A Planner for the Control of Problem-Solving Systems

Norman Carver and Victor Lesser

Abstract—**As part of research on sophisticated control for sensor interpretation, we have developed a planning-based control scheme for blackboard systems. A planner's goal/plan/subgoal structure provides explicit context information that can be used to index and apply large amounts of context-specific control knowledge. The key obstacle to using planning for the control of problem solvers is the need to deal with uncertain and dynamically changing situations without incurring unacceptable overhead. These problems have been addressed in several ways: our planner is script-based, planning and execution are interleaved, plans can invoke *information gathering actions*, plan refinement is controlled by plan-specific focusing heuristics, and the system's focus of attention can be dynamically shifted by the *refocusing mechanism*. Refocusing makes it possible to postpone focusing decisions and maintain the *opportunistic* control capabilities of conventional blackboard systems. Planning with refocusing results in a view of the control process as both a search for problem solutions and a search for the best methods to determine these solutions. The planner has been implemented in the RESUN (REsolving Sources of UNcertainty) interpretation system and has been used with a simulated aircraft monitoring application and a system for understanding household sounds. Our experience confirms that the combination of control plans with context-specific focusing heuristics provides a modular framework for developing and maintaining complex control strategies. In experiments, we have been able to achieve significant performance improvements as a result of the ability to encode sophisticated control strategies despite the overhead of the planning mechanism.**

I. Introduction

ONE of the major problems facing any artificial intelligence (AI) system is the *control problem*: what should the system do next? Control decisions in AI systems are uncertain because these systems do not have the knowledge that would allow them to choose the right action to take at each decision point. This is due to the lack of (tractable) optimal decision procedures for AI problems and/or uncertainty in the information that is necessary to make optimal decisions (e.g, models of the state of the world, applicability of operators). As a result, AI problem solving often involves *heuristic search* [33]. When we refer to "AI problem-solving systems" in this paper, we are specifically referring to systems that solve problems via heuristic search.

The performance of AI problem solvers that use heuristic search is dependent on the knowledge they can apply to control search. In some domains, "sophisticated control strategies" may be required to achieve acceptable levels of performance. By this we mean (in part) control that involves large amounts of highly context-specific heuristic search knowledge. In order to use sophisticated control strategies a problem-solving system must be able to encode and apply significant amounts of control knowledge without this knowledge overwhelming the problem solver and causing unacceptable overhead. Also of concern are issues of knowledge acquisition and engineering—i.e., does the framework make it easy to identify and maintain appropriate control knowledge? Thus, when we speak about whether or not a problem solver "supports" the use of sophisticated control strategies, we are talking about the notion of *heuristic adequacy* (being "efficient enough to be useful in practice" [36]), rather than any absolute ability/inability to implement particular search strategies.

In this paper, we will examine the use of planning-based[1] approaches for the control of AI problem-solving systems and we will present a new planning-based control framework that was developed for the RESUN interpretation system. The RESUN control planner grew out of efforts to extend the range of problem-solving strategies that are available to *sensor interpretation* systems. Sensor interpretation involves the determination of high-level *explanations* of sensor and other observational data. Interpretation can be difficult because there may be combinatorial numbers of alternative possible explanations of the data, the correctness of each interpretation hypothesis will be uncertain, creating each hypothesis may be computationally expensive, and the volume of data is often too great for complete examination [4].

Interpretation problems have typically been approached using *blackboard systems*. The blackboard model of problem solving is appropriate for sensor interpretation because blackboards support the incremental and opportunistic styles of problem solving that are required for such problems [3], [5], [8], [28]. However, despite the power of the blackboard model, most blackboard-based interpretation systems have used a very limited range of strategies

Manuscript received March 30, 1992; revised February 17, 1993. This work was supported by the Office of Naval Research under University Research Initiative Grant N00014-86-K-0764 and by DARPA under ONR Contract N00014-89-J-1877.

The authors are with the Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

IEEE Log Number 9212932.

[1]We use the term "planning-based" here to distinguish our sense of the term "planning" from the more restricted sense in which it is used in "classical planning" research—see [36]. By planning, we will mean "deciding on a course of action before acting" [11] (at least on a partial course of action) and developing an explicit goal/plan/subgoal structure as part of the control process.

for resolving hypothesis uncertainty. Specifically, they have used variations of *incremental hypothesize and test* that indirectly resolve uncertainty through the aggregation of evidence for the hypotheses. By contrast, strategies like *differential diagnosis*[2] can directly resolve the sources of uncertainty about interpretation hypotheses [4], [5].

In part, the strategy limitations of blackboard-based interpretation systems are a result of inadequate representations of evidence and uncertainty [4], [5], [28]. For instance, the ability to use more sophisticated strategies for resolving uncertainty requires that a system be able to identify when these strategies are applicable. This means that an interpretation system must understand the *reasons* why its hypotheses are uncertain (including the relationships between competing and cooperating hypotheses). The RESUN system provides this information by associating symbolic *source of uncertainty* (SOU) statements with its hypotheses. These SOU's are derived from RESUN's model of the uncertainty in (abductive) interpretation inferences [4], [5].

RESUN's SOU's make it possible to use a wide range of strategies for resolving uncertainty. However, we found that existing blackboard control mechanisms did not support the type of goal-directed (context-specific) reasoning that is necessary to deal effectively with such a range of interpretation strategies. For example, the process of making control decisions requires that the system consider why its overall goals remain unmet, use the SOU's to post subgoals representing what must be accomplished to meet its goals, select the best subgoals to devote its resources to, choose from among numerous alternative methods for satisfying those subgoals, determine how to specialize these methods to the particular situation, coordinate sequences of actions to implement the methods, and perhaps even reconsider its goal and method decisions as problem solving proceeds.

The RESUN control planner was developed to enable sophisticated interpretation strategies to be driven from RESUN's model of uncertainty. A planning mechanism was chosen because of its ability to support the use of context-specific control knowledge. The selection and refinement of goals and plans can be handled by plan-specific focusing knowledge, while the goal/plan/subgoal structure instantiated by a planner provides detailed and explicit information about the context of decisions. This results in a modular representation of control knowledge that makes it possible to efficiently index and apply large amounts of control knowledge. It also eases the task of encoding and updating this knowledge. The application of planning mechanisms to control problem-solving systems can be viewed as a logical step in the evolution of AI control from being implicit and reflexive toward being more explicit and deliberative. Experience with knowledge-based systems suggests that the use of explicit control mechanisms supports the development, understanding, and maintenance of systems with sophisticated control strategies [2], [13].

[2]For interpretation problems, incremental hypothesize and test means that hypothesis uncertainty is resolved by attempting to confirm the existence of all the data that would have resulted if the hypothesis were correct. Differential diagnosis means that hypothesis uncertainty is resolved by attempting to discount the possible *alternative explanations* for a hypothesis' supporting evidence. The possibility of alternative explanations for data is the key source of interpretation uncertainty since interpretation is based on abductive inference [5].

We believe that a planning-based control mechanism can support the use of sophisticated control strategies and this will be a major focus of this paper. However, relatively few AI problem solvers have used planning mechanisms. Why? Because planning is a highly goal-directed process, it can be difficult for planners to deal with uncertain and dynamically changing situations (without having unacceptable overhead from replanning). As we will see, the RESUN planner addresses these problems in several ways. The primary innovation of the framework is its *refocusing mechanism*. Refocusing makes it possible for the planner to maintain the *opportunistic* control capabilities of more conventional blackboard problem-solving systems.

In the next section, we will examine the issues involved in the control of AI problem-solving systems and the use of planning-based approaches to provide this kind of control. Section III provides a detailed look at the RESUN control planner. This is followed in Section IV by a summary of the results of our experiences and experiments with RESUN. In Section V we review a number of specific planning and control mechanisms to show how our work relates to other AI research. The paper concludes with a summary of the contributions of the framework and directions for future research.

II. Control of AI Problem Solvers

A. Basic Issues

AI systems have typically made control decisions in a two-stage process. Davis [13] terms these the *retrieval* and the *refinement* stages. The retrieval stage determines the actions that the system might take next, given the current state of problem solving. The refinement stage selects the action(s) that actually will be taken next (out of those identified by the retrieval stage). Retrieval is usually implemented with a fixed knowledge indexing scheme that is appropriate for the characteristics of the problem—e.g., data directed, goal directed, difference directed, etc. Refinement involves *heuristic* decisions about the "best" course of action in the situation (heuristic control knowledge may also be applied in the retrieval stage, but this makes it more difficult to understand and modify).

Traditionally, AI systems used monolithic evaluation functions to make refinement-stage decisions. A single ("global") evaluation function computed a numeric rating for each of the retrieval-stage alternatives, which reflected that alternative's "likelihood" to succeed. One of the key problems with this approach is what Doyle [14] termed the "inaccessibility of control information." All the reasoning being done by the evaluation function is *implicit* in the function's computations and the resulting numeric rating. This can make it difficult to encode (and maintain) sophisticated heuristic control knowledge because users have trouble understanding complex rating functions—especially when they incorporate ratings associated with hypotheses and data objects.

Implicit control reasoning also makes it difficult to reconsider control decisions. Because of the uncertainty inherent in their decisions, AI systems must be able to reconsider decisions and select alternative(s) to pursue—i.e., they must be able to review their decisions. This is usually done through backtracking or truth maintenance system (TMS) mechanisms. However, if the only thing that a revision process has to work with is the final numeric rating of each alternative, it cannot take failures into consideration when revising a decision. Because it cannot understand the factors that went into the ratings it can do little more than select the next most highly rated alternative. However, this alternative may also be likely to fail if its relatively high rating results from the same factors as an already failed choice. Note also that dependency-directed backtracking mechanisms do not solve this aspect of the decision/revision problem [4].

To deal with these issues, a number of researchers have advocated that the process of making control decisions be viewed as a problem-solving task in itself and that this process be made more explicit—e.g., [9], [12], [13], [21], [27]. This involves providing a body of explicit meta-level (heuristic) knowledge about the domain and actions that is used to guide the refinement process. With such knowledge, a system can "reason explicitly" about control because the meta-level knowledge explicitly identifies the factors that influence control decisions. For production (rule-based) systems, Davis [13] proposed that explicit meta-level knowledge might take the form of a set of *meta-rules*. Meta-rule conditions would refer to key features of the problem-solving situation. Meta-rule "actions" would then state preferences for (domain) rules whose characteristics are appropriate for the current situation. The meta-rules would be applied during the conflict-resolution (refinement) stage of a production system to select the object-level rule(s) to be fired.

