go east on Oak to the newsstand
```

On cycle 123, there are two new KSARs, both of which are in the Operation level Focus and, therefore, most likely to be scheduled. In fact, all of the most recent five KSARs on the To-Do-Set are in Focus. Choose-KSAR schedules KSAR Terminate-Route-121. It resumes the Operation level decision series with a decision to go east on Oak Street to the newsstand.

In summary, this excerpt shows how OPM dynamically implements, interrupts, and resumes a planned series of problem-solving activities: (a) Choose-KSAR adapts to an operative Strategic Focus, scheduling actions at the Operation level; (b) Choose-KSAR adapts to an operative Policy favoring important KSARs, interrupting the planned series of Operation level actions by scheduling important Outcome and Procedure level actions; and (c) having scheduled all pending important actions, Choose-KSAR resumes scheduling the planned Operation level actions. The result is strategic opportunism.

A.3. Planning for different plan intervals

Fig. A.3 ignores level of abstraction and shows the progression of plan intervals encompassed by successive decision sequences. A decision sequence is a series of decisions referring to contiguous plan intervals. The set of plan intervals encompassed by a sequence is indicated by a horizontal line in Fig. A.3. For example, the first sequence, at the top of Fig. A.3, comprises decisions 6–19 and encompasses the entire plan. The last sequence, at the bottom of Fig. A.3, comprises decisions 140–143 and encompasses a short, early plan interval including the initial task.

If there were a strong relationship between decision sequence and temporal interval in the plan, Fig. A.3 would show a small number of long decision sequences, all of which proceeded either forward or backward over plan intervals. By contrast, Fig. A.3 displays numerous irregularities in the relationship between decision sequences and temporal interval in the plan. The ten decision sequences in Fig. A.3 include some that are unordered with respect to plan interval (e.g., —6–19—), some that move forward over plan intervals

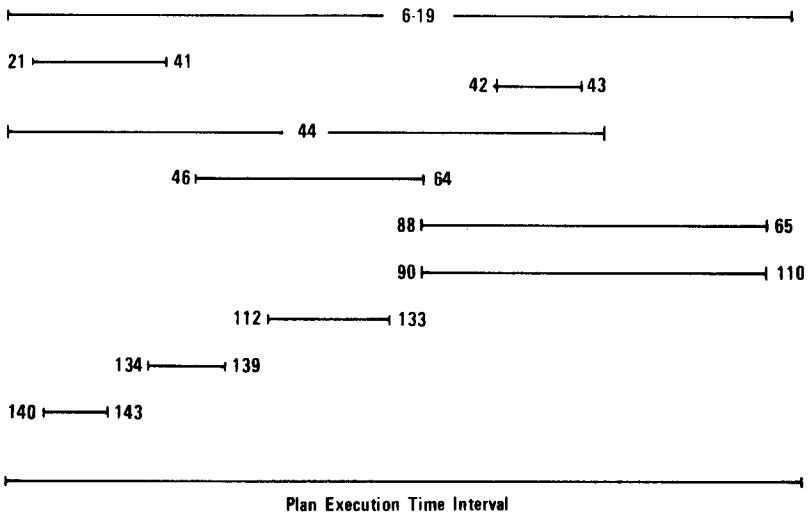

FIG. A.3. No pattern over plan execution intervals.

(e.g., 21—41), and some that move backward over plan intervals (e.g., 88—65). Successive decision sequences may reflect overlapping, but not identical plan intervals (e.g., —6–19— to 21—41), forward discontinuity (e.g., 21—41 to 42—43), backward discontinuity (e.g., 112—133 to 134—139), temporal reversal (e.g., 88—65 to 90—111), or some combination of these transitions (e.g., 46—64 to 88—65).

These irregularities reflect the absence of any Strategic Focus on plan interval or specific Policy that systematically discriminates among plan intervals. Any apparent local regularity in Choose-KSAR's scheduling of KSARs encompassing different plan intervals is an artifact of Focus and Policy decisions that really have nothing at all to do with plan interval. For example, a Focus decision that promotes decisions at a given level of abstraction may coincidentally lead to decisions at contiguous plan intervals. A Policy to favor KSARs with recent triggering events may do the same thing. However, because there is no Strategic Focus or Policy that explicitly discriminates among plan intervals, plan generation shows no global pattern with respect to plan interval.

