The Evolution of Blackboard Control Architectures*

Norman Carver and Victor Lesser

CMPSCI Technical Report 92-71
October 1992

Abstract

This paper examines the issues that arise in the control of blackboard systems for applications with large and complicated search spaces by analyzing the evolution of blackboard control architectures. We feel that the issues addressed here apply more generally to AI application domains involving complex multi-dimensional search, in which control knowledge is as important to successful problem solving as domain knowledge. Evolution is viewed largely from the context of the Hearsay-II (HSII) speech understanding system. The appeal of the blackboard model is that it provides great flexibility in structuring problem solving. On the other hand, many of the features that are responsible for this flexibility make effective control difficult because they complicate the process of estimating the expected value of potential actions. Among the key themes in the evolution of blackboard control is the development of mechanisms that support more sophisticated goal-directed reasoning. In the basic control mechanism of HSII, control decisions could consider only the local and immediate effects of possible actions. Thus, the value of potential actions in meeting the system goals could be evaluated in only a limited manner. The development of appropriate abstractions of the intermediate state of problem solving can be used to evaluate the non-local effect of actions relative to the overall problem-solving goals. In addition, blackboard systems went from the implicit representation of goals in HSII to explicit representation of the goals that must be satisfied in order to meet the overall goals of the system. This allowed the implementation of various styles of goal-directed reasoning (e.g, subgoaling and planning) that were not supported in the basic HSII control mechanism. Other architectural mechanisms were concerned with efficiency issues. We will examine a number of different blackboard control architectures that have evolved from the basic model of HSII: HASP/SIAP's event-based control, CRYSALIS' hierarchical control, the DVMT's goal-directed architecture, the control blackboard architecture (BB1), model-based incremental planning for the DVMT, the channelized/parameterized control loop version of the DVMT, ATOME's hybrid multistage control, CASSANDRA's distributed control, and the RESUN interpretation framework.

*This is a revised and expanded version of a paper that will appear in *Expert Systems with Applications*, special issue on The Blackboard Paradigm and Its Applications. The major differences between the two versions include: Section 6.3 was significantly rewritten; Sections 6.6, 6.7, and 6.8 were added; and there are several additional figures in Section 6.

Contents

1 Introduction **3**

2 The Blackboard Architecture **4**
2.1 The Blackboard Model of Problem Solving . . . 5
2.2 Agenda-Based Control . . . 7

3 Control in Blackboard Systems **12**
3.1 The Control Problem . . . 12
3.2 Goal-Directed Control . . . 14
3.3 The Termination Problem . . . 15
3.4 Blackboard Problem-Solving Strategies . . . 16

4 Themes in the Evolution of Blackboard Control **17**

5 Additional Control Mechanisms in Hearsay-II **19**
5.1 Predict and Verify . . . 20
5.2 Large-Grained KSs . . . 21
5.3 STOP: Termination . . . 21
5.4 Generator and Policy KSs . . . 21
5.5 WORD-SEQ . . . 22
5.6 KSI Clustering . . . 23
5.7 Triggered versus Executable KSs . . . 23

6 Blackboard Control Architectures **24**
6.1 HASP/SIAP: Event-Based Control . . . 25
6.2 CRYSALIS: Hierarchical Control . . . 28
6.3 The Goal-Directed Blackboard Architecture . . . 31
6.4 BB1: The Control Blackboard Architecture . . . 36
6.5 Model-Based Incremental Planning . . . 40
6.6 The Channelized, Parameterized Blackboard Architecture . . . 41
6.7 ATOME: Hybrid Multistage Control . . . 45
6.8 CASSANDRA: Distributed Blackboard Control . . . 46
6.9 RESUN: Planning to Resolve Sources of Uncertainty . . . 48

7 Emerging Directions **52**

8 Conclusion **54**

List of Figures

1 Hearsay-II's agenda-based blackboard control architecture. 8
2 The basic control loop for Hearsay-II's agenda-based control architecture. 9
3 An example showing the possibility of interrelationships among possible actions. . . 13
4 The HASP/SIAP blackboard architecture. 26
5 The basic control loop for HASP. 27
6 The CRYSALIS blackboard architecture. 29
7 The basic control loop for CRYSALIS. 30
8 The goal-directed blackboard architecture. 32
9 The basic control loop for the goal-directed blackboard architecture. 33
10 The BB1 blackboard architecture. 37
11 The channelized, parameterized blackboard architecture. 42
12 The meta-controller of the channelized, parameterized blackboard architecture. . . . 43
13 The parameterized control loop for the channelized, parameterized blackboard architecture. 44
14 The basic RESUN control planning loop. 49

1 Introduction

The blackboard model of problem solving arose from the Hearsay speech understanding systems [Erman80, Lesser75, Lesser77b, Nii86a]. Some of the key ideas of the model were developed in a simple form in *Hearsay-I* [Reddy73a, Reddy73b]. These ideas were then extended into what we now think of as the standard blackboard architecture in *Hearsay-II* (HSII). The blackboard model has proven to be popular for AI problems and in the years since HSII a variety of blackboard-based systems have been developed. For example, blackboard architectures have been used for interpretation problems such as speech understanding, signal understanding and image understanding [Carver91a, Durfee86, Lakin88, Lesser83, Maksym83, Nii78, Nii82, Srihari87, Williams88]; planning and scheduling [Hayes-Roth79, Pearson88, Smith85]; arrangement-assembly (structure identification) [Hayes-Roth86a, Hayes-Roth86b]; and machine translation [Nirenburg89].

The blackboard architecture was developed to deal with the difficult characteristics of the speech understanding problem: a very large search space; errorful or incomplete input data; and imprecise and/or incomplete problem-solving knowledge. These characteristics require a problem-solving model that supports the incremental development of solutions, can apply diverse types of knowledge, and that can adapt its strategies to the particular problem situation. The blackboard model has been so popular for complex problems because it supports incremental problem solving and because it provides a great deal of flexibility in structuring the problem-solving process. For example, blackboards allow problem solving (search) in which the system dynamically switches among different levels of abstraction (to work at) and in which multiple (competing, cooperating, or independent) lines of reasoning are concurrently pursued.

While the blackboard model supports the kind of problem solving that is appropriate for complex problems, there are still many issues that must be addressed in order to successfully solve such problems. In particular, effective control is critical in blackboard applications that involve significant uncertainty in the data and problem-solving knowledge. However, it can be difficult to effectively utilize the features of the model that are the source of its flexibility because they complicate the process of selecting actions with the maximum expected value. As a result, control was a major issue in the formulation of the HSII architecture and control continues to be an active area of research in the field of blackboard systems.