The addition of meta-rules allows a system to reason explicitly in making refinement decisions, but problems remain. One problem is that there may be multiple meta-rules that are applicable in any given situation. While this could be handled through a meta-rule refinement process (e.g., using meta-meta-rules), the "advice" from multiple meta-rules is typically combined by resorting to numeric ratings—e.g., using meta-rule actions like "rules whose premise mention x are likely (.6) to be useful." Davis believes that this approach is "effective where decisions can be made by combining the contributions of a number of independent mechanisms." Still, it eliminates many of the advantages of the meta-rule format since the object-level factors explicitly considered by the meta-rules are lost in the conversion to numeric ratings. In addition, this approach can make it difficult to realize a particular strategy since the ratings from various meta-rules will have to be balanced. Other problems with the meta-rules format (and any other form of meta-level knowledge) relate to the potential for greatly increased control overhead due to the need to retrieve and apply what might be a very large number of meta-rules.

In order to take full advantage of meta-level knowledge such as meta-rules, there must be additional structure to the knowledge that identifies the context in which the knowledge applies in order to index (and partition) the knowledge [4], [13]. Davis recognizes this as a solution to the overhead problem and suggests that meta-rules be associated with the (domain action) retrieval properties—e.g., goals. However, we would like to go further and eliminate meta-rules conflicts as well when appropriate—i.e., when we do not need to numerically combine advice from independent factors. If sufficiently detailed contexts can be specified, then there only needs to be a single piece of meta-level knowledge for any context. We believe that this type of approach has many advantages (modularity and explicit resolution of strategy conflicts).

Another critical issue for control in many AI problem solvers is the need to integrate both data-directed (event-directed) and goal-directed control factors. Early blackboard systems relied heavily on data-directed control mechanisms in order to provide *opportunistic* control capabilities: The ability to rapidly shift the focus of attention of the system based on factors such as developing solution quality or the appearance of new data. Opportunistic control allows a system to deal with uncertain and dynamic situations because the system can redirect its efforts as necessary—instead of being limited to a predetermined (and possibly inappropriate) strategy. However, there has been a steady evolution of blackboard control toward more goal-directed mechanisms [7]. The reason for this is clear: without (explicit) goal-directed control it is difficult for a system to identify actions critical to meeting its overall goals (especially when a sequence of actions is necessary to accomplish some goal [12]). Thus, a key issue is how to make control more goal directed without sacrificing opportunism.

B. Planning for Control

In this section we will look at how planning-based approaches to control can address the issues raised in the previous section. First of all, it is important to understand that "classical planning" research [36] is not generally appropriate for the control of problem-solving systems because it involves *strategic planning*: determining a complete sequence of actions that solve a problem prior to taking any actions. Strategic planners are not useful for applications in which blackboard systems would typically be used because the situations are too uncertain and/or dynamic to permit problem-solving actions to be completely preplanned. In other words, the problem with classical planners is that they are neither *opportunistic* nor *reactive* [36]. Recently, there has been a significant amount of research on *reactive planning* [16], [18] and on the interleaving of planning and execution [26]. As we will see, these types of approaches can make planning appropriate for the control of problem-solving systems.

One of the key motivations for using a planning-based control framework is that the goal/plan/subgoal hierarchy that is instantiated by a planner provides detailed and explicit context information for control decisions. In other words, control decisions result from planner focusing decisions and when focusing decisions are made, it is clear from the hierarchy exactly what the context of the decision is: the purpose of the decision in terms of the goals and subgoals to which it pertains, the relationships among the various decision alternatives, etc. Having detailed and explicit context for each decision facilitates the implementation of sophisticated control strategies and explicit control reasoning. This is because the context information can be used to structure and index the control knowledge.

Another way in which planning facilitates explicit control reasoning is that it separates the process of determining how it is *possible* to accomplish a goal from the process of deciding how it is *best* to accomplish the goal (in the current situation). In other words, the plans represent the "methods" that the system can use to try to achieve its goals while the focusing knowledge that selects plans and goals represents the "strategies" the system should use in particular situations. This separation makes it easier to encode complex control strategies since the control reasoning does not have to simultaneously consider what actions might be used in satisfying a goal and what is the best way to proceed. In addition, the uncertain, heuristic portion of the control knowledge is to some extent now separated into the heuristics.

It should be noted that these two components of planning control decisions (e.g., identifying the plans that are applicable to a goal and choosing one) are not quite the same as the retrieval and refinement stages of conventional control methods described above—though they are analogous. A key difference between a planning-based control approach and that of most other control frameworks is in the way that *control decisions*—in the sense of decisions about the *domain action* to take next—are made. In conventional systems, decisions are made by directly identifying potential actions and selecting from among them—i.e., retrieval and refinement. In a planning framework, action decisions result from a sequence of planner focusing decisions that are made during plan expansion and refinement. This is because domain actions correspond to primitive plans—the leaves of the planning tree. As a result, there is a search process inherent in the selection of system actions whenever planning-based control is used. With complex decisions, searching for the best action may be more appropriate than trying to directly consider all the relevant factors.

Another important capability that a planning-based system has is the ability to coordinate sequences of actions to meet goals. In part, this ability is related to the separation of methods and strategies. Having explicit plans for accomplishing goals keeps the system's actions focused on the goals that it has decided to pursue. Decisions have a more pervasive effect on the actions of the system because they are represented explicitly and they affect all lower-level decisions. Revision of decisions is, likewise, more efficient since reconsideration of goal and plan decisions can affect an entire sequence of potential actions.

Because planning-based control is highly goal-directed and creates detailed system subgoals, it provides capabilities that most agenda-based blackboard frameworks lack. For example, a planning-based system can control the amount of data that undergoes (any) processing and can actively direct data gathering. These are important capabilities for interpretation problems, which may involve passive sensors that continuously generate large amounts of data and active sensors whose operation may be controlled by the interpretation system. Recent work on the BB1 framework [24] does provide a mechanism that blackboard systems might use to deal with large amounts of data being generated by sensors. Here, the control component provides filters to autonomous processors that limit the data that is passed on to the main reasoning component. However, with the kind of detailed planning we propose, the control component itself can provide much finer control by directly considering the data to be examined. We will provide examples of such strategies in the next section.

III. The RESUN Control Planning Mechanism

The planner that we developed for control in the RESUN interpretation system maintains the advantages of planning frameworks that were outlined in the previous section, while it addresses the problems that planners can have in dealing with uncertain and dynamically changing situations. Specifically, the RESUN planner is opportunistic and reactive.[3] In the first subsection of this section, we will discuss the need for explicit goals in planning-based systems and the way that RESUN identifies its goals. The next subsection will illustrate the basic planning mechanism. In Sections III-C and III-D, we will examine the focusing and refocusing mechanisms that are critical to the successful use of the basic planning mechanism. The final two subsections of this section look in more detail at the kinds of actions that are used in the system and the way control plan schemas are defined.

[3]Opportunistic control and reactive planning/control are related, but distinct concepts. In part, this is because reactive control is typically studied in domains with real-time constraints. Opportunistic control means being able to take advantage of new information or situations to change strategies and improve problem-solving performance. Reactive control means being able to "react in an acceptable amount of time to any changes that occur in the world" [36]. To be opportunistic in a domain with real-time constraints requires that a system be reactive. However, being reactive does not guarantee that a system will be opportunistic—except in a very weak sense. Note also that the concept of reactive planning is useful even in domains without real-time constraints. This is because reactive planning implicitly includes the notion of controlling the deliberation (control reasoning) of a planner. Controlling the overhead of a planner is a key concern when using planning-based control mechanisms and it is this sense in which we talk about the RESUN planner being reactive.

A. Goals and Interpretation Uncertainty

Since planning involves problem reduction [33], a planning-based approach to control requires that the subgoals the system must accomplish if it is to satisfy its overall goal be made explicit (and those subgoals must be decomposable). RESUN supports the creation of explicit subgoals through its model of the uncertainty in (abductive) interpretation inferences. This model allows the system to create a set of symbolic statements, *SOU*'s, that represent the *sources of uncertainty* in the evidence for a hypothesis. Interpretation is then implemented as an incremental process of resolving particular sources of uncertainty in the hypotheses. When the level of uncertainty in a hypothesis must be reduced, the SOU's associated with the hypothesis are used to post subgoals for resolving uncertainty. Through the planning process, these subgoals drive the system to pursue methods and take actions that are appropriate for the current situation. The overall problem-solving process is driven by a symbolic representation of the reasons why the system's termination goals remain unsatisfied.

The examples in this section will be from the simulated aircraft monitoring application that we have been using to experiment with RESUN. In aircraft monitoring, data from acoustic and radar sensors must be correlated and integrated in order to identify and track aircraft moving though a specified region. Interpretation of the data is complicated by the occurrence of nonaircraft signals due to noise or ghosting, by incomplete or imprecise sensor data, and by the need to deal with large amounts of data from passive sensors. Termination requires that the system not only sufficiently resolve its uncertainty about potential aircraft that have been identified, but that it also has examined enough data to be sufficiently certain that there are not additional, unidentified aircraft.

An interpretation system for aircraft monitoring creates *track* hypotheses that represent possible aircraft and their movements over time. Track hypotheses are supported by *vehicle* hypotheses representing possible aircraft "sightings," which in turn are supported by sensor data. The symbolic SOU's associated with hypotheses represent the reasons for uncertainty in the hypotheses due to factors such as incomplete support, possible alternative explanations for support, partial satisfaction of constraints, and competing interpretations of the data. At the problem-solving level, termination uncertainty involves factors such as potential answer (aircraft track) hypotheses that are too uncertain to be accepted, areas in the region to be monitored for which no data has been examined, and sensor data that has not been ruled out as being from an additional aircraft. The aircraft monitoring application and the model of uncertainty used in RESUN are described in more detail in [4], [5].

B. Plan Expansion and Refinement

The RESUN approach to planning can be described as script based, incremental planning augmented with context-specific focusing and refocusing mechanisms. *Script-based* planning means that the methods which the system can use to satisfy its goals (and subgoals) are defined as a set of control plan *schemas*. There are two classes of control plans: primitive and nonprimitive. *Primitive* control plans correspond to *actions* that the system can take to immediately satisfy some goal. *Nonprimitive* control plans specify a sequence of subgoals which, if they can be satisfied, accomplish the plan goal.

Incremental planning means that planning and execution are interleaved—i.e., plans are only expanded and refined to the point where the next action (primitive plan) that can be executed is identified. This action is then executed and its results used to update the planning structure prior to further planning. The basic RESUN control planning loop is detailed in Fig. 1. The planning process creates a hierarchical graph structure like that shown in Fig. 2. This plan refinement/expansion structure consists of instantiations of plans, plan versions, and subgoals (plan versions represent alternative versions of a single plan instantiation due to the selection of alternative bindings of the plan variables—for the sake of readability, they are not shown in Fig. 2).