A.4. General comments

The protocol abstracted in Fig. A.1 and analyzed in Figs. A.2 and A.3 represents OPM's performance under the successive-refinement strategy. This strategy was adopted and implemented dynamically during the effort to solve a particular planning problem. Other problems benefit from Refine-or-Chain?'s alternative temporal-chaining strategy: incorporate plan elements in the order in which they would execute in the world. For such problems, OPM adopts the

temporal-chaining strategy and produces the opposite results: little or no order in the scheduling of KSARs with respect to levels of abstraction, but roughly sequential scheduling of KSARs for successive plan intervals. These patterns of problem-solving behavior are quite similar to those produced by people solving the same problems with the same strategies.

The analysis in this section shows how the blackboard control architecture combines strategy and opportunism in problem-solving behavior. First, its control knowledge sources generate OPM's several kinds of control heuristics: Strategy decisions to define sequential plans for problem-solving actions; solution-based Focus decisions to emphasize actions appropriate for dynamic problem-solving situations; Policy decisions to impose general scheduling criteria throughout the problem-solving process. Its control blackboard preserves the distinctions among these heuristics in its different levels of abstraction. The blackboard control architecture's three basic knowledge sources (Update-To-Do-Set, Choose-KSAR, and Interpret-KSAR) provide a uniform scheduling mechanism to control OPM's generation of domain and control decisions and to integrate and adapt to the prescriptions of all operative control decisions. The result is the apparently complex problem-solving behavior illustrated in Fig. A.1. However, the decomposition of potential control factors in Figs. A.2 and A.3 reveals the orderly flow of Strategic behavior, overlaid with opportunistic interruptions and resumptions induced by non-strategic Focus and Policy decisions.

Appendix B. Replication of HEARSAY-II and HASP Control Behavior

This section shows how the blackboard control architecture would replicate the idiosyncratic, knowledge-intensive control behaviors of HEARSAY-II and HASP.

B.1. Replication HEARSAY-II's control behavior

HEARSAY-II interprets a subset of spoken English approximately in real time. Beginning with a parameterized representation of the speech signal, HEARSAY-II attempts to identify the data base query it represents. In so doing, it generates intermediate hypotheses regarding the Segments, Syllables, Words, Word Sequences, and Phrases the signal represents. These seven kinds of hypotheses appear at different levels of abstraction on the HEARSAY-II blackboard. The blackboard's solution intervals represent different temporal loci in the speech signal.

HEARSAY-II uses an implicit two-phase control strategy. During phase 1, it operates bottom-up, exhaustively scheduling all KSARs that produce hypotheses at the Parameter level, then the Segment level, then the Syllable level, and finally the Word level. During phase 2, it operates opportunistically, scheduling KSARs that have the highest ratings on several criteria: knowledge

source efficiency, knowledge source reliability, credibility of the triggering hypothesis, credibility of related hypotheses on the blackboard, need for hypotheses in targetted areas of the blackboard. The two-phase strategy was developed through extensive experimentation with a variety of methods and chosen because, of those tried, it worked the best. In particular, it compensates for the uncertainty of HEARSAY-II's low-level signal-processing knowledge sources. Opportunistic scheduling at higher levels of abstraction is effective only after establishing lower-level hypotheses up to the word level with relatively high confidence.

HEARSAY-II's strategy is implicit because the scheduler, which is the only 'official' repository of control knowledge, has no knowledge of it and does not explicitly implement it. In fact, the scheduler has knowledge only of the opportunistic scheduling criteria defined for its phase 2 processing. On each problem-solving cycle, regardless of strategic phase, the scheduler evaluates pending KSARs against its criteria and chooses the one with the highest weighted sum of ratings.

HEARSAY-II achieves bottom-up processing in phase 1 through careful engineering of four knowledge sources (one each for Parameter, Segment, Syllable, and Word levels) to insure: (a) that the appropriate one is triggered on each of the first four problem-solving cycles; and (b) that their actions generate all possible hypotheses at the associated levels in a single execution. In effect, while the scheduler has no knowledge of the bottom-up phase, it has no alternative but to select these knowledge sources in the desired bottom-up sequence. It is only after all four initial KSARs have executed that multiple KSARs are triggered simultaneously and the scheduler exerts any real influence over problem-solving activities. Thus, although HEARSAY-II's speech-understanding behavior follows an implicit control plan, its scheduler does not explicitly execute that plan.