In this paper we will examine the evolution of blackboard control since HSII. Viewing blackboard architectures from the perspective of HSII is useful because HSII is the best known blackboard system and its basic mechanisms can serve as a common point of reference for the comparison of other systems. In addition, while speech understanding was the first problem to use a blackboard system, it is still one of the more complex problems to be tackled with this framework. As a result, many of the control issues that were of concern in later blackboard applications were addressed in some way in the HSII implementation. However, several of the mechanisms used in HSII were ad hoc and failed to fully address the issues. In analyzing later blackboard systems, we will show how they generalized elements of the HSII implementation by making particular mechanisms and strategies an explicit and integral part of their control architectures. We will also show that in other blackboard systems, mechanisms and strategies were eliminated or simplified when they were not necessary for effective problem solving in an application domain. Typically this involved trading flexibility for efficiency.

One of the key themes in the evolution of blackboard control is the development of mechanisms that support more sophisticated goal-directed control strategies. Goal-directed control provides systems with an understanding of the global and long-term effects of actions as well as the local and immediate effects. HSII had a limited ability to do goal-directed reasoning through its use of abstract models of the intermediate state of problem solving to estimate the non-local effects of actions. However, this mechanism was domain-specific and did not explicitly represent the relationships between potential actions and the problem-solving goals. Many later blackboard control architectures permit the explicit representation of detailed goals and their relationships to the overall system goals. This has allowed them to implement various styles of goal-directed reasoning (e.g, subgoaling and planning) that were not supported in the basic HSII control mechanism.

We will also see later architectures that address issues that were not of concern in HSII because of its particular task. For example, HSII dealt with the interpretation of data from a single phrase and had soft real-time constraints. In other tasks, solutions may involve an indeterminate number of (mainly independent) components (e.g., vehicles in a vehicle monitoring system); there may be multiple sensors and sensors that continuously generate large amounts of data; or the system may need to deal with hard real-time deadlines. These factors complicate the control problem. For example, having multiple solution components complicates the termination problem (the problem of deciding whether the system has done enough work to find the best answer); continuous passive sensors necessitate the use of mechanisms for keeping the blackboard and agenda from being overloaded; and real-time deadlines require predictable control cycles, the ability to estimate the duration of potential actions, and the capability of dynamically changing the criteria for acceptable solutions.

Section 2 introduces the basic blackboard architecture in terms of the concepts behind the blackboard model of problem solving and the agenda-based control architecture of HSII. Section 3 examines the general issues in the control of blackboard systems to show why blackboard control can be difficult. This is followed in Section 4 with a discussion of what we feel are the major themes in the evolution of blackboard control. These themes are then examined further by reviewing various specialized control mechanisms that were developed for HSII (Section 5) and by reviewing a number of major blackboard control architectures (Section 6). The paper concludes with a section on emerging directions in blackboard research and a summary of the key issues in blackboard control.

2 The Blackboard Architecture

This section introduces the basic blackboard problem solving architecture as it was developed in Hearsay-II. Because our principal focus is control, the other components of the blackboard architecture will be discussed only to the extent necessary to understand control issues. More complete discussions of the blackboard model can be found in [Carver92, Engelmore88, Erman80, Nii86a, Nii86b]. The first subsection of this section introduces the concepts that underlie the blackboard model, while the following subsection examines the agenda-based control mechanism used in HSII.

2.1 The Blackboard Model of Problem Solving

In the basic model that came out of Hearsay-II, a blackboard system is composed of three main components: the *blackboard*, a set of *knowledge sources* (KSs), and a control mechanism. The blackboard is a global database (shared by all the KSs) that contains the data and *hypotheses* (potential partial solutions). The blackboard is structured as a (loose) hierarchy of *levels*, particular classes of hypotheses are associated with each level, and hypotheses are typically linked to hypotheses on other levels.[1] The levels are themselves structured in terms of a set of *dimensions*. This makes it possible to provide efficient *associative retrieval* of hypotheses based on the notion of an "area" of the blackboard. The set of knowledge sources embody the problem-solving knowledge of the system. KSs examine the state of the blackboard and create new hypotheses or modify existing hypotheses when appropriate. Ideally, KSs should be *independent*: their execution should not explicitly depend on the execution of other KSs and any communication of information between the KSs occurs only via the creation and modification of hypotheses on the blackboard.

Among the key ideas behind the blackboard model are that problem solving should be both *incremental* and *opportunistic*. Incremental problem solving means that complete solutions are *constructed* piece by piece and at different levels of abstraction. The standard strategy for blackboard problem solving is often referred to as *incremental hypothesize and test* (or *evidence aggregation*). This involves first hypothesizing a (partial) solution based on incomplete data and then attempting to verify additional data to confirm that hypothesis. Incremental hypothesize and test differs from *generate and test* [Rich91] in that hypotheses need not be complete solutions. Thus, the test stage can refine a hypothesis as well as resolve uncertainty about the hypothesis.

To better understand this process, consider that most blackboard problems can be viewed as constraint satisfaction problems in which the data constrains the set of acceptable solutions. Blackboard-based problem solving then involves a search process in which constraints are incrementally identified and solutions constructed/modified to be consistent with those constraints. Having multiple blackboard levels facilitates incremental problem solving. Hypotheses at successively higher levels of the blackboard represent the application of a more complete set of constraints in terms of both the amount of data and the characteristics of the data that are necessary for acceptable solutions. For example, in HSII, hypotheses at the word level represented the application

[1] What we are describing here is the basic blackboard model as it was developed in HSII. Considerable variation can be found in later "blackboard systems" and terminology has not been standardized for many aspects of the blackboard model. For example, the types of relations between hypotheses on different levels depend on the application. The relations for interpretation problems (like HSII) are supports/explains and higher-level hypotheses represent more abstract explanations of the data [Carver91a]. The relations for arrangement-assembly problems are typically part-of/includes [Hayes-Roth87]. In addition, there may or may not be explicit relations among hypotheses on the same level. Furthermore, the (overall) blackboard database may itself have a structure. It may consist of multiple independent blackboards or multiple hierarchically related blackboards (sometimes called *panels*) and there may be multiple instances of particular (sub) blackboards. Finally, blackboards and KSs may be used for more than just domain problem solving. For example, they may be used for control (see Sections 6.3 and 6.4). In these cases, we must speak more generally of blackboard "objects" rather than hypotheses.

of constraints based on data from a relatively brief portion of the complete utterance using incomplete models of speech (that describe the acceptable appearance of each potential word at the data level). At the phrase level (a higher level), hypotheses represent the application of constraints based on data from longer portions of the utterance using additional models of speech (involving knowledge about grammatically valid sequences of words and co-articulation phenomena).