The initial (top-level) subgoal in Fig. 2, Have-Reduced-Hyp-Uncertainty, has been posted by expansion of a higher level plan for meeting the termination goals (based on the model of overall problem-solving uncertainty). Here, the system has chosen to try to reduce the uncertainty in a particular possible answer hypothesis: track-hyp$_2$. When the subgoal is posted, two plans are identified that are capable of satisfying the subgoal: Eliminate-Hypothesis-SOUs and Reduce-Hypothesis-SOUs. These two plans differ in how much work they do to resolve uncertainty each time they are called—i.e., they differ in grain size (this will be discussed further in Section III-D). Because there are alternative methods for satisfying this subgoal, the focusing mechanism is invoked to decide which plan(s) to pursue. For this example, we have assumed that the focusing knowledge selects Eliminate-Hypothesis-SOU's as the "better" choice. This results in an instantiation of the Eliminate-Hypothesis-SOUs plan (whose definition is given later in Fig. 3) being created and placed onto the list of focus points to be pursued next (actually it is the initial *plan version* of this plan instantiation that is placed onto the list of focus points).

When the Eliminate-Hypothesis-SOUs focus point is pursued, initial expansion results in the posting of a single subgoal, Have-Hypo-SOU. The system finds a primitive plan, Identify-Hypothesis-SOU, that can accomplish the subgoal. An instantiation of this primitive is created (with its ?hyp variable bound to track-hyp$_2$) and added to the list of focus points. When this focus point is pursued, the function that implements the primitive is executed. This action returns a list of the SOU's currently associated with the hypothesis track-hyp$_2$ (actually it returns a multiple-valued value as explained in Section III-C). The list of SOU's is bound to the ?sou variable of the primitive and the primitive's status is set to "finished." This initiates

```
Initialize current-focus-points to the top-level plan-version
repeat: repeat: Pursue-Focus on each element of current-focus-points
                until null(current-focus-points)
        set current-focus-points to next-focus-points
        until null(next-focus-points)

Pursue-Focus(focus-point)
 case on type-of(focus-point):
  plan-version Focus on variables to be used in plan expansion which have multiple-valued values
                 and select a set(s) of bindings to pursue.
               Create a new plan-version for each selected alternative set of bindings.
               Expand each selected plan-version and post its next active subgoals.
               Focus on each plan-version's subgoals to select the subset of the subgoals to pursue next.
               Match each selected subgoal against the relevant defined control plans.
               For each subgoal with multiple matching plan types, focus on the matching plans to select
                 the plan(s) to use to pursue the subgoal.
               Create a new plan-version for each selected plan type and add it to next-focus-points.
  primitive    Execute the function associated with primitive to get status and results.
               Update plans to select new focus element for next-focus-points:
                 propagate status and results of primitive to matching subgoal
                 and then up the control plan hierarchy to identify in-progress plan-versions.
```

Fig. 1. The basic control planning loop.

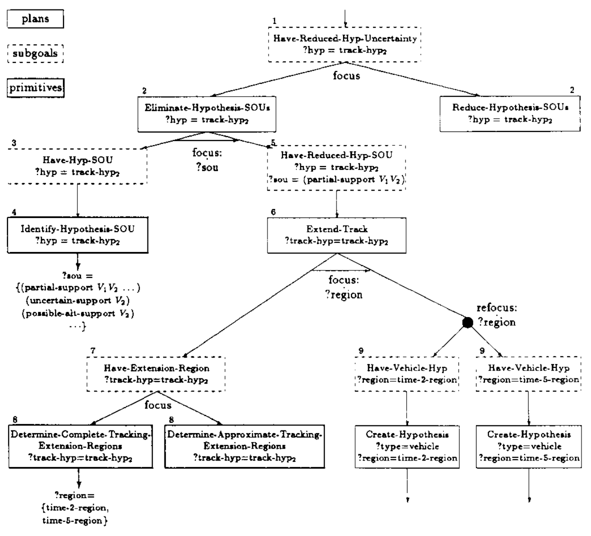


Fig. 2. An example of the structure created by planning (for simplicity, plan versions are not included). Numbers above the nodes indicate the order of creation.

Plan Definition:

Plan Name	Eliminate-Hypothesis-SOUs
Description	Iteratively eliminates sources of uncertainty from the hypothesis ?hyp while the degree-of-belief in ?hyp is less than ?min-belief.
Goal Form	(Have-Reduced-Hyp-Uncertainty ?hyp ?min-belief)
Input Variables	(?hyp ?min-belief)
Output Variables	()
Internal Variables	(?sou)
Grammar	(:ITERATION (< (BELIEF ?hyp) ?min-belief) (:SEQUENCE Have-Hyp-SOU Have-Reduced-Hyp-SOU))

Subgoal Definition:

Subgoal Name	Have-Hyp-SOU
Goal Form	(Have-Hyp-SOU ?hyp ?sou)
Input Variables	(hyp)
Output Variables	(sou)

Fig. 3. An example of a control plan definition.

the process of updating the status of higher level subgoals and plans. Here, the list of SOU's is bound to the ?sou variable of the Have-Hyp-SOU subgoal (and its containing plan) and the status of this subgoal becomes "finished."

With the change in status of its subgoal, the state of the Eliminate-Hypothesis-SOUs plan changes. However, it is not yet finished, so the plan is again added to the list of focus points. When this focus point is pursued, further expansion of the plan will result in the subgoal Have-Reduced-Hyp-SOU being posted. Since this subgoal has the variable ?sou as an input variable and ?sou is bound to a list of SOU's, the focusing mechanism must be invoked to select the value(s) of ?sou to be used in posting the new subgoal(s) [i.e., to select the "best" version(s) of this plan]. For this example, we have assumed that focusing selects the *partial-support* SOU as the single best SOU to be resolved next (this SOU represents uncertainty in the hypothesis track-hyp$_2$ due to an incomplete set of supporting vehicle hypotheses). Following the focusing decision, a subgoal whose ?sou variable is bound to the partial-support SOU is posted. There is a single plan, Extend-Track, that can satisfy this new subgoal, so an instantiation of the plan is created and added to the list of focus points.

Initial expansion of Extend-Track results in the subgoal Have-Extension-Region being posted. Two plans (primitives) match the newly posted subgoal (representing alternative methods for identifying the regions in which to look for data to extend Track-Hyp$_2$). We have assumed that focusing selects the "complete tracking" primitive plan. Execution of the primitive results in two possible bindings for ?region so focusing is applied prior to further expansion of Extend-Track. At this point, the example shows a situation in which both alternative bindings have been chosen to be pursued. In this case, the system is postponing its decision in order to gather more information about the available data before choosing how to extend the track. We will return to this later when we cover the refocusing mechanism.

This example illustrates the basic operations of the control planner, but it has barely touched on the focusing and refocusing mechanisms. Furthermore, because it shows planning as a depth-first search process, the notion of focus points within the planning structure may not appear to be very useful. In the remainder of this section, we will see how the RESUN planner allows multiple plans and subgoals to be pursued in parallel and how refocusing makes it possible to opportunistically change focus points.

This example does illustrates the advantages of planning-based control for coordinating sequences of actions to meet a goal. In order to resolve a track partial-support SOU, a sequence of actions is necessary (determine where to look for data to extend the track, perform the inferences necessary to create a vehicle hypothesis, etc.). In a conventional (nonplanning-based) blackboard control scheme each of these actions will have to be separately considered and scheduled. One problem with this is that it can be difficult to judge the importance of the actions without an understanding of the role they play in fulfilling some goal—e.g., because the "quality" of intermediate-level hypotheses may be poor from a purely data-directed perspective [12]. Another problem with this approach is that it can incur substantial overhead since the decision about whether to continue with the sequence is effectively being

being reconsidered on each control cycle. By contrast, in a planning-based approach like RESUN's, the decision to extend the track is made at one level in the structure and only decisions about how to actually implement the plan need be made during the planning cycles below the initial decision.

The basic design of the RESUN control planner resulted from the requirement of reactive planning for controlling a problem-solving system. One way that planning can be made more reactive is by making the planning process more tractable (than classical planners). Making a planner script-based makes the planning process more tractable because the use of plan schemas limits the "reasoning about actions" that is a major source of combinatorial complexity in classical planners [36]. However, it is important to note that planning is still a combinatorial problem due to uncertainty about the best plans to use to satisfy subgoals and the best plan versions (method instances) to be pursued. In other words, script-based planning still involves search.

Another way that planning is made more reactive is by enabling a planner to deal with uncertainty about the outcome of actions and uncertainty about the exact state of the world. The RESUN planner deals with this uncertainty through incremental planning and by maintaining only a partial model of the state of the world. Forming complete plans prior to executing them is not possible for control in applications like interpretation where the outcome of actions is uncertain and where external agents can affect the world [26]. Our control planner is a reactive planner, in part, because it interleaves planning with execution and bases further expansion of its plans on the outcome of earlier actions. Likewise, while classical planners maintain a complete model of the state of the world, this is inappropriate for most problem-solving domains. In the RESUN planner, only partial models of the world are maintained through the bindings of plan variables. The plans contain only the world state information that they require and this is updated as necessary through the use of explicit information gathering actions.

C. Heuristic Focusing

As our discussion of the example of Fig. 2 showed, the heuristic focusing mechanism controls the plan refinement process whenever there are alternative ways to refine the plans—i.e., it controls the planner's search process. Focusing decisions may have to be made at three different points in the planning process. To provide context and structure to the heuristic knowledge, we partition focusing knowledge into three corresponding classes of heuristics: subgoal, match, and variable.

Subgoal focusing heuristics select among the active subgoals for a plan instance when a control plan schema's grammar specifies that certain subgoals may be satisfied in parallel. Each subgoal heuristic is associated with a specific parallel subgoals construct (e.g., a :SHUFFLE) in the sequence grammar of a particular control plan schema. Subgoal heuristics are useful because even when it is possible for subgoals to be pursued in parallel, it may be preferable to sequence the subgoals—e.g., due to uncertainty whether certain subgoals can be satisfied given the current situation. Rather than encode this kind of heuristic knowledge in the plan grammar, it is encoded in subgoal focusing heuristics so that the system has more flexibility in applying the knowledge.

Match focusing heuristics select which plan(s) to pursue when there are multiple control plan schemas whose Goal Forms unify with a particular subgoal's Goal Form. In other words, they select among competing methods for satisfying a subgoal. Each match heuristic is associated with a specific subgoal of a particular control plan schema (plan type).

Variable focusing heuristics select which binding(s) for plan variables should be pursued when there are multiple possible bindings for a variable. This means that they select among competing plan versions—i.e., competing instances of a method. Each variable heuristic is associated with a specific set of variables of a particular control plan schema. Variable focusing is invoked whenever there are competing bindings for a variable that is used in a subgoal to be posted or in a subgoal grammar condition. Competing variable bindings are represented as special units called *multiple-valued values*. Actions return multiple-valued values whenever focusing should be invoked to select a plan parameter value. In effect, a multiple-valued value is a representation of uncertainty about the correct binding for a parameter.