HEARSAY-II implements another scheduling factor independent of the scheduler. A few special knowledge sources monitor the developing solution and create 'shadow hypotheses' to identify blackboard regions where little problem-solving progress has been made. These shadow hypotheses directly trigger relevant knowledge sources (which otherwise would not be triggered), combining triggering and attentional focusing functions in a mechanism entirely outside of the scheduler.

Although this account of HEARSAY-II's scheduler and other control mechanisms omits a few details, it is substantially correct (Lee Erman, personal communication).

Under the blackboard control architecture, eight control knowledge sources, in addition to Update-To-Do-Set, Choose-KSAR, and Interpret-KSAR, would operate as follows to produce the prototypical HEARSAY-II control plan illustrated in Fig. B.1.

(1) Accept-Problem accepts a user-specified problem description and

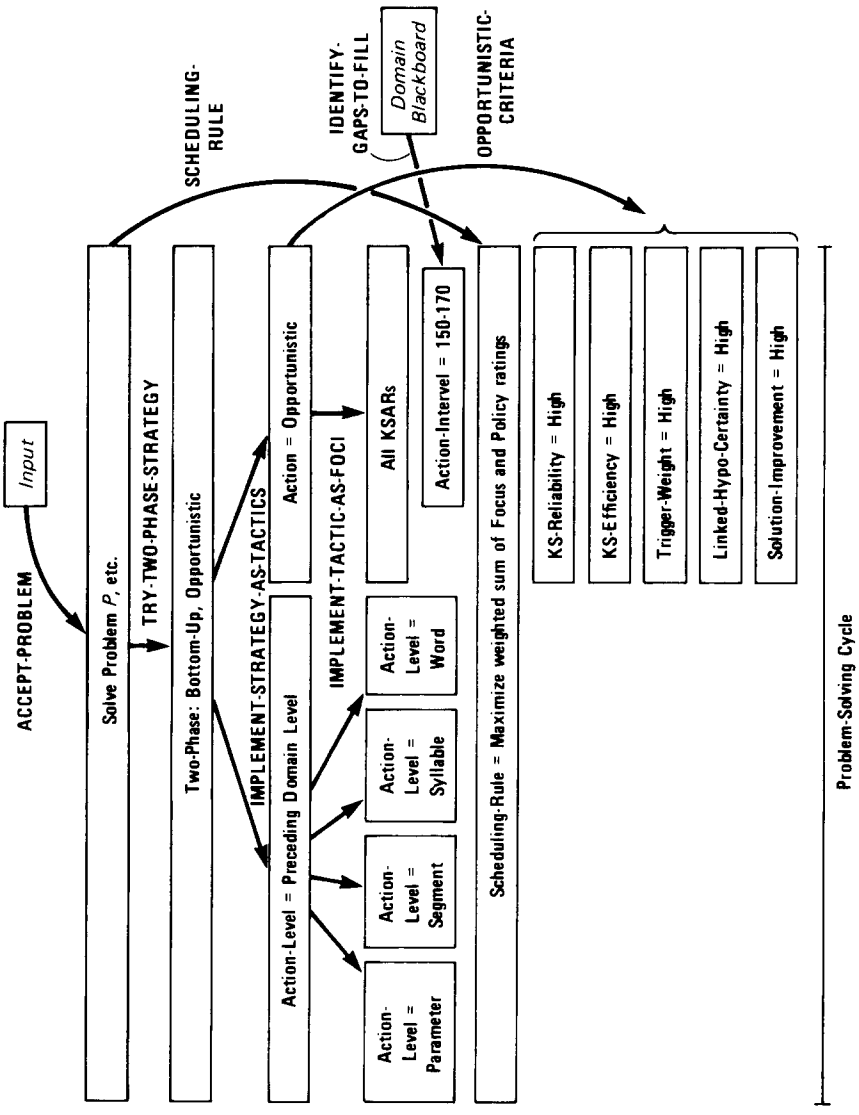

FIG. B.1. Prototypical control plan for HEARSAY-II.

creates a corresponding Problem decision, thereby initiating problem-solving activity.

(2) Scheduling-Rule, triggered by the Problem, creates Policy1, whose Goal specifies 'maximize the sum of ratings on all operative Foci and Policies'.

(3) Try-Two-Phase-Strategy, also triggered by the Problem, creates a Strategy decision representing the two-phase strategy described above.

(4) Implement-Strategy-As-Tactics, triggered by the appearance of the Strategy, creates a Tactic decision representing the bottom-up phase of the strategy described above.[8]

(5) Implement-Tactic-As-Foci, triggered initially by the appearance of the bottom-up Tactic, creates a Focus whose Goal is 'Action-Level = "Parameter"' and Criterion specifies that no Parameter-level KSARs remain in the To-Do-Set.