Opportunistic problem solving means that the system chooses the actions to take next that it determines will allow it to make the best progress toward meeting its goals given the current situation—i.e., given the available data and the intermediate state of problem solving (as represented by the current set of competing, cooperating, and independent hypotheses on the blackboard). By contrast, many AI architectures insist on the use of strategies for sequencing actions that can depend on the emerging problem-solving situation in only a very limited way. For example, rule-based systems typically force all reasoning to be either forward or backward chaining. Blackboard systems are able to select the approach that is most appropriate for the situation. Likewise, blackboard systems can use an "island driving" strategy in which hypotheses that are judged to be likely ("islands of certainty") are used as the basis for developing complete solutions. This allows the system to work where it can make the best progress instead of being limited to a fixed, predetermined strategy that may not be best for every problem situation. In speech understanding, for example, well-sensed words can be used to constrain the search for poorly sensed words. This can be more effective than a fixed strategy such as interpreting the phrase strictly from beginning to end [Lesser77a].

Opportunistic problem solving is facilitated by another key aspect of the HSII model: the ability to concurrently pursue *multiple lines of reasoning* (alternative solution paths). This is possible because HSII had an *integrated* representation of hypotheses: the blackboard is the sole, global representation of all the developing solution hypotheses and there are no structural barriers that separate hypotheses arising from different lines of reasoning. In addition, the exact relationships among the hypotheses can be determined because each hypothesis is kept linked to the higher-level and lower-level hypotheses and data that "support" and "explain" it.[2] For instance, this hierarchical structure can show when two hypotheses are competing alternatives and can show exactly why this is so (in terms of shared support).[3]

An integrated, global database is also appropriate for KSs to be independent because it allows the KSs to readily recognize when they should be invoked without having to directly call each other. Partitioning the problem-solving knowledge of the system into a set of independent KSs

[2]Note that this type of "integrated" approach to representing hypotheses can be contrasted with the "possible worlds" approaches of Truth Maintenance Systems which make it difficult to understand in detail the relationships between alternative hypotheses [Carver90b].

[3]In some applications it is possible to determine when hypotheses are competing alternatives without examining the supporting substructure. For example, in the HSII application, there can be only one correct interpretation at every abstraction level for any given time portion of the utterance. Thus, any two hypotheses at the same level that overlap in the time they cover, are alternatives. However, recognizing alternatives is not always so simple and it may also be necessary to understand the exact structural (evidential) relationships among hypotheses to decide how to resolve conflicts. Finally, it is worth noting that HSII's integrated representation had certain limitations that prevented the supporting substructure from being viewed from alternative perspectives [Carver90a, Lesser77b, Nii86b]. This problem was corrected by Hearsay-III's context mechanism [Erman81] and the RESUN framework's (Section 6.9) extensions representation.

produces a highly modular representation of problem-solving knowledge. This modularity facilitates experimentation with alternative problem-solving methods and strategies—i.e., alternative KSs and alternative control mechanisms or control knowledge. Such experimentation is often crucial for the successful development of complex knowledge-based systems. In addition, because KSs communicate only with each other via a predefined and uniform representation (the hypotheses on the blackboard), they are free to use any type of internal problem solving architecture. This means that the blackboard architecture is appropriate for integrating diverse types of problem-solving knowledge or methods implemented with a diverse set of languages.

Another important aspect of the HSII model that permits KS independence is that KSs are *self-activating*. In other words, each KS has a precondition-action format in which the precondition determines when the action is applicable based only on the current state of the blackboard. This gives blackboard systems like HSII the strong *data-directed*[4] characteristic that is necessary for supporting opportunistic problem solving. Of course, as we begin to talk about activating KSs, we move into the area of control. In the next subsection, we will discuss the agenda-based control mechanism of HSII. Section 6 will examine a variety of blackboard control architectures, including several in which KSs are not self-activating and not totally independent.

Because blackboard systems incrementally develop hypotheses representing alternative partial solutions and are able to do this concurrently and at different levels of abstraction, it is often important to be able to judge the credibility of each hypothesis. As a result, while not strictly a necessary part of the blackboard model, in most blackboard applications hypotheses are assigned *credibility ratings* as they are created and modified.[5] These ratings are derived from a consideration of how encompassing and how critical the set of verified constraints are for each hypothesis and how well the hypotheses meet any soft constraints. Credibility ratings are used in making control decisions and in judging the reliability of the final answer. Hypothesis rating functions may be embedded in the KSs or they may be general routines that are called by the KSs or are KSs themselves. Since ratings combine the overall effect of multiple KSs on a hypothesis and since the effect of changes must be propagated throughout the hypothesis sub/super-structure, it makes more sense to use rating routines that are not embedded in the KSs. If the rating routines are KSs, the system can reason about when to re-rate hypotheses so that rating need not be done following every modification (HSII used a KS, *rpol*, to rate hypotheses).

[4] When we use the term data-directed, we mean forward reasoning search (forward chaining) [Rich91] in which the system searches from the initial state to find a goal state using the set of possible actions it can take to change the problem state. In goal-directed or backward reasoning search, the system reasons from its goals (using problem reduction) to identify actions that can satisfy those goals. Most blackboard systems should more properly be referred to as *event-directed* since the set of actions they consider are triggered by a set of defined events (see Section 2.2). Whether they are data-directed or goal-directed then depends on the nature of the events.

[5] Hypothesis rating schemes have usually been ad hoc. A better approach would be to have the ratings correspond to probabilities (of the hypotheses being a part of the correct solution). This approach is taken in the RESUN framework (Section 6.9) where the relationships between the hypotheses are treated as evidential.

2.2 Agenda-Based Control

Because KSs in the HSII blackboard model are both independent and self-activating, in a sense there is no need for any (additional) control mechanism: when a KS finds that it is applicable it could execute. Despite the appeal of a model without any control component, this approach has two serious problems.[6] First, because most computers have had a single processor the execution of KSs must be sequentialized. This means that KSs cannot really execute as soon as they become applicable. It also means that the checking of KS preconditions must compete with the execution of KS actions for processor resources. The second problem with the "no control" approach is that blackboards are typically applied to combinatorially explosive problems. Such problems become intractable if the system attempts to execute all the KS actions (that are applicable prior to finding a solution). When "unpromising" actions are executed, they "distract" the system by triggering further actions that are also not useful and that compete with the useful actions for the limited computation resources. This can greatly delay the construction of the final solution.