In our control planning framework, focusing is viewed as a *meta-level* process that operates on top of the basic planner. That is, focusing decision points are not explicitly denoted in the control plans, but occur as part of the control of the planning process. For example, instead of a plan such as: (:SEQUENCE Identify-Available-Sensors, *Choose-Sensor*, Activate-Sensor) we leave the sensor choice step implicit in the focusing meta-process (:SEQUENCE Identify-Available-Sensor, Activate-Sensor). This approach eliminates the need to write these meta-steps into the plans and it means that all meta-reasoning is handled in a consistent manner, outside of the control plans. Because of this, it is easy to apply the refocusing mechanism to all three types of focusing decisions. Were variable focusing handled in the control plans, the more complex refocusing operations would be difficult to implement.

One drawback of this approach is that it may create confusion over the type of value that should be returned by certain actions since the value determines whether focusing will be applied or not (i.e., depending on whether the value is a multiple-valued value or not). For example, the action that satisfies the subgoal Identify-Available-Sensor would differ from the one that satisfies the subgoal Identify-Available-Sensors when multiple sensors are identified. With the subgoal, Identify-Available-Sensor, the multiple sensors must be represented as a single multiple-valued value rather than as a set. This may not seem to be the natural way of thinking about such an action, but if we think of the plan in terms of goals rather than actions, our format becomes more reasonable. Thus, the two steps Identify-Available-Sensors and Choose-Sensor really accomplish the goal of having a sensor that can be activated, so our plans would be written as (:SEQUENCE Have-Available-Sensor, Have-Activated-Sensor). See [31] for a different view of this same issue.

As we stated earlier, we believe that a planning approach to control provides an excellent framework for defining and applying complex heuristic control knowledge. We index our heuristic focusing knowledge by plan schema, by decision type, and by the particular instance of that decision type within the plan schema. In addition, heuristics may examine the planning structure to determine the exact goal or purpose of the decision they are making. This allows us to structure the control knowledge based on the exact *purpose* of each focusing decision in terms of its role in satisfying the system goals and subgoals. As a result, focusing nondeterminism can be limited to individual heuristics and handled through the refocusing mechanism (which supports intelligent backtracking).

For example, one type of control decision that interpretation systems must repeatedly make is to select sensor data to undergo interpretation. Interpretation systems that use a global decision scheme would have, in effect, heuristic knowledge for selecting data that would be written something like "prefer acoustic data in time slices with low data density" (though this heuristic may be embedded in the rating functions of the system rather than being explicitly represented). The problem with general, global heuristics like this is that we may end up with conflicting heuristics. We have been able to implement a number of highly context-specific data selecting strategies using our focusing framework. These heuristics make it clear that there is no single way to categorize the "goodness" of data without reference to the specific purpose the data is to be put to.

One example of where different strategies should be used for different contexts is in the selection of data for identifying possible additional aircraft versus the selection of data for resolving uncertainty about an already identified potential aircraft. For acoustic sensors, two characteristics of the data that help to determine whether the data is desirable to examine are the relative loudness with which the data is sensed and the density of data for a particular time slice. However, the relative importance of these characteristics really depends on the purpose the data is to be put to. For instance, loudly sensed data tends to have less uncertainty about position and frequency, but under many conditions (e.g., a battlefield) "noise" is just as likely to be loudly sensed as aircraft. On the other hand, the lower the data density in a time slice, the less likely it is that randomly selecting a cluster from the time slice will yield "noise" data. We have implemented a range of data selection strategies that use different purpose/goal-dependent criteria to evaluate the data—e.g., when resolving position uncertainty versus looking for additional aircraft.

Other context-specific strategies we have written limit the amount of frequency spectrum data that is processed in providing additional evidence for interpretation hypotheses. When identifying new aircraft or when looking for additional evidence for a very uncertain track hypothesis, the system should try to identify as complete a frequency spectrum for the aircraft as possible since this results in the least uncertainty from this set of data. However, tracking aircraft over time can provide even better evidence than spectral completeness due to the fact that spectra are often not completely sensed (so there is residual uncertainty resulting from a single spectrum). We have taken advantage of such an approach via heuristics that select plans that work in a data-directed way and process complete spectra when attempting to identify new aircraft or when working on highly uncertain track hypotheses, but select goal-directed plans that identify key portions of the spectra to process when tracking reasonably supported aircraft.

Another goal-directed strategy that we have implemented is able to limit the amount of data that the system examines to meet its termination criteria. We introduced this strategy to show how a highly goal-directed control mechanism is able to deal with large amounts of noisy data. The strategy is to limit the system to (completely) examining only every "nth" data time slice. The basis for the strategy is that data from an actual aircraft will extend over a number of time slices. Thus, in order to be reasonably certain that all possible vehicles have been identified, it is not actually necessary to examine all of the data since the data not initially examined will be identified and interpreted through the tracking process (i.e., forming a complete vehicle track). Of course, this strategy may miss vehicles that travel quickly around the boundary of the area being monitored, but these can be identified by additional processing of any data that is close to the boundaries or examining only the boundaries of the time slices that are being "ignored."

The ability to encode goal-directed strategies that can reason in detail about the current situation has allowed us to deal with another issue that causes problems for most blackboard-based interpretation systems: the possibility of missing data as a result of signals being missed by sensors or faulty transmissions of information from the sensors. The possibility of missing data is difficult for data-directed systems to deal with because it greatly expands the search space so its consideration must be carefully controlled. As a result, blackboard-based interpretation systems have typically ignored the possibility of missing data at the cost of a certain amount of brittleness. Based on RESUN's explicit model of uncertainty and context-specific focusing heuristics, we have been able to successfully implement strategies that consider the possibility of missing data. This is done by making missing data assumptions only under particular conditions and doing it only as part of a plan which "understands" that this assumption has been made and must be verified (preventing uncontrolled expansion of the search space as a result of the assumption).

The context-specified nature of the focusing heuristics means that the expression of focusing knowledge is highly *modular*. This makes it much easier to encode and modify control knowledge than it is in systems based on global focusing schemes (e.g., blackboard scheduler functions). In addition, the explicit subgoals in the plan schemas help the user to identify what control knowledge he must provide to the system because they identify the points where focusing decisions may have to be made. For example, if a developer must supply a set of global (meta-level) heuristics for identifying good data, he can never be sure that he has covered all the situations that might occur. When using a planning approach, however, much more information is available because the subgoals that select data can be identified and the possible contexts in which these subgoals might appear can be determined from the defined plans.

D. Refocusing for Opportunistic Planning

The heuristic focusing mechanism described in the previous section provides the foundation for controlling the planner's search. However, this mechanism has limited capabilities for dealing with the uncertainty in the focusing decisions themselves because it cannot reconsider its decisions. When there is uncertainty about decision choices the only thing that can be done using the heuristics alone is to pursue multiple choices and hope that the correct choice is within the set. Search paths can be pruned by eliminating choices, but decisions cannot be reconsidered to add new choices. In addition, choices can only be pruned when one of the alternatives succeeds or fails—decisions cannot be based on the relative "quality" of the developing alternatives.

The most significant innovation in the RESUN planner is its *refocusing mechanism*. In effect, the refocusing mechanism provides the RESUN system with an intelligent, nonchronological backtracking scheme [4]. While the planner needs to have backtracking capability, we have not used an implicit backtracking mechanism in this planner. For example, chronological backtracking is not used because it would be inappropriate in a reactive planner. Also, backtracking should not be driven by *failure* alone. Methods may require long sequences of actions and it can become clear that they are not optimal long before they actually fail (they may still succeed but produce "poor" results). Our refocusing mechanism allows the system to set backtrack points so that it can control which focusing decisions may be reconsidered and can direct this reconsideration (i.e., shift its focus of attention) based on a wide range of conditions. In addition, because of the way that refocusing conditions are tested, this mechanism allows our planner to have opportunistic control capabilities.

The refocusing mechanism is activated when a focusing heuristic posts a *refocus unit* along with its decision choices. Focusing heuristics instantiate refocus units in order to define the decision point as a backtrack point and to define the conditions under which backtracking and reconsideration of the decision will occur. A refocus unit is associated with the particular decision point in the planning structure where it was created. Each refocus unit consists of a condition, a handler, and a removal condition. The *condition* is used to determine when to invoke the handler. The *handler* is used to reconsider the decision-point focus decision (the original focusing heuristic is not simply applied again). The *removal condition* is used to determine when the refocus unit is no longer appropriate.

Using the refocusing mechanism, we can extend the focusing mechanism to allow decisions to be postponed and/or reconsidered. Focusing decisions can now be viewed as being in one of three classes: absolute, postponed, or preliminary. *Absolute* focus heuristics operate as described in Section III-C—they do not use refocus units. These heuristics simply select one or more paths to be pursued with any pruning being made through the plan updating process.

In other cases, focusing heuristics may not be able to select the "best" alternatives given the information they have available. Instead, they may need to partially expand each of several competing options to gather more specific information about the situation before being able to select the best alternative. *Postponed* focus heuristics select multiple options to be pursued and post a refocus unit. The refocus unit effectively monitors the expansion of the alternatives to recognize when sufficient information is available to make a better decision. At that point, the initially selected choices are reconsidered and pruned (the handler may only partially prune the alternatives and post another refocus unit for further reconsideration).

By postponing the focusing decision in this manner, the planner can integrate information about the quality of the data (i.e., data-directed control factors) with its goal-directed focusing knowledge. Another way of looking at postponed focusing decisions is that the system is actually performing a *search* to find the best method (or method instance) to use. We will return to the notion of explicit method search later in this section.

In still other cases, focusing heuristics may be able to choose a single "best" choice to pursue, but recognize that their decision rests on certain assumptions about the situation or the likely effectiveness of the choice. Should these assumptions turn out to be incorrect, the choice should be reconsidered. *Preliminary* focus heuristics select a single choice and post a refocus unit. The refocus unit effectively monitors the situation to make certain that the assumptions underlying the initial choice remain reasonable. Should this not be the case, the refocus unit handler will be invoked to consider whether the original alternative should be continued or whether a different one of the original possible choices is now better—given the changed situation.

An example of a situation in which postponed focusing might be used is shown in the later refinement of the control plan in Fig. 2. There are two possible bindings for the variable ?region—i.e., two regions the track could be extended into (at either end of the existing track). Typically, the system will be uncertain about how to proceed because it cannot be sure which region will contain the best data for extending the track. To handle this situation, the focus decision may be postponed by selecting both options to be refined further and posting a refocus unit. The refocus unit will cause the alternatives to be reconsidered once they have been refined to a point where information about the actual data is available (but before the data is completely selected and inference actions taken). This point is identified to the system by the refocus unit condition that was defined by the focusing heuristic. When this condition becomes true, the refocus handler is invoked to evaluate the alternative regions again and select the best one to pursue further and actually use to extend the track.