(6) Domain KSARs whose Action-Level = 'Parameter' execute in an arbitrary order.

(7) Implement-Tactic-As-Foci, triggered by the absence of Parameter-level KSARs in the To-Do-List, creates a Focus whose Goal is 'Action-Level = "Segment"' and Criterion specifies that no Segment-level KSARs remain in the To-Do-Set.

(8) Domain KSARs whose Action-Level = 'Segment' execute in an arbitrary order.

(9) Implement-Tactic-As-Foci, triggered by the absence of Segment-level KSARs in the To-Do-List, creates a Focus whose Goal is 'Action-Level = "Syllable"' and Criterion specifies that no Syllable-level KSARs remain in the To-Do-Set.

(10) Domain KSARs whose Action-Level = 'Syllable' execute in an arbitrary order.

(11) Implement-Tactic-As-Foci, triggered by the absence of Syllable-level KSARs in the To-Do-List, creates a Focus whose Goal is 'Action-Level = "Word"' and Criterion specifies that no Word-level KSARs remain in the To-Do-Set.

(12) Domain KSARs whose Action-Level is 'Word' execute in an arbitrary order.

(13) Implement-Strategy-As-Tactics, triggered by the absence of Word-level KSARs in the To-Do-List, creates the opportunistic Tactic, 'Action = "Opportunistic"'.

(14) Implement-Tactic-As-Foci, triggered by creation of the opportunistic Tactic, creates a Focus whose Goal is 'All KSARs'.

(15) Opportunistic-Criteria, also triggered by creation of the opportunistic Tactic, creates Policy decisions whose Goals favor KSARs with efficient and

[8] Because HEARSAY-II's two-phase strategy is more complex than either of OPM's strategies, the control blackboard has a Tactic level to permit an intermediate level of Strategy refinement before the Focus level refinement.

reliable knowledge sources, whose triggering information is credible, that create credible new hypotheses, that link new hypotheses to credible existing hypotheses, and that improve the solution.

(16) Domain KSARs execute in an order determined by the sum of their ratings against these Policies.

(17) Identify-Gaps-To-Fill, triggered by the comparative emptiness of solution interval '150–170', creats a Focus whose Goal is 'Action-Interval = 150–170'.

(18) Domain KSARs operate in this blackboard region and then in other regions.

(19) Detect-Solution monitors the developing solution and, when it detects a solution that meets the Problem decision's Criterion, changes the Problem's status to 'solved' and terminates problem-solving activities.

As the example illustrates, the blackboard control architecture can replicate HEARSAY-II's control behavior while providing several additional advantages. First, as discussed in Section 4, it achieves the behavioral goals set forth in Section 2. Second, it explicates important control reasoning done by HEARSAY-II's system builders, including some that is obscured in the scheduling program and some that is achieved implicitly with other mechanisms. Third, it illuminates control heuristics applicable to different problem domains. For example both HEARSAY-II and OPM use the control knowledge sources Accept-Problem and Identify-Gaps-To-Fill. HEARSAY-II's Implement-Strategy-As-Tactics and Implement-Tactic-As-Foci are computationally equivalent to OPM's Implement-Strategy-As-Foci. Fourth, it produces a control plan that is easy to comprehend, analyze, and compare to other control plans.

B.2. Replication of HASP's control behavior

HASP interprets sonar signals in real time. Beginning with a coded 'Line' representation of the signal, it attempts to characterize a 'Situation Board' that identifies, locates, groups, and characterizes the movement of all ships and other platforms in a circumscribed area of the ocean. In so doing, HASP generates intermediate hypotheses about the Harmonics in the signal, Sources of harmonics such as propellers or engines, and Vessels such as submarines or aircraft carriers. These five kinds of hypotheses appear at different levels of abstraction on the HASP blackboard. Its solution intervals represents different intervals in the signal.

HASP uses an iterative control strategy based on three event categories: Clock events must be processed before specified target times. Expected events can be processed when specified other blackboard events occur. Simple events can be processed at any time. Within each category, particular *types* of events trigger pre-determined sequences of knowledge sources. HASP's control program operates on three lists of unprocessed clock, expected, and simple events and a table of event types and associated knowledge source sequences to implement

the following procedure:

1. For each currently due clock event, selected in last-in-first-out order, execute the prescribed sequence of triggered knowledge sources associated with the event's type.