Because of these problems, a blackboard system must typically have some control mechanism—i.e., some mechanism that applies knowledge to focus the search process. HSII used an *agenda-based* control mechanism: all possible actions are placed onto the agenda and on each cycle the actions are rated and the most highly rated action is chosen for execution. The major elements of the agenda-based architecture (of Hearsay-II) are shown in Figure 1, and Figure 2 summarizes the main control loop of such a blackboard system. The elements of the HSII architecture concerned with control are: the blackboard monitor, the agenda, the scheduler, and the focus-of-control database.

The first step in agenda-based control is to identify the actions that could be taken by the system. New actions become possible when changes are made to the blackboard, so KS preconditions must be checked following the execution of each KS (action). However, since it could get very expensive to check the preconditions of every KS after each set of blackboard changes, HSII introduced a mechanism for limiting this checking that has been used in most later agenda-based blackboard systems. All changes to the blackboard are categorized in terms of a set of *blackboard event types* and each KS provides a list of the blackboard event types in which it is "interested." Every time a KS action is executed, the changes to the blackboard are described in terms of the blackboard event types. These event descriptions are passed to the *blackboard monitor*, which identifies the KSs that should be *triggered*—i.e., those KSs whose preconditions should be checked given the types of blackboard changes.

[6] When we talk about the problems associated with having "no control" we are assuming that KS preconditions identify all the situations when the KS is applicable. In other words, we are talking about uncontrolled search. It would, of course, be possible to encode control knowledge directly into the KS preconditions so that they would identify only those situations in which it was useful (plausible) to apply the KS. In fact HSII KSs did some heuristic evaluation—see the discussion of "generator KSs" in Section 5.4. However, heuristic evaluations were not based on factors like hypothesis credibility ratings since these ratings might change, but there was no way to then re-trigger the KSs. The chief drawback of encoding control knowledge directly in the KSs is that it makes the overall system control strategies difficult to understand and modify (this is different from using explicit *control KSs* as in BB1). This issue has been much discussed in the context of rule-based systems (precondition-action KSs are analogous to rules) [Davis80, Rich91]. In addition, "compiling" control strategies into the KSs could complicate the use of "non-local" information to make control decisions (see Section 3).

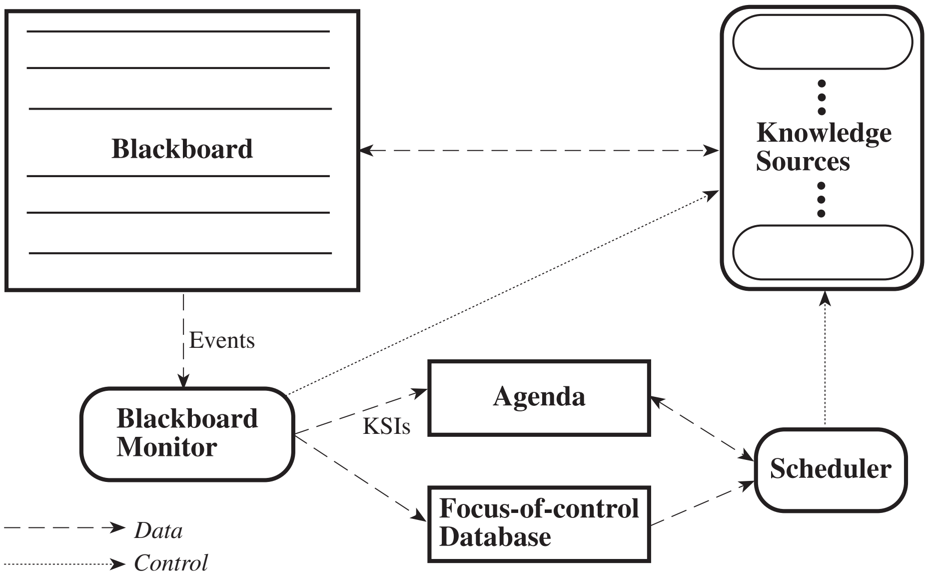

Figure 1: Hearsay-II's agenda-based blackboard control architecture.
Changes made to the blackboard by KSs are described to the *blackboard monitor* as a set of *events* (blackboard event type instances). The blackboard monitor identifies which KSs should be *triggered* by new events and invokes the precondition components of the triggered KSs. Successful KS preconditions return stimulus and response frame information that the blackboard monitor uses to create KSIs (representing activated KSs) that it places onto the *agenda*. The blackboard monitor also updates the *focus-of-control database* based on the new events. The *scheduler* rates the KSIs on the agenda, selects (and removes) the highest rated one, and invokes the appropriate KS action component.

The blackboard monitor invokes the precondition component of each triggered KS to see whether the KS is able to be executed given the current state of the blackboard.[7] Besides signalling success or failure, the precondition returns any context information that is necessary to execute the action portion of the KS, such as hypothesis bindings. HSII preconditions also returned information that was used by the scheduler when rating the possible action: the *stimulus frame* and the *response frame*. The stimulus frame represented the key hypotheses that satisfied the precondition of the KS—typically those hypotheses that would be used as input to the action portion of the KS. The response frame was a stylized representation of the approximate blackboard changes that the KS was likely to produce.[8]

If a KS precondition is found to be satisfied, the KS is said to be *activated*. Instead of immediately executing (the action component) of the activated KS, the blackboard monitor creates

[7] While precondition-action KSs are analogous to the rules of a rule-based (production) system, KS preconditions are typically more complex. In rule-based systems, rule preconditions (LHSs) typically use uniform frameworks that allow the specification of relatively simple patterns that can be quickly matched against the database. In many blackboard systems, KS preconditions can be arbitrary pieces of code that search the blackboard and do significant reasoning. For example, blackboard shells like GBB [Gallagher88] provide blackboard access functions that allow the specification of complex retrieval patterns. Since KSs are much larger-grained pieces of knowledge than rules, the greater complexity of KS preconditions is appropriate.