The track extension situation again illustrates the kind of detailed, context-specific control decisions of which the system is capable. While we talked as if a track extension decision would always be postponed due to uncertainty about the correct decision, this is not the case. In some situations, the system's goals may give it a definite preference. For example, the aircraft may have the potential to attack friendly forces or time may be running out to determine an answer, so it is of primary importance to determine the aircraft's most recent positions (i.e., track it ahead in time). Even in situations where there is not goal-based preference, it may not be worth incurring the overhead of a search to determine the better choice. This will depend upon whether the search overhead is likely to be more than offset by better constraining the track (and thereby reducing uncertainty about further extensions).

Thus, the heuristic that we implemented took into account the system goals plus the uncertainty in the existing track in deciding whether to postpone the decision and perform a search. It also was able to recognize that if it should find during the search that interpretation work has already been done in one of the regions of interest, then it is worth pursuing that region first because of the time saved by not doing low-level interpretation. Because the focusing heuristic that decides whether to delay a decision about track extension is specific to the plan where this decision must be made, and because the refocusing condition is determined by the heuristic when it executes, the process of arriving at a decision can be very specific to the particular tracking situation.

One place in which preliminary focusing might be used is in conjunction with a plan like Eliminate-Hypothesis-SOUs shown in Figs. 2 and 3. The grammar for this plan contains an iteration construct that may result in the plan executing for an extended period of time, while it attempts to resolve the uncertainty in one particular hypothesis. Pursuing a single hypothesis to the exclusion of all other alternatives is not always a good idea, however, because it could result in the system missing other important domain activities or losing the opportunity to gather useful data. In fact, in Fig. 2 we showed two different plans that were applicable to the Have-Reduced-Hyp-Uncertainty subgoal and mentioned that they differed in grain size. Reduce-Hypothesis-SOUs does not include an iteration. It selects a single SOU in the hypothesis, attempts to resolve it, and then returns to its containing plan which decides what problem-solving uncertainty to work on next. This gives the system more flexibility because it is reconsidering which problem-solving uncertainties it should work on frequently. However, this can result in a significant amount of control overhead if the same hypothesis should be pursued for a number of cycles. With the refocusing mechanism, overhead can be reduced by using the large-grained Eliminate-Hypothesis-SOUs method—without losing opportunistic capabilities. A preliminary focus decision is used that limits the amount of effort expended on this plan by posting a refocus unit whose condition becomes true after a set limit on the amount of time has been expended. The handler could then reexamine the new situation and determine whether or not to continue pursuing the same hypothesis.

The refocusing mechanism can also be used to implement opportunistic control—i.e., dynamic, data/event-directed control decisions. This is accomplished by extending the refocusing mechanism so that refocus units are activated in a "demon-like" fashion and their conditions can refer to characteristics of the available data as well as the characteristics of the planning structure and the hypothesis structure. This makes it possible for the system to shift its focus of attention between competing goals and methods in response to the characteristics of the developing plans and factors such as data availability. For example, the Eliminate-Hypothesis-SOUs could be interrupted if new data should become available within a critical region (as well as if a fixed amount of time has been devoted to it as mentioned above). Active data gathering actions (see Section III-F) can also use this mechanism to "suspend" a primitive plan while waiting for the sensor to generate the data and then reactivate the primitive plan once the data becomes available.

Because of the search process inherent in constructing plans, a planning-based approach to control extends the search view of control (as compared with a conventional retrieval-refinement approach). Instead of just a search for the correct answer, the control process now also involves an explicit search for the best *methods* to use to determine the answer. This comes out clearly in the postponed focusing example mentioned above where the system performs a search for the best way to extend a track. In addition, the method search view of control leads us to view opportunism as arising naturally from explicit method search instead of as some special form of control that must be added to a system (e.g., through special opportunistic knowledge sources). Opportunism simply results from the use of particular types of conditions that cause the system to redirect its method search.

It may also be of interest to note that the notion of opportunism that we have differs somewhat from that in the BB1 paradigm [22] (discussed in Section V). In that system, opportunistic actions are allowed to post new "subgoals" for the system. While these new subgoals are implicitly part of the overall problem-solving goal of the system, the exact role of the subgoal need not be made explicit via connection to existing goals and plans. Because RESUN's refocus units are associated with particular focus decisions, all that they can do is refocus the planner within its existing subgoals. This seems to be the correct approach given a planning-based control mechanism since the top-level goal of the system is intended to embody the overall system and goal. Thus, it makes little sense to have subgoals being posted that are "independent" of the top-level goal and it is not clear how one would judge the merits of such a subgoal without an understanding of its role in the overall system goal.

E. Plan Schema Definitions

Nonprimitive control plan schemas are defined using specifications like those in Fig. 3. The Goal Form clause specifies the goal that can be satisfied by successfully completing the plan. Goal Forms are predicate expressions where the variables are denoted with a leading "?". Each instantiation of a control plan schema maintains a local environment with bindings for the variables listed in the Input Variables, Output Variables, and Internal Variables clauses. The Input and Output Variables clauses denote the status of the variables in the Goal Form expression. Input Variables are bound upon instantiation of a plan from the corresponding locations in a unifying (sub)goal expression. Output Variables must be bound upon successful completion of the plan. This allows plans to return information to higher level plans (this is discussed more below).

The sequence of subgoals that can be used to realize the goal of the plan is defined by the Grammar clause of the schema. This clause uses a shuffle grammar that can represent sequential, concurrent, conditional, and iterated subgoal subsequences. For example, the Grammar clause in Fig. 3 specifies that two subgoals, Have-Hyp-SOU and Have-Reduced-Hyp-SOU, must be sequentially satisfied (Have-Hyp-SOU must have been satisfied before Have-Eliminated-Hyp-SOU is attempted to be satisfied), and that this process will be iteratively continued until the degree of belief in the hypothesis pointed to by the variable ?hyp is greater than or equal to the belief goal represented in ?min-belief. Other grammar operators not appearing in the figure schema include: :SHUFFLE, :LIST-SHUFFLE, :CONDITIONAL, and :XOR (see [4]).

For each subgoal listed in the Grammar clause of a plan definition, a corresponding subgoal definition form must be provided. Subgoal definitions identify the Goal Form, Input Variables, and Output Variables for each subgoal. The function of each of these clauses is analogous to the function of the corresponding clauses in the plan definitions. The Goal Form is a predicate expression which represents the goal that must be accomplished to satisfy the plan subgoal. The Goal Form is used to identify those control plans that could be used to satisfy the subgoal. This is done by unifying the *instantiation* of the subgoal Goal Form (in a particular instantiation of a control plan) with the Goal Forms of the set of defined control plan schemas.

Primitive plans are defined using specifications similar to those for nonprimitive plans. The specifications of primitive plans include the Goal Form, Input Variables, Output Variables, and Constraints clauses just as for nonprimitive plans. Primitive plans also include a Function clause that identifies the (Lisp) function that implements the primitive plan (action). This function is passed the bindings for the Input Variables of the primitive and returns both a status value and a list of bindings for the Output Variables of the primitive (if any).

A number of clauses are permitted in RESUN plan schema definitions that do not appear in the example of Fig. 3. One set of clauses relates to the specification of constraints on the values of variable bindings in a plan instantiation. Rather than use general constraints that cannot be efficiently operationalized, constraint information is partitioned into categories that can be interpreted with special mechanisms. Six categories of constraints have been identified as being useful so far and are part of the current implementation: plan In-constraints, plan In-bindings, plan Out-bindings, subgoal Out-constraints, subgoal In-bindings, and subgoal Out-bindings. Examples of plan In-constraints, plan In-bindings, and subgoal Out-bindings are shown in Fig. 4. The plan In-constraints clause allows additional constraints to be placed on the goals that are considered to match the plan by allowing the specification of predicates that Input Variables' bindings must meet. In Fig. 4, the plan In-constraints limit the acceptable bindings of ?track-answer-sou. Similarly, the subgoal Out-constraints clause allows additional constraints to be placed on what are considered satisfactory results for a subgoal's matching plan. The In-bindings and Out-bindings clauses constrain and operationalize the relations between the variables of the plan. They specify the values that variables must have at particular points in the expansion of the plan in terms of the current bindings of other variables. In Fig. 4, the plan In-bindings obtain an element out of a structure that is bound to ?track-answer-sou and initialize the internal variable ?status. The subgoal Out-bindings in the figure determine an appropriate value for ?status based on the results of pursuing the subgoal.

Another form of condition that can be used in the control plan specifications is the plan *Precondition*. Preconditions differ from In-constraints in that they should be based on factors other than the values of subgoal parameters (Preconditions are only checked after a plan meets its In-constraints). However, they are not intended to guarantee that a plan will succeed. There is an obvious tradeoff between the complexity of the preconditions and the possibility of plan failure. Very complex precondi-

```
Plan Definition:
 Plan Name       Complete-Track-Answer-Hyp
 Goal Form       (Have-Reduced-PS-HYP-SOU ?track-answer-sou ...)
 In-constraints  (and (typep ?track-answer-sou 'uncertain-answer-ps-sou)
                      (typep (uncertain-answer-ps-sou-answer-hyp ?track-answer-sou) 'track-hyp))
 In-bindings     ((track-hyp . (uncertain-answer-ps-sou-answer-hyp ?track-answer-sou))
                  (status . :unknown))

Subgoal Definition:
 Subgoal Name    Have-Extended-Track-Ext
 Goal Form       (Have-Reduced-Ext-SOU ?track-ext ... ?extended-track-ext ?status)
 Out-bindings    ((track-ext . (if (eq ?status :failed) ?track-ext ?extended-track-ext)))
```

Fig. 4. Examples of constraint clauses in control plan definitions.

tions could be used to guarantee that a plan would succeed (barring unanticipated changes in the world). This may require, though, that the precondition do nearly the same work as would be required for achieving the solution (without actually generating a solution). Furthermore, placing significant reasoning into Preconditions would cause this reasoning to fall outside the control of the focusing heuristics and refocusing mechanism. Finally, complex Preconditions can adversely affect the modularity of the plans.

In the control plans that we have defined, we have used few preconditions. Instead, we have written plans in which the preceding steps of the containing plan implicitly satisfy the "preconditions" of the subplans or else the "evaluation" of these conditions is simply handled through the planner's search process. Consider, for example, a plan that selects acoustic sensor data to be used to extend an existing Track hypothesis. One way to define this plan would be without any Precondition clause. In this case, the plan would first take actions to determine the available sensors and then examine the characteristics of their data to find some data—if any—that meets the criteria. This plan will fail if there are no acoustic sensors available or if there are acoustic sensors, but none of them have acceptable data. These failures would be handled by the heuristic focusing process which allows the user to define backtrack points to reconsider decisions when failures occur. Alternatively, one might include a Precondition clause for this plan that confirms that there are at least some currently working acoustic sensors. This is the type of condition that is intended to go into the Precondition clause, one that involves a computationally simple action that would reduce the possibility of the plan failing.