2. For each expected event whose specified blackboard events have occurred, selected in last-in-first-out order, execute the prescribed sequence of triggered knowledge sources associated with the event's type.
3. For the most recent simple event, execute the prescribed sequence of triggered knowledge sources associated with the event's type.
4. Repeat 2 and 3 until all expected events and simple events are processed.
5. Repeat 1–4.

Although this account of HASP's control behavior and mechanism omits some details, it is substantially correct (H. Penny Nii, personal communication).

Under the blackboard control architecture, six control knowledge sources, in addition to Update-To-Do-Set, Choose-KSAR, and Interpret-KSAR, would operate as follows to produce the prototypical HASP control plan illustrated in Fig. B.2.

(1) Accept-Problem accepts a user-specified problem description and creates a corresponding Problem decision.

(2) Scheduling-Policies, triggered by creation of the Problem, creates Policy1, whose Goal specifies 'maximize the sum of ratings on all operative Foci and Policies'. It also creates a Policy decision whose Goal is 'Action-Blackboard = Control' and Weight is high and a Policy decision whose Goal is 'KS-Priority = High' and Weight is low. These Policies and their Weights insure that HASP's procedural strategy executes invariably and that, within a specific Focus (selected event), triggered knowledge sources execute in the prescribed order.

(3) Try-Iterative-Strategy, also triggered by the Problem, creates a Strategy decision representing the iterative strategy described above.

(4) Iterate-Strategy-As-Tactics, triggered initially by creation of the Strategy, creates a Tactic whose Goal is 'Triggering-Event-Category = "clock" '.

(5) Iterate-Tactic-As-Step, triggered by creation of the clock-event Tactic, creates a Step whose Goal is 'Triggering-Event-Category = "clock" '.[9]

(6) Iterate-Step-As-Foci, triggered by creation of the clock Step, creates a Focus whose Goal is 'Triggering-Event = E-005 (that is, the most recently generated clock event)'.

(7) All pending domain KSARs with that Triggering-Event execute in descending order of KS-Priority.

[9] HASP's Strategy requires an additional Step level intervening between Tactic and Focus levels.

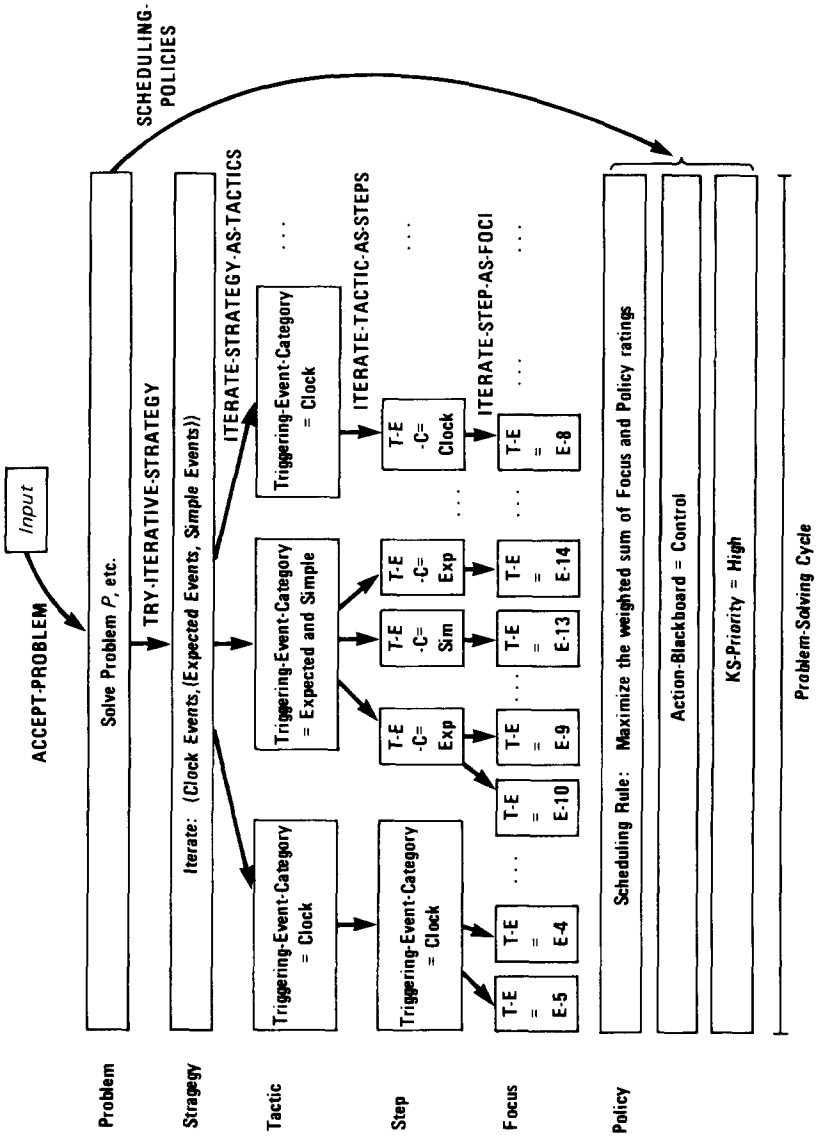

FIG. B.2. Prototypical control plan for HASP.