[8] The response frame lists the effects of a KS in terms of the types of changes that may be made to the blackboard and the area of the blackboard that will be affected. The response frame is an *approximate* representation of the actions of the KS because it is impossible, in general, to predict the exact effects of a KS without executing it. In HSII "a non-procedural abstraction of each KS is used to estimate the RF directly from the SF." [F. Hayes-Roth77]

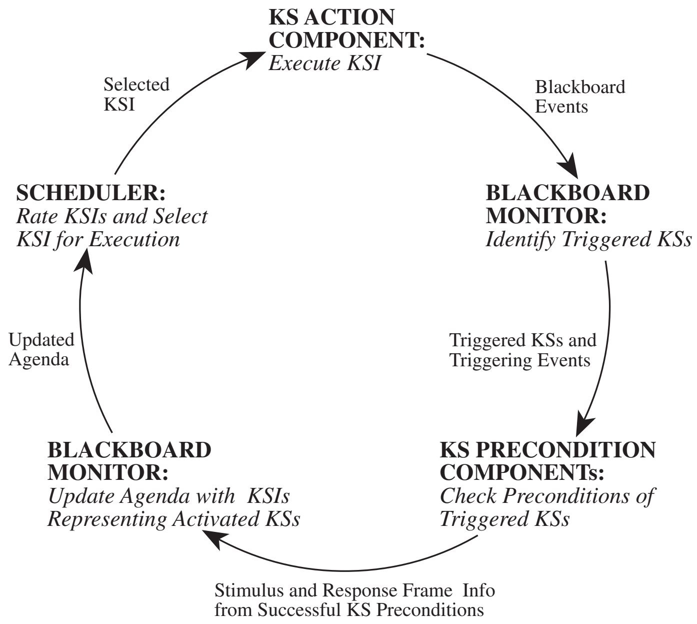

Figure 2: The basic control loop for Hearsay-II's agenda-based control architecture.

a *knowledge source instantiation* (KSI)[9] and places it onto the agenda. The KSI includes all the information needed by the action component of the KS and any other information for control (e.g., stimulus and response frame information). Each KSI represents an action that could be taken by the system, and at any given time, the set of KSIs on the agenda represents all the actions that the system could possibly take next.[10]

After the preconditions of all the KSs triggered by a set of blackboard changes are processed and any activated KSs have had their new KSIs placed on the agenda, the scheduler must choose the KSI (action) to be executed next. It starts by rating each of the KSIs on the agenda. It then selects the highest rated KSI, removes it from the agenda, and executes it (i.e., it executes the action part of the associated KS using the context specified in the KSI). Execution of a KSI may result in the creation or modification of hypotheses on the blackboard, bringing the system back to the beginning of the control loop. The blackboard control cycle repeats until acceptable answers have been found or else there are no KSIs on the agenda (the problem of determining whether acceptable answers have been found is often referred to as the *termination problem* and is discussed in Section 3.3).

An agenda-based control architecture as described here is inherently opportunistic because KSs are activated in a data-directed manner and because all possible actions are considered during each control cycle. This allows for rapid refocusing (at each control cycle) between different lines of reasoning, different levels of abstraction, and so on. However, effective blackboard control typically requires the integration of both data-directed and goal-directed control factors (see Section 3). In

[9] A knowledge source instantiation is also sometimes referred to as a *knowledge source activation record* (KSAR) [Hayes-Roth85a] or just a *knowledge source activation* (KSA) [Gallagher88].

[10] Note that multiple KSIs based on the same KS—but with different contexts—may be triggered by a given set of events and may reside on the agenda at any one time.

the basic agenda-based blackboard architecture, it is the scheduler's rating function that embodies the control (strategy) knowledge of the system. HSII rated KSIs independently, without taking into account the existence of other KSIs on the agenda. Ratings were computed as a linear function of several (fixed) factors using static weights. The factors were chosen to provide some goal-directed reasoning capabilities by attempting to "estimate the usefulness of the action in fulfilling the overall system goal" [Lesser77b]. This was possible because the scheduling function had available to it the stimulus and response frame for each KSI as well as information about the overall "state of problem solving" that was contained in the focus-of-control database.

The *focus-of-control database* identified the best hypotheses in each "area" of the blackboard and how much time had elapsed since these hypotheses were generated. It was updated by the blackboard handler as the blackboard was modified. Focus-of-control information was useful in determining whether it was likely to be worthwhile to work in a particular area or pursue a particular hypothesis further. The stimulus frame allowed the rating function to consider the "credibility" of the data or hypotheses that triggered the creation of the KSI—i.e., the data-directed control factors. The response frame and the focus-of-control database allowed the rating function to consider the "desirability" of creating the hypotheses which the KSI was expected to create—i.e., the goal-directed control factors.

In addition to the basic steps in the agenda-based control cycle discussed above, there are two other issues that merit discussion here. First is the overhead of rating possible actions—in particular, the potential overhead from continually re-rating the same KSIs. While our basic description of the agenda control loop implied that all KSIs were rated on each cycle, in fact, most agenda-based blackboard systems employ some scheme for limiting the amount of re-rating of actions. For example, in HSII, the stimulus frame of each KSI identified "key hypotheses" such that changes to these hypotheses would require re-rating the KSI (re-rating was triggered through the tag mechanism mentioned below). Likewise, the response frame was linked to the focus-of-control database so that changes in areas that might be affected by the KSI would cause re-rating of the KSI. Otherwise, KSIs were not re-rated. Difficulties can arise, however, because in many applications it may not be so straightforward to determine when KSIs need to be re-rated. Actions can affect the usefulness of possible actions that are not even concerned with the same hypotheses (e.g., through goal relations [Lesser89]). The cost of re-rating KSIs is also an issue in systems like BB1 (Section 6.4) where the rating functions being used by the scheduler can be dynamically changed.

Another issue arises because there can be a delay between when a KS's precondition is checked and when the resulting KSI is executed. As a result, the situation on the blackboard may have changed and the system must verify that the KSI is still applicable before it can be executed. Blackboard systems have handled this problem in several ways from simply re-evaluating the precondition component of the KS (actually a modified version of the precondition procedure) after a KSI is scheduled for execution to methods that allow the system to recognize when KSIs are no longer applicable so that they can be removed from the agenda as soon as possible. HSII used *tags* [Lesser75] to recognize when KSIs should be removed from the agenda. Blackboard data structures in the stimulus frame of a KSI were tagged and the KSI was "informed" if any changes were made to this data. The precondition was then re-evaluated in light of the changes to see whether the KSI was still applicable and whether changes had to be made in it to account for the new context. Another technique for recognizing when KSIs should be removed from the agenda is to use *obviation conditions* [Hayes-Roth85a] defined with each KS. The obviation condition of a KS specifies the conditions under which KSIs (based on the KS) are no longer applicable. While checking the conditions can involve significant computational cost, this may be more than offset by removing KSIs if the rating process is expensive or if KSIs are repeatedly re-rated. Also, obviation conditions can be simpler than KS preconditions because only some of the objects in the stimulus frame may be able to change.

3 Control in Blackboard Systems

This section discusses some general issues in blackboard control, including the reasons why blackboard control can be difficult, goal-directed control, the termination problem, and basic problem-solving strategies used in blackboard systems.