On the other hand, the Precondition clause for this plan could be extended so that it not only checks if there is an active sensor, but actually examines the sensor data to see if there is data for the time of interest and the region of interest. This would guarantee the success of the plan, but it would be extremely wasteful because it has to do much of the work of the plan—without producing information in a form necessary to complete the plan. Also, the order in which sensors and data points are examined could not be controlled as it could be using the focusing mechanism during planning. Thus, there is a tradeoff between the sophistication of preconditions and the ability to define intelligent and flexible control strategies. Preconditions, when used, should embody easily verified conditions that would definitely rule out the applicability of a method. They should not involve a search process—e.g., seeing whether there exists some sensor that has data with particular characteristics.

Failure conditions are another form of plan condition that are not used in the example specification in Fig. 3. Failure conditions provide a method for the plan writer to deal with subgoal interactions that cause an in-progress plan to fail. A Failure condition is a predicate that will become true when an instance of the plan is definitely not going to successfully complete. When a control plan has an associated Failure condition, the predicate will be evaluated prior to each expansion of an instantiation of the plan to verify that the plan instance is still viable. Using these conditions, the system can be prevented from expending effort on a plan instance once conditions become such that the plan must eventually fail. For example, the Failure clause can be used to deal with subgoals whose goals need to remain satisfied during some portion of the plan, once they have been achieved. The Failure condition for such a plan would include predicates that determine the current stage of the plan and determine whether the necessary subgoals have been undone (by other plans). This approach allows more flexibility than would the definition of a protection interval since it might allow for achieved goals to be temporarily undone if permissible, and would keep suspended plans from interfering with active plans. Failure conditions have not been exploited in the current set of aircraft monitoring plans.

The other side of subgoal interactions is when the actions to satisfy one subgoal result in other subgoals being satisfied as well. To account for these *serendipitous* events, we allow the specification of *Satisfaction* conditions within each subgoal definition. Satisfaction conditions are evaluated when a subgoal is instantiated and are then instantiated as demons that act whenever an active subgoal is satisfied. In general, satisfaction conditions are applicable to all subgoals except those that are guaranteed not to be satisfied by an independent process. This is the case for all information gathering goals—i.e., any goal that is done only for the results it returns and that has no side effects. The subgoal Have-Hyp-SOU in Fig. 3 is such a goal. Failure to specify satisfaction conditions does not change the semantics of a plan; it merely means that goals may be achieved multiple times by the control process.

In classical planners, detecting and handling subgoal interactions is a major source of complexity [36]. Because of the uncertainty in most problem-solving domains, we limit the consideration of subgoal interactions during the plan expansion and refinement process. While the focusing heuristics may reason about potential interactions (like resource interactions), subgoal interactions are generally handled by replanning when interactions actually occur—instead of trying to detect and prevent them. This is the purpose of the Failure and Satisfaction conditions. A similar approach has been used in other reactive planners—e.g., Firby's RAP system [16]. It is important to realize, however, that this approach requires the kind of intelligent backtracking capabilities that are provided by our refocusing mechanism.

F. Actions

Incremental planning makes it possible to deal with the uncertainty about the outcome of actions that is an inherent part of most problem-solving systems. Because each plan is expanded only to the point where the next action resulting from the plan can be identified prior to executing that action, the outcome of actions is able to influence later plan expansion. One way that RESUN actions can influence further plan expansion is through their *execution status*—actions may succeed or they may fail. A failed action will cause its matching subgoal and the subgoal's containing plan to fail unless the failure is handled via the refocusing mechanism. Successful actions can also influence later plan expansion by returning results that are used to bind plan variables. Plan variable bindings affect the subgoals that are posted by later plan expansion via the variables in the subgoal forms. Plan variable bindings may also affect plan expansion via the plan grammar constructs that include conditions (:ITERATION, :XOR, :CONDITIONAL) since these conditions may reference the current values of plan variables.

Each RESUN primitive can be categorized into one of three classes: inference actions, information gathering actions, and data-gathering actions. *Inference actions* are the standard domain problem-solving actions—e.g., making evidential inferences that create or modify hypotheses on the interpretation blackboard. Inference actions are taken primarily for their side effects—e.g., making an evidential inference that results in the creation or modification of hypotheses. However, inference actions may also return results that influence later planning. For example, even when interpretation inferences "fail" they still generate (negative) evidence for hypotheses. It may be important for further planning to know that negative evidence resulted from an inference action, so the system may not want to consider an action to have "failed" (in the sense that its matching subgoal and containing plan will also be considered to have failed) but simply know the type of results produced.

Inference actions correspond to the knowledge sources (KS's) in a conventional blackboard system. However, the functions that implement inference actions are not identical to the functions that would be used to implement standard blackboard system KS's. A KS has two major components: a precondition and an action. Inference action functions only include the action portion of the KS. The precondition portion of a KS determines the applicability of the KS and creates bindings of KS variables to be used in instantiating KS's [creating knowledge source instantiations (KSI's)]. In a planning-based system, the precondition functionality of a KS is accomplished by the planning process; inference actions are only selected when they can satisfy subgoals that planning has determined are appropriate and variable binding occurs via the unification of goal forms.

Unlike inference actions, *information gathering actions* are taken solely for the results that they return and should not have any side effects. They are invoked by control plan schemas at those points where the system needs information about the current state of the world—e.g., the SOU's in a hypothesis, the availability of data or sensors, etc. Results from these actions are bound to variables that will be used in later subgoals. The primitive action Identify-Hypothesis-SOU of Fig. 2 was an information gathering action that was used to integrate symbolic uncertainty information into the planning process.

The use of explicit information gathering actions enhances the reactivity of the planner in two different ways. First, tractability of the planner is enhanced because the system does not maintain a complete world model (this is a major source of complexity in classical planners due to the frame problem [36]). The RESUN planner maintains only partial state information through the variable bindings in the individual plans; each plan models just what it needs to know about the world and keeps it sufficiently up-to-date with information gathering actions. The second way in which information gathering actions enhance reactivity is by giving the system a way to deal with the possibility of external agents affecting the world—e.g., sensors continuously making new data available to an interpretation system. Even when state-based approaches can be made computationally tractable, they cannot deal well with dynamic and unpredictable domains.

Data gathering actions are used to instruct active sensors to gather additional data. These actions can be used to tune the parameters and the positioning of appropriate sensors in order to gather the best possible evidence to resolve particular uncertainties. For example, an aircraft monitoring system could adjust the dwell time of a radar unit in order to get the most precise position data for an aircraft if that is a key uncertainty that the system needs to resolve. The ability to make effective use of active gathering of data requires a system that understands how it wants to use the data. The goals and subgoals of the planning structure provide this information. While data gathering actions are also domain actions in one sense (as opposed to the information gathering actions that are used by planning only) and might just be considered as inference actions, one of the reasons that we have set them apart from normal inference actions is that they may take a substantial amount of time to complete. As we mentioned in Section III-D, our refocusing mechanism can be used to change the focus of attention of the system while it is waiting for a data gathering action to complete so that the system is not idled.

IV. Status and Experimental Results

We originally implemented the RESUN control architecture described in this paper using a simulated aircraft monitoring application [3], [4]. The implementation is in Common Lisp on a Texas Instruments Explorer using GBB [17] for the domain blackboard. Aircraft monitoring is an appropriate application for evaluating this architecture because it has characteristics that are difficult for other blackboard control frameworks to handle because they require highly context-specific, goal-directed control: many potential alternative interpretations of data (due to the modeling of ghosting, noise, and signals missed by sensors); complex interactions among competing hypotheses; the availability of a wide variety of methods for resolving uncertainty (hypothesize and test, differential diagnosis, etc.); passive sensors continuously generating large amounts of data; and active sensors that can be controlled by the system. The current interpretation methods are defined in terms of more than 30 control plans with about 100 subgoals and approximately 40 primitives.

More recently, the RESUN framework has been used in the development of the integrated processing and understanding of signals (IPUS) signal understanding testbed [30]. IPUS is currently being applied to the understanding of household sounds that would be required by a robot. While this is also an interpretation application, it is different enough from aircraft monitoring that certain features of the RESUN mechanism have been more fully exercised. One of the key differences is that in IPUS the basic data from sampling the environment can be processed many times using a number of algorithms and parameter settings. Reprocessing of data must be tightly controlled, driven by particular uncertainties in the developing high-level models [32].

As we said in the introduction, the question of how well a problem solver supports the use of sophisticated control strategies is largely an issue of heuristic adequacy. Because of this, it is difficult to make definitive statements about the RESUN control planner from any limited set of experiments. Nevertheless, some conclusions can be drawn from our experiences to this point. First, the RESUN framework can indeed be used to encode context-specific, goal-directed strategies that would be difficult to implement using conventional (nonplanning-based) blackboard control mechanisms. Second, these kinds of "sophisticated strategies" can be applied without incurring unacceptable overhead—at least for interpretation applications. In other words, the increased cost of control was more than offset by the ability to better constrain the search for correct interpretations. Third, the refocusing mechanism does make it possible to obtain opportunistic control capabilities from a planner. Fourth, the framework is usable in practice on realistic problems. A description of the aircraft monitoring experiments is in [4].

One purpose of the experiments was to evaluate whether or not the RESUN framework could be used to implement sophisticated interpretation strategies and to analyze the importance of the different features of the system. In part, we judged the appropriateness of the framework by showing that the system could be used to implement a wide range of strategies for resolving uncertainty (based on the SOU's). Furthermore, we have not yet come up with any strategies in either aircraft monitoring or IPUS that we wanted to implement, but could not. We have, however, identified several additional plan sequencing constructs that would make it easier to write certain strategies. Another way that we judged the framework was by showing that strategies could be written that make the system's actions very responsive to: the termination criteria (i.e., the overall system goals), the amount and the characteristics of the data, the *a priori* likelihoods of alternative explanations of data, and the need for timely actions in real-time situations. In analyzing the performance of the system, we find that the major source of performance improvements comes from the ability to use highly context-specific, goal-directed control knowledge. However, both explicit method searches (via postponed focusing decisions) and opportunistic refocusing are crucial in allowing the use of such goal-directed control since there are situations in which such control would perform very poorly.