(8) Iterate-Step-As-Foci, triggered repeatedly by the absence of KSARs with the currently Focused Triggering-Event from the To-Do-Set, creates a series of Focus decisions whose Goals specify previously generated clock events.

(9) For each Focus decision, all pending domain KSARs with the Focused Triggering-Event execute in descending order of KS-Priority.

(10) Iterate-Strategy-As-Tactics, triggered by the absence of clock-event KSARs from the current To-Do-Set, creates a Tactic whose Goal is 'Event-Category = "expected and simple" '.

(11) Iterate-Tactic-As-Steps, triggered initially by creation of the new Tactic, creates a Step whose Goal is 'Triggering-Event-Category = "expected" '.

(12) Iterate-Step-As-Foci, triggered by creation of the expected-event Step, creates a Focus whose Goal is 'Triggering-Event = E-010 (that is, the most recently generated expected event)'.

(13) All pending domain KSARs with that Triggering-Event execute in descending order of KS-Priority.

(14) Iterate-Step-As-Foci, triggered repeatedly by the absence of KSARs with the currently Focused Triggering-Event from the To-Do-Set, creates a series of Focus decisions whose Goals specify previously generated expected events.

(15) For each Focus decision, all pending domain KSARs with the Focused Triggering-Event execute in descending order of KS-Priority.

(16) Iterate-Tactic-As-Steps, triggered by the absence of expected-event KSARs from the current To-Do-Set, creates a Step whose Goal is 'Triggering-Event-Type = "simple" '.

(17) Iterate-Step-As-Foci, triggered by creation of the simple-event Step, creates a Focus whose Goal is 'Triggering-Event = E-013 (the most recently generated simple event)'.

(18) All pending domain KSARs with that Triggering-Event execute in descending order of KS-Priority.

(19) If the executed domain KSARs generate new expected events, Iterate-Tactic-As-Steps is triggered again to create another expected-event Step. It is followed by multiple triggerings of Iterate-Step-As-Foci, creating a series of Foci on expected events, effectively reverting to item (12) above.

(20) For each Focus, all pending domain KSARs with that Triggering-Event execute in descending order of KS-Priority.

(21) If there are pending simple events, Iterate-Tactic-As-Steps is triggered again to create another simple-event Step. It is followed by a triggering of Iterate-Step-As-Foci, creating a Focus on the most recent simple event, effectively reverting to item (17) above.

(22) All pending domain KSARs with that Triggering-Event execute in descending order of KS-Priority.

(23) Iterate-Strategy-As-Tactics, triggered by the absence of expected-event

and simple-event KSARs in the To-Do-Set, updates the Clocktime and creates a new clock-event Tactic, effectively reverting to item (4) above. It continues creating new iterations of the Strategy in this fashion indefinitely.

As the example illustrates, the blackboard control architecture can replicate HASP's control behavior while providing several additional advantages. First, as discussed in Section 4, it achieves the behavioral Goals set forth in Section 2. Second, it explicates important control reasoning obscured in HASP's control program. Third, it illuminates control heuristics applicable to different problem domains. For example HASP, HEARSAY-II, and OPM all use the control knowledge source Accept-Problem. HASP's knowledge sources Implement-Strategy-As-Tactics, Implement-Tactic-As-Steps, Implement-Step-As-Foci are iterative variations on OPM's Implement-Strategy-As-Foci and HEARSAY-II's Implement-Strategy-As-Tactics and Implement-Tactic-As-Foci. Fourth, it produces a control plan that is easy to comprehend, analyze, and compare to other control plans.