3.1 The Control Problem

In an AI system, the control problem is the problem of selecting the "best" action to execute next. The value of an action is ultimately determined by how much it contributes toward progress in problem solving relative to the computational costs of the action—where progress is judged by how much the action reduces the remaining amount of computational effort required to meet the system goals (and terminate problem solving) [Whitehair92]. This definition is very general because it not only admits actions that generate (some part of) the correct solution, but also actions that make progress by producing information that provides a better understanding of what actions to take—e.g., actions that discount potential solutions and thereby eliminate incorrect solution paths (prune the search space).

Given this definition of the control problem, the task of the control component in a blackboard system is to determine which of the KSIs currently on the agenda has the maximum expected value. What makes blackboard control difficult is that can be highly problematic to determine the expected value of KSIs because there may be complex interrelationships among the KSIs. These relationships arise from many of the features that give the blackboard model its power: opportunistic incremental construction of solutions at different levels of abstraction, the ability to concurrently pursue multiple lines of reasoning, the possibility of multiple derivation paths for solutions, and the existence of multiple methods for accomplishing goals.

Because possible actions are often not independent, the control component in a blackboard system must not only consider the local and immediate effects of each KSI, but must also consider its global and long-term effects. However, the relationships among possible actions are not readily observable in a blackboard system. For instance, the control component would have to recognize the existence of multiple lines of reasoning by analyzing the structural relations that

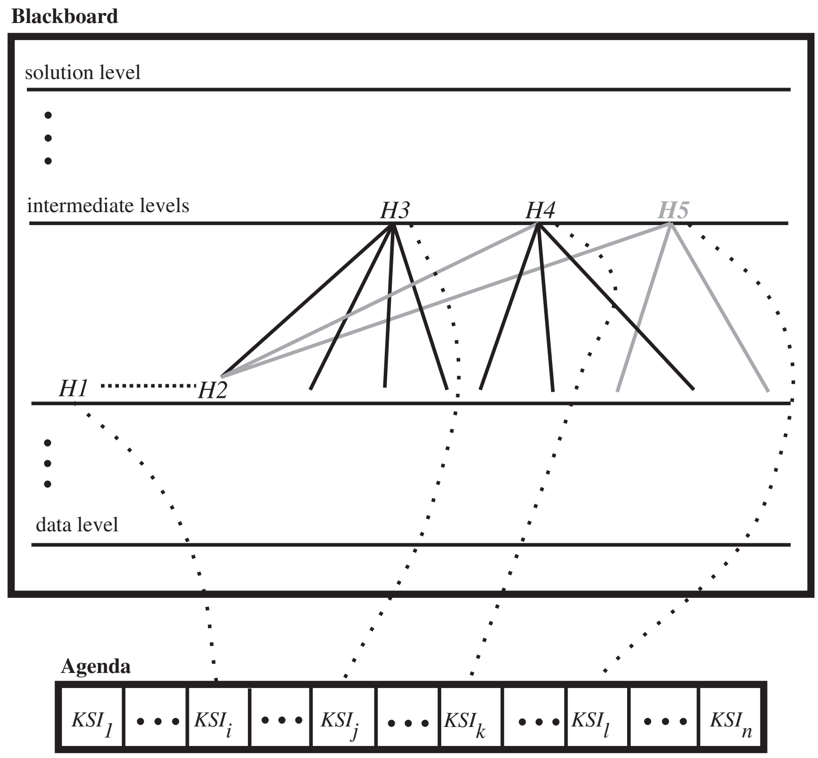

Figure 3: An example showing the possibility of interrelationships among possible actions.
Here, some of the hypotheses on two of the intermediate abstraction levels of the blackboard are visible. The situation is as follows: hypotheses H_1 and H_2 are on the lower of the two intermediate levels and are currently known to be competing alternatives. Hypotheses H_3, H_4, and H_5 are on the upper level. H_3 and H_4 currently exist, while H_5 has not yet been created. H_3, H_4, and H_5 are competing, alternative "explanations" for H_2. However, this relationship is not currently known because links have not been established between H_2 and H_4, and because H_5 has not yet been created. In addition, H_3, H_4, and H_5 are also competing alternatives to H_1 because they use H_2 as support and because H_2 and H_1 are competing alternatives. Among the possible actions on the agenda are KSIs that are associated with these hypotheses—e.g., they produce additional support for an existing hypothesis or create a new hypothesis, etc. For example, KSI_i is associated with H_1 since it produces additional support for H_1 from lower level hypotheses.

determine the competing and cooperating relations among hypotheses with which the actions are associated [Lesser89]. Furthermore, the ultimate value of an action can depend on when it is executed and on the actions that follow it. As a result, the effectiveness of applying costly procedures to estimate value is limited since the value of a possible action can change following the execution of any other action.

These issues can be illustrated by the example in Figure 3. Actions that are (directly) associated with hypothesis H_1 interact with actions associated with hypotheses H_2, H_3, H_4, or H_5 here since changes in the credibility of H_1 may affect the credibility of all these other hypotheses (via the evidential relations among the hypotheses). Thus, while KSI_i and KSI_j might have similarly significant (local) impacts on the credibility of H_1 and H_3, respectively, one or the other could have greater (global) value depending on its overall effect on the credibility of H_1, H_2, H_3, H_4, and H_5. However, it can be quite difficult to make such a determination; for instance, since H_5 has not yet been created. In addition, the value of an action must be considered relative to the system goals and the eventual need to execute alternative actions. While KSI_j might have a greater immediate effect on the affected hypotheses than KSI_i, its ultimate value may not be greater if KSI_i must still be executed because H_1 is not sufficiently proved/disproved by executing KSI_j.

Another way of talking about these blackboard control issues is to distinguish between *solution uncertainty* and *control uncertainty* [Lesser91a]. Solution uncertainty means uncertainty whether a hypothesis is (part of) the correct solution. Hypothesis credibility ratings (and the reliability of these ratings) are concerned with solution uncertainty. Control uncertainty means uncertainty about the expected value of possible actions. Control uncertainty involves solution uncertainty (whether the hypotheses associated with any possible action will be part of the final solution), but it also involves uncertainty about the exact results that will be produced by possible actions and how these results will affect the effort required to meet the system goals (by changing the search space).

3.2 Goal-Directed Control

Every system has some overall goals that it is trying to satisfy. When we speak in general terms about goal-directed control, we mean control that considers the role and the ultimate value of actions in satisfying these system goals. Effective control typically requires the integration of both goal-directed and data-directed control factors. Goal-directed factors tell a system what it would most like to do in order to solve the problem. Data-directed factors tell a system what it is best able to do given the available data. Without goal-directed control, effort may be wasted working on data and hypotheses that are not important for meeting system goals. Without data-directed control, effort may be wasted pursuing goals that cannot be easily satisfied or pursuing ineffective methods for satisfying goals.