A key issue for the use of sophisticated control strategies and deliberative control mechanisms is whether the increased cost of control is worth it—i.e., whether the cost of such control is more than offset by better constraining the search. To evaluate this question, we ran experiments using sophisticated strategies that we compared against benchmark experiments in which the strategies did not make use of any of the sophisticated features of our framework. We attempted to make these benchmark control strategies correspond to the kinds of strategies that are used in more conventional blackboard systems—e.g., (earlier versions) of the distributed vehicle monitoring testbed (DVMT) [29], a system that used a similar vehicle monitoring task. The experiments showed that with relatively simple data scenarios, the application of context-specific strategies could reduce CPU time by more than 20% and the number of hypotheses created (a measure of the amount of the search space being explored) could be reduced by 45%. With more complex data scenarios involving a good deal of noise, CPU time could be reduced by more than 50% using highly goal-directed strategies for limiting the amount of data examined by the system. Thus, for this interpretation application at least, the cost of more sophisticated control is justified. The experiments also demonstrated that postponed focusing with a resulting method search can be used to improve performance despite increasing the control costs.

Another way of judging the performance of a control mechanism is in terms of control overhead: the amount of CPU time spent reasoning about what to do next as a percentage of the total CPU time it takes to solve the problem (which includes the time to actually take actions as well). However, while we often talk about overhead, it is important to remember that the appropriateness of a set of control strategies is really dependent on the overall performance of the system; there is no absolute level that defines acceptable or excessive overhead. In addition, comparisons between systems is problematic since overhead depends in part on the relative costliness of actions in particular application domains, and because few implementations have been optimized. Despite these caveats, it may still be useful to look at overhead figures from our experiments. In the benchmark strategy experiments, overhead ranged from 12–19%. In the experiments that used complex control strategies, overhead reached 30–35%. Overhead was lower in both cases when using larger data sets. IPUS uses much more computationally intensive KS's than the aircraft monitoring application—i.e., numeric signal processing algorithms like Fast Fourier Transforms. Experiments with IPUS have produced control overhead of around 10%. Thus, this data suggests what one would expect: sophisticated control is most likely to be effective (appropriate) for problems that involve larger search spaces (e.g., large amounts of interpretation data) and in which the cost to search is high (e.g., computationally expensive KS's).

One of the most exciting conclusions from the experiments was confirmation that the combination of explicit control plans with context-specific focusing heuristics provides an extremely flexible framework for developing interpretation strategies. The modularity of the control plans and the focusing heuristics has made it easy to write highly context-specific, goal-directed strategies as compared with systems using global focusing mechanisms (e.g., conventional blackboard scheduling functions). It also supports the incremental development of complex control strategies. Another related issue is that for the knowledge acquisition phase, having planning-based control makes it easier to recognize when the system's control strategies/methods are inadequate since this results in planning failures. With agenda-based blackboard control schemes, it can be difficult to determine whether adequate strategies have been defined because a system may take actions even if they do not contribute to solutions. One limitation of the current system is that it does not include any default backtracking strategies that could deal with the possibility of inadequate methods. We believe that the modularity of the framework should make it relatively easy to implement such strategies since they can be defined independent of the rest of the control knowledge.

V. Related Research

As we noted earlier, "classical planning" research is not directly applicable to the control of problem solvers because it involves strategic planning. Strategic planning represents one extreme on the continuum between completely deliberative and completely reflexive control. One response to the combinatorics of classical planning and its problems in dealing with uncertain and dynamic situations has been to go to the other extreme and argue that planning is not needed at all. In the *reactive control* or *situated action* approach [1], [34], actions result from rules or procedures whose invocation is based solely on properties of the immediate situation. This is equivalent to having a totally data/event-directed control mechanism and it seems clear from research on blackboard systems that this approach is insufficient for much problem solving [7], [12], [22], [25]. An intermediate approach that has received considerable recent attention is often referred to as *reactive planning* [16], [18]. Proponents of this approach recognize the limitations of planning based on complete world models and perfectly predictable actions, but they argue that planning is still necessary for intelligent control.

One approach to reactive planning is Firby's *RAP* system [16], [20]. Reactive action packages (RAP's) are similar to our control plans and the RAP expansion process is similar to RESUN's basic planning process. The RAP system emphasizes the importance of checking the state of the world via conditions associated with the RAP's that are, again, very similar to the Precondition and Satisfaction conditions of our control plan schemas. The major drawback of the RAP approach is its lack of focusing mechanisms: there is no framework for encoding knowledge to make decisions when multiple RAP's are applicable to a subgoal, no way to control the instantiation of RAP's, and no opportunistic mechanisms for suspending or terminating RAP's.

Another approach to reactive planning is the *procedural reasoning system* (*PRS*) of Georgeff *et al.* [18], [19]. PRS is also a script-based, incremental planner and like RESUN it is both opportunistic and reactive. The main difference is in the way that plans and control knowledge are invoked. Both plan schemas and (meta-level) control knowledge are stored in *knowledge areas* (*KA*'s) which function very much like control KS's in the BB1 system (see below). PRS differs from BB1, though, in directly selecting (domain) actions and in using a more complex KA invocation scheme than an agenda. PRS also includes a database of its beliefs about the world and it views plan and subgoal instantiations as statements of intention. Because these internal models can be referred to in KA invocation conditions, "meta-level KA's" in PRS can do reflective reasoning to choose among multiple relevant KA's, decide whether additional reasoning (overhead) is acceptable, etc. However, because the application of all control knowledge is event/state triggered, it seems likely that PRS would be less efficient for indexing and applying large amounts of application-specific control knowledge (e.g., at nearly every decision point) than RESUN's "hardwired" mechanisms. On the other hand, because meta-level KA's can be interrupted, PRS is able to place an upper bound on its response time—an important feature for applications like robot control.

Relatively few "problem-solving systems" have made use of planning-based control. One early example is McDermott's *NASL* system [31]. *NASL* was a script-based, incremental planner, but it was neither opportunistic nor reactive. Planning was controlled by *choice rules* that were indexed/retrieved like plan schemas. The main focus of NASL was to explore a view of planning that is different from the conventional search view. McDermott saw human problem solving as a process of "try something, wait until an error has been made, and then correct it." As a result, choice rules always selected a single plan to achieve a *task* (goal) and choice decisions could not be reconsidered (e.g., through backtracking).

Instead, ''planning errors'' were dealt with by attempting to ''restate'' a task. ''Execution errors'' resulted in ''error-correcting'' subtasks being added to the failed task. These error-correcting subtasks were treated just like any other task—i.e., as a problem to be solved. NASL used a theorem proving mechanism for plan retrieval, but McDermott recognized that this was inappropriate because plan retrieval needed to be simple and efficient.

Clancey's work on NEOMYCIN was an attempt to provide more structure to Davis' meta-rules format through a planning-based mechanism. Control knowledge in NEOMYCIN was defined in terms of *tasks* [9], [10], which were made up of a sequence of conditional actions represented as meta-rules. A prime motivation for NEOMYCIN was the recognition that similar ''strategies'' (what we have called ''methods'') were being repeatedly encoded in meta-rules that were relevant to different situations. The task/meta-rule format provides a more abstract and explicit representation of possible control methods than meta-rules alone and is somewhat similar to our control plan/subgoal format. However, in the task/meta-rule framework, meta-rules directly invoke subtasks—i.e., the meta-rules not only identify the applicable methods, but also encode the strategies for selecting the methods. As a result, the framework does not support method search nor intelligent backtracking as the RESUN framework does.

The desirability of planning for control of problem-solving systems was recognized early on in blackboard system research. For example, Hearsay-II included special mechanisms that always scheduled certain knowledge sources together [28]. The goal-directed blackboard architecture of Corkill and Lesser [12] included subgoaling and precondition-action backchaining mechanisms (that are an implicit part of any planning frameworks), but did not construct plans. The incremental planning approach of Durfee and Lesser [15] for a blackboard-based vehicle monitoring system builds abstract models of the data and uses these models to develop plans that guide the selection of knowledge sources. Planning in their approach effectively occurs at two different levels of abstraction. First, the data abstraction model is analyzed to identify the possible alternative vehicle tracks and their relative likelihoods so that plans for incrementally constructing these tracks can be formed. These plans consist of ''intermediate-level'' goals to develop appropriate supporting vehicle hypotheses. A second level of planning is then used to choose knowledge source sequences to meet these intermediate-level goals. Because both the data abstraction model and the planning mechanism here are closely tied to vehicle tracking, though, their application is limited.

The BB1 system [22], [24], [35] is the first blackboard framework to use a true planning-based approach to control and it remains one of the most sophisticated blackboard control frameworks. The control planning mechanism of BB1 extends the blackboard model with a separate control blackboard and control knowledge sources. BB1 maintains the opportunism of classic blackboard models because the identification of possible domain *and* control actions is done in a data/event-directed fashion via a (single) agenda. Because it is based on an agenda mechanism, planning in BB1 is somewhat different from planning in RESUN and classical planners. Refinement of BB1 control plans does not result in the direct identification of actions. Instead, refinement of BB1 control plans results in the posting of ''foci.'' Associated with each of the foci is a set of ''heuristics'' that are used to rate the potential actions on the agenda. Thus, the lowest level ''subgoals'' in BB1 plans (i.e., the foci) are often somewhat general and may influence the selection of several actions—e.g., ''prefer actions that do work in region x.''

BB1's control plans provide a more explicit and modular representation of strategy knowledge than the other blackboard control frameworks mentioned above. However, control reasoning is typically less explicit than in RESUN. This is because goal and strategy conflicts are often resolved via numeric ratings calculations—action ratings are a combination of the ratings from each multiple active foci—rather than explicit decisions. BB1 could ''directly'' select actions if the foci gave a nonzero rating to only a single action and if control KS's were added that dynamically changed the combination functions to explicitly resolve all ''goal conflicts.'' However, BB1 would be less efficient than RESUN for this style of control (e.g., actions would have to be identified by searching the agenda, whereas they are directly linked to RESUN subgoals). In part, this is because BB1 and RESUN have taken different approaches to maintaining opportunistic control capabilities in a planning mechanism: an agenda mechanism versus refocusing. BB1 is more general and flexible than RESUN, but potentially less efficient for RESUN's particular style of control.

VI. Conclusion and Future Research

Based on our experience with two complex interpretation domains, we have found the RESUN control architecture to be very powerful. We have been able to successfully encode numerous sophisticated control strategies. For example, we have been able to implement strategies that require detailed reasoning about the state of problem solving (like differential diagnosis), limit the amount of data that is processed, and control the portion of the search space that is examined (so that more realistic models of the domain that include the possibility of missing data can be used). The modularity of the framework has made it relatively easy to incrementally construct and debug these strategies. We have also found that this kind of planning mechanism can, in fact, be used in realistic domains without incurring excessive overhead.

The RESUN control planner maintains the traditional capabilities of planning approaches, but also supports opportunistic control. Among the key advantages of planning-based control are: the goal/plan/subgoal tree provides explicit context for decisions, the selection of actions is driven by explicit subgoals, sequences of actions can be coordinated to satisfy goals, control knowledge is separated into method and strategy components, and control (domain action) decisions result from plan refinement decisions (which may involve a search process). Because of these characteristics, control decisions are highly context-specific and control knowledge is very modular. This helps support more explicit control reasoning, and eases the creation and refinement of complex strategies.