Goal-directed control also refers to specific styles of control reasoning that involve *problem reduction* [Rich91]—e.g., subgoaling, precondition-action backchaining, and planning. Subgoaling involves the reduction of abstract, high-level goals into detailed, low-level goals that can be directly solved. Subgoaling can be used to focus low-level processing so that hypotheses will be created that can extend existing high-level hypotheses. Precondition-action backchaining involves identifying actions that can enable other actions that are necessary to satisfy a goal. Precondition-action backchaining is useful when a certain action (KS) is necessary to meet a goal, but the action is not yet executable because its precondition is not (fully) satisfied. The key notion behind planning is the identification of an entire *sequence* of actions needed to satisfy some goal (though planning techniques often involve subgoaling and precondition-action backchaining). Planning keeps a system focused on long-term goals, allows sets of actions to be scheduled or eliminated as appropriate, and may be necessary when there are destructive interactions among possible actions.

While HSII's scheduler rating function gave it some goal-directed reasoning capabilities, they were limited because HSII did not have an explicit representation of its goals, did not understand in detail the relationships between possible actions and goals, and could not do many types of goal-directed reasoning. For example, coordinating sequences of actions to accomplish goals can be hard to do with a conventional agenda-based approach like HSII because the eventual importance of intermediate results (and the actions that will create them) may not be able to be correctly judged without a more detailed understanding of system goals. Furthermore, there is a limit to the sophistication of the control strategies that can be encoded using the HSII approach since the rating function used a static set of weights and was a linear function of a small number of fixed factors. In Section 5 we will discuss several additional mechanisms that extended the goal-directed reasoning abilities of the HSII implementation, and we will later consider other approaches that address the limitations of this basic architecture.

3.3 The Termination Problem

In blackboard problem solving, the overall system goals are often referred to as the *termination criteria* (the criteria that must be met for problem solving to terminate) and determining whether they have been met is referred to as the *termination problem.*[11] Termination is an issue in many blackboard applications because, when viewed as constraint satisfaction problems, the applications are *underconstrained*—i.e., there can be multiple potential solutions that are consistent with the constraints. Often this is a result of uncertainty in the data and/or the problem-solving knowledge for these applications. Consider, for example, a sensor interpretation task as part of an aircraft monitoring application. While each aircraft type might have an ideal spectral signature, a range of spectral data would have to be considered as consistent with an aircraft type to account for propagation distortions, sensor limitations, and so on. Furthermore, sensor interpretation is based on abductive inference and so retains the inherent uncertainty of abduction [Carver90a]. Blackboard problems can also be underconstrained because the size of the problems prevents systems from considering every constraint—e.g., examining all the available data.

When blackboard applications are underconstrained, blackboard problem solving must consider "how well" hypotheses meet the current set of constraints and how reliable (discriminatory) those constraints are. Finding *a* solution that meets the current constraints does not necessarily constitute solution of the overall problem since most heuristic search strategies are not guaranteed to find the best answer first.[12] In other words, the construction of an acceptable solution hypothesis (e.g., an answer-level hypothesis that is "sufficiently complete" and has a "reasonably high" credibility rating) does not exclude the possibility that a better alternative has simply not yet been produced. For applications where "the best" solution must be found, the termination criteria may require systems to pursue alternative solution paths simply to gather evidence that a potential solution is the best. Thus, when termination is a key issue, a system needs to be able to determine whether it would be worthwhile to continue to look for alternative solutions. This requires a more global

[11] Obviously, applications like continuous sensor interpretation involve a different notion of termination. In such problems, the termination problem may involve determining when to quit working on interpreting the data from particular periods of time or when to quit trying to resolve uncertainty in potential interpretations that are no longer of interest.

[12] Heuristic search strategies that are able to guarantee that the best answer is produced first are termed *admissible*—see, for example, [Woods77].

view of the state of problem solving than is afforded by individual actions and was (part of) the reason for HSII's focus-of-control database.

There are blackboard applications in which termination is not a significant issue because it is relatively easy to determine whether the problem-solving goals have been met. For example, in some arrangement-assembly problems the goal of the system is to construct *a* (partial) solution that embodies the constraints represented by the data. These systems typically are able to represent and accept a significant amount of uncertainty—e.g., they can represent *sets* of solutions that meet the constraints. In such problems, the system simply runs until there are no more constraints that can be applied. Of course, complex problem-solving activity may still be required since the order in which constraints are examined can be critical to making the construction of a consistent solution (set) computationally tractable.

3.4 Blackboard Problem-Solving Strategies

If blackboard problem solving is considered as a constraint satisfaction process, then one way of viewing the opportunistic problem-solving strategies used in blackboard systems is that they are chiefly concerned with how to best accumulate the constraints necessary for meeting the termination criteria. As a result, blackboard control is often more concerned with whether hypotheses (and thus KSIs) are competing, cooperating, or independent than whether problem solving should be data or model-driven. There are four main high-level problem-solving strategies that are used in the blackboard model of problem solving:

1. Search depth-first (be directed in pursuing a particular solution path) where there are statistically significant reasons for preferring one solution path. Typically this means when there is a partial solution with high credibility ("island driving") or where there are not competing alternatives with similar ratings. The advantage of directed search is that the creation of higher-level plausible solutions, representing more encompassing constraints, can be used to prune the search space by limiting the set of hypotheses that must be considered to develop a complete solution (through what is effectively a form of constraint propagation). The danger of this approach is that if it is applied too soon, using hypotheses or data whose ratings are unreliable, it can take a long time to recognize that an incorrect solution path is being pursued and much effort may be wasted. In addition, the ultimate value of a directed search depends on the termination criteria. While a directed search can reduce the cost of completing a particular solution, it may not eliminate the need to consider alternative solution paths in order to satisfy the system goals (Section 3.3). However, the creation of potential high-level solutions with more reliable credibility ratings may still reduce the computational effort necessary to pursue the alternatives.
2. Search breadth-first (be exhaustive in pursuing potential solution paths) when there are not statistically significant reasons for preferring any particular solution path. Typically this means when partial solutions have low credibility (or the credibility ratings are unreliable) or when there are many competing alternatives with similar ratings. The advantage of this approach is that it allows the system to build up a set of constraints that can be used to

more reliably direct the construction of higher-level hypotheses so that further search can be directed. This approach is often necessary at lower levels because credibility ratings of lower-level hypotheses may be unreliable (since they represent very weak constraints). Another reason for working breadth-first is that the termination criteria may require that most of this work be done anyway. This means that the ultimate value of doing directed search based on weaker constraints is low, so it is better to gather more complete constraints before doing directed search.