The refocusing mechanism is one of the key innovations in RESUN. Refocusing gives the system the ability to postpone focusing decisions until it can gather more information about the particular situation by performing a search for the best methods (plans) to pursue. The ability to perform explicit method searches allows the system to control its overhead because the system can reason about the appropriate balance between decision uncertainty and the overhead of method search. This view of control makes it clear that problem solvers are not just performing a search for "the answer," but are also searching to find the best methods to use to determine the answer. The refocusing mechanism also provides RESUN with opportunistic control capabilities since refocusing conditions can include data/event-related factors and refocus units are handled like interrupts (the posting of a refocus unit is effectively like adding a temporary, special-purpose BB1 control KS). Using the refocusing mechanism for opportunistic control means that opportunism results from the ability to dynamically refocus the method search.

Though the RESUN control planning framework has been presented here within the context of sensor interpretation and blackboard systems, we feel that it is has more general applicability. The RESUN focusing and refocusing mechanisms provide a kind of "language" for expressing control knowledge: decision-specific (plan/subgoal) heuristics, postponed decisions allowing method search, and opportunistic shifts of focus of attention. We found that this way of structuring control knowledge was natural and appropriate for developing complex strategies. A key issue that must be resolved to use a planning-based approach for other problem-solving domains is whether it is possible to construct explicit subgoal hierarchies for these domains. In RESUN, these subgoals were largely supplied by the symbolic model of uncertainty in interpretation inferences.

A number of research directions are currently being explored. One of the most important is the development of an appropriate nonprocedural language for expressing focusing heuristics—particularly those for real-time problem solving. As part of this research, we are trying to understand how to represent the factors that affect control decisions and the relationships between alternative choices. In addition, since focusing decisions may involve making choices about the best decision procedure to use and may require that evidence be gathered to make this choice, we are studying the extension of the planning paradigm to the focusing decision level.

Sophisticated real-time control may also require the use of a meta-level process that selects focus points to pursue based on their resource requirements (the current approach is to cyclically allocate one plan expansion cycle to each focus point). Another issue which remains to be explored is that of actually modifying system goals when dealing with resource limitations in real-time interpretation. For example, when the time allotted to produce an answer is running out, the system might want to accept a lower level of belief for "answer" hypotheses so that it can provide a broad, but less in-depth, solution. This requires a meta-level process that adjusts the parameters in the top-level goal form and then propagates these changes and updates plan states. Currently we simply use focusing heuristics that key off of the goals and resources to determine how to proceed—rather than adjusting the goals.

We have also developed a distributed version of the RESUN system [16]. The goal/plan/subgoal structure (along with RESUN's symbolic model of uncertainty) provides the system with a rich language for communicating with other agents about the state of its problem solving. Another area of interest that we are currently pursuing is the development of a parallel architecture implementation of RESUN. The ability to have multiple active focus points due to concurrent subgoals and parallel method search provides a good basis for implementing parallelism.

References

[1] P. Agre and D. Chapman, "Pengi: An implementation of a theory of action," in *Proc. AAAI-87*, pp. 268-272, 1987.

[2] B. Buchanan and R. Smith, "Fundamentals of expert systems," in *The Handbook of Artificial Intelligence, Vol. 4*, Avron Barr, Paul Cohen, and Edward Feigenbaum, Eds. Reading, MA: Addison-Wesley, 1989.

[3] N. Carver and V. Lesser, "Control for interpretation: Planning to resolve uncertainty," Tech. Rep. 90-53, Dept. of Comput. Infor. Sci., Univ. of Massachusetts, 1990.

[4] N. Carver, "Sophisticated control for interpretation: Planning to resolve uncertainty," Ph.D. thesis, Dept. of Comput. Infor. Sci., Univ. of Massachusetts, 1990.

[5] N. Carver and V. Lesser, "A new framework for sensor interpretation: Planning to resolve sources of uncertainty," in *Proc. AAAI-91*, pp. 724-731, 1991.

[6] N. Carver, Z. Cvetanovic, and V. Lesser, "Sophisticated cooperation in FA/C distributed problem solving systems," in *Proc. AAAI-91*, pp. 191-198, 1991.

[7] N. Carver and V. Lesser, "The evolution of blackboard control," to appear in *Expert Systems with Applications*, special issue on The Blackboard Paradigm and Its Applications, vol 7, no. 1, 1993 (also available at Tech. Rep. 92-71, Dept. of Comput. Sci., Univ. of Massachusetts, 1992).

[8] W. Clancey, "Heuristic classification," *Arti. Intell.*, vol. 27, pp. 289-350, 1985.

[9] W. Clancey and C. Bock, "Representing control knowledge as abstract tasks and metarules," Tech. Rep. KSL 85-16, Knowledge Systems Lab., Dept. of Comput. Sci., Stanford Univ., 1986.

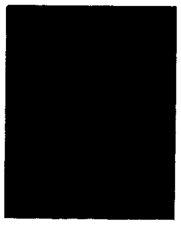
[10] W. Clancey, "From GUIDON to NEOMYCIN and HERACLES in twenty short lessons," *AI Mag.*, vol. 7, no. 3, pp. 40-60, 1986.

[11] P. Cohen and E. Feigenbaum, Eds., *The Handbook of Artificial Intelligence, Vol. 3*. Los Altos, CA: Kaufmann, 1982.

[12] D. Corkill, V. Lesser, and E. Hudlicka, "Unifying data-directed and goal-directed control: An example and experiments," in *Proc. AAAI-82*, pp. 143-147, 1982.

[13] R. Davis, "Meta-rules: Reasoning about control," *Arti. Intell.*, vol. 15, pp. 179-222, 1980.

[14] J. Doyle, "A model for deliberation, action, and introspection," Ph.D. thesis, Tech. Rep. AI-TR-581, Arti. Intell. Lab., M.I.T., 1980.
[15] E. Durfee and V. Lesser, "Incremental planning to control a blackboard-based problem solver," in *Proc. AAAI-86*, pp. 58-64, 1986.
[16] R. J. Firby, "An investigation into reactive planning in complex domains," in *Proc. AAAI-87*, pp. 202-206, 1987.
[17] K. Gallagher, D. Corkill, and P. Johnson, *GBB Reference Manual*, Tech. Rep. 88-66, Dept. of Comput. Infor. Sci., Univ. of Massachusetts, 1988.
[18] M. Georgeff and A. Lansky, "Reactive reasoning and planning," in *Proc. AAAI-87*, pp. 677-682, 1987.
[19] M. Georgeff and F. Ingrand, "Decision-making in an embedded reasoning stem," in *Proc. IJCAI-89*, pp. 972-978, 1989.
[20] S. Hanks and R. J. Firby, "Issues and architectures for planning and execution," in *Proc. DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pp. 59-70, 1990.
[21] B. Hayes-Roth and F. Hayes-Roth, "A cognitive model of planning," *Cognitive Sci.*, vol. 3, no. 4, pp. 275-310, 1979.
[22] B. Hayes-Roth, "A blackboard architecture for control," *Arti. Intell.*, vol. 26, pp. 251-321, 1985.
[23] B. Hayes-Roth and M. Hewett, "BB1: An implementation of the blackboard control architecture," in *Blackboard Systems*, Robert Engelmore and Tony Morgan, Eds. Reading, MA: Addison-Wesley, 1988.
[24] B. Hayes-Roth, R. Washington, R. Hewett, and M. Hewett, "Intelligent monitoring and control," in *Proc. IJCAI-89*, pp. 243-249, 1989.
[25] F. Hayes-Roth and V. Lesser, "Focus of attention in the Hearsay-II speech understanding system," *Proc. IJCAI-77*, pp. 27-35, 1977.
[26] J. Hendler, A. Tate, and M. Drummond, "AI planning: Systems and techniques," *AI Mag.*, vol. 11, no. 2, pp. 61-77, 1990.
[27] E. Hudlicka and V. Lesser, "Meta-level control through fault detection and diagnosis," *Proc. AAAI-84*, pp. 153-161, 1984.
[28] V. Lesser and L. Erman, "A retrospective view the Hearsay-II architecture," in *Proc. IJCAI-77*, pp. 790-800, 1977.
[29] V. Lesser and D. Corkill, "The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks," *AI Mag.*, vol. 4, no. 3, pp. 15-33, 1983 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, Eds. Reading, MA: Addison-Wesley, 1988).
[30] V. Lesser, H. Nawab, M. Bhandaru, N. Carver, Z. Cvetanovic, I. Gallastegi, F. Klassner, "Integrated signal processing and signal understanding," Tech. Rep. 91-34, Dept. of Comput. Infor. Sci., Univ. of Massachusetts, 1991.
[31] D. McDermott, "Planning and acting," *Cognitive Sci.*, vol. 2, no. 2, pp. 71-109, 1978.
[32] H. Nawab and V. Lesser, "Integrated processing and understanding of signals," in *Symbolic and Knowledge-Based Signal Processing*, Alan Oppenheim and Hamid Nawab, Eds. New York: Prentice Hall, 1992.
[33] E. Rich and K. Knight, *Artificial Intelligence*. New York: McGraw-Hill, 1991.
[34] W. Swartout, Ed., "Report: DARPA Santa Cruz workshop on planning," *AI Mag.*, vol. 9, no. 2, pp. 115-131, 1988.
[35] R. Washington and B. Hayes-Roth, "Input data management in real-time AI systems," in *Proc. IJCAI-89*, pp. 250-255, 1989.
[36] D. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann, 1988.



Norman Carver received the B.A. degree in physics in 1977 from Kalamazoo College, Kalamazoo, MI, and the Ph.D. degree in computer science in 1990 from the University of Massachusetts, Amherst, MA.

He is currently a Senior Postdoctoral Research Associate in the Department of Computer and Information Science at the University of Massachusetts. His major research interest include the control of complex AI systems, reasoning under uncertainty, sensor interpretation, and distributed artificial intelligence.

Victor R. Lesser received the B.A. degree in mathematics in 1966 from Cornell University, Ithaca, NY, and the M.S. and Ph. D. degrees in computer science in 1969 and 1972, respectively, from Stanford University, Stanford, CA.

For five years he was a Research Computer Scientist at Carnegie-Mellon University, Pittsburgh, PA, where he was responsible for the system architecture of the Hearsay-II Speech Understanding System. Since 1977, he has been a Professor of Computer Science at the University of Massachusetts, Amherst. His major research focus is on the control and organization of complex AI systems.

Dr. Lesser is a Fellow of the American Association of Artificial Intelligence (AAAI), and has done extensive research in the areas of blackboard systems, distributed AI, and real-time AI. He has also made contributions in the areas of computer architecture, diagnostics, intelligent user interfaces, parallel AI, and plan recognition,