3. Expand the search space incrementally, only as necessary. In many applications, there is *a priori* knowledge about the likelihood of solutions having particular characteristics. From a constraint perspective, this knowledge can be used to order a set of constraints from tighter to looser based on the likelihood that a final solution will meet the constraints. The system should apply the tightest constraints (examine the a priori most likely possibilities) first to limit size of the search space and should open up the search space (using looser constraints) only as needed—based on the state of problem solving. In other words, if the system is able to meet the termination criteria applying the tighter constraints, it is unlikely that better solutions can be obtained by opening up the search space (considering a priori less likely alternatives). This strategy was used in HSII via thresholds on generator KSs (see Section 5.4) and in the RESUN system through the posting of inference failure assumptions (see Section 6.9).
4. Perform *differential diagnosis* whenever practical. In other words, when they are available and cost effective, use methods that can directly differentiate between competing solutions instead of using an incremental hypothesize and test approach. When a hypothesis is uncertain due to the existence of a competing, alternative hypothesis, differential diagnosis means that the system should attempt to find constraints (data) that are consistent with only one of the alternatives. For example, in interpretation problems, this means that the system should try to discount the potential *alternative explanations* for a hypothesis' supporting evidence. Differential diagnosis is a direct method for resolving hypothesis uncertainty whereas hypothesize and test is an indirect method [Carver90a]. The advantage of using direct methods for resolving uncertainty is that they can have a much higher value in terms of meeting the termination criteria than indirect methods [Carver90b]. The use of a differential diagnosis strategy is made possible by the blackboard's integrated representation of alternative hypotheses. However, HSII did not do explicit differential diagnosis due to certain limitations in its representation of hypothesis relations (see Section 5.6). It has been used extensively in the RESUN system (Section 6.9).

4 Themes in the Evolution of Blackboard Control

As we saw in the previous section, effective control in blackboard systems can be difficult. The blackboard model can lead to complex interrelationships among potential actions, numerous strategies for pursuing each possible solution path, and even uncertainty whether the system goals have been satisfied. These factors can severely complicate the blackboard control problem since they make it difficult to reliably select the actions with the maximum expected value. We have identified four key themes in the evolution of blackboard control that address these issues. The initial development of these lines of evolution can, in fact, be seen in several mechanisms used in the HSII implementation and will be discussed in the next section. The four evolutionary themes are:

1. Extension and formalization of mechanisms for goal-directed control without loss of opportunistic control capabilities.
2. Development of abstract models of the search space that can be used to make more accurate estimations of the long term, global value of potential actions and to evaluate satisfaction of the termination criteria.
3. The development of architectures that support the specification and application of explicit and sophisticated (highly context-specific) control strategies.
4. Concern with the efficiency of blackboard control.[13]

In order to make effective control decisions, the expected value of potential actions (toward meeting the termination criteria) must be judged not only in terms of the local and immediate effects of the actions, but also in terms of their global and long-term effects. The development of explicit representations of detailed goals makes it possible for the system to understand how potential actions can contribute to the overall problem-solving goals and to understand the relationships among the actions. There has been a significant trend toward making goal-directed control techniques like subgoaling and planning an explicit part of blackboard control mechanisms. The key issue in using goal-directed reasoning techniques for blackboard systems is how to make control more goal-directed without sacrificing the opportunism of the basic agenda-based control model. For example, "classical planners" are neither opportunistic nor reactive, which makes them unsuitable for blackboard tasks that involve uncertainty and/or dynamic situations. Thus, the planners used for blackboard control are typically *incremental* planners in which planning and execution are interleaved [Carver93].

Effective blackboard control may also require abstract models of the emerging structure of the search space. By this we mean that it may be necessary to build appropriate models in order for the system to understand how alternative actions (taken at various intermediate states of problem solving) can affect the search process and to allow it to assess whether the termination criteria have been met. Effectively, this involves being able to more reliably judge the value of potential actions in terms of moving the system closer to meeting the termination criteria.

In the basic agenda-based blackboard architecture, all the control (strategy) knowledge of the system is represented in the single scheduler rating function. This makes it difficult to encode and modify complex control strategies because the knowledge and reasoning are not explicit [Carver93]. A major area of evolution has been to make control reasoning more explicit to support the use of *sophisticated* control strategies—i.e., control strategies that involve large amounts of highly context-specific search knowledge. This allows for better estimation of the value of potential actions because

[13] "Efficiency" here does not mean effective search control. Effective search is a key issue for problem solving in general. "Control efficiency" refers to *overhead*: the time spent making control decisions relative to taking actions. A control mechanism could be effective at limiting search and yet be so inefficient at making decisions that overall performance (e.g., CPU time) is not acceptable. Another issue in the development of blackboard systems has been efficient storage and retrieval of hypotheses [Corkill86]. Since our focus is control, we will not address this subject.

criteria that are specific to each particular problem-solving situation (context) can be used rather than a complex, general mechanism.

The first three evolutionary themes are all concerned with the development of techniques that allow for more accurate estimations of the long-term value of potential actions in meeting system goals. An important idea that has been used in conjunction with these techniques is that instead of attempting to compute absolute expected values for all potential actions, control decisions can often be made through comparative (relative) evaluations of a subset of the potential actions. The advantage of this approach is that comparative evaluations can be more reliable than global evaluations since a limited set of factors needs to be considered. For example, in systems with explicit subgoals, if particular subgoal(s) can be identified as having the highest value to the system, only those actions that are relevant to the subgoal(s) need to be evaluated. In making the relative evaluations of these actions, only their value in satisfying the particular subgoal(s) needs to be considered—not the value of the subgoal(s) in meeting the overall goals. This approach does require that a system have enough information to be statistically effective in identifying the appropriate subset of actions to evaluate—i.e., that the probability and average cost of being wrong are both sufficiently low that better relative evaluations lead to more effective control overall.

Effective control not only involves making appropriate control decisions, it also requires that those decisions be made *efficiently*. Efficient blackboard control involves techniques for preventing the blackboard and the agenda from becoming overloaded; making the selection (rating) of KSIs efficient; and limiting the KSIs that are repeatedly re-rated. Allowing a large number of hypotheses onto the blackboard affects the cost of control because it may make it more costly to check KS preconditions and because it is likely to lead to more KSIs on the agenda since more KSs will be triggered. Allowing a large number of KSIs onto the agenda can make control less efficient because it can make the action selection process more costly and because it requires that ratings be more reliable. HSII had an relatively inexpensive numeric mechanism for rating KSIs. However, even inexpensive numeric rating schemes can become ineffective when faced with large numbers of KSIs—especially if the KSIs are re-rated relatively often. In such cases, it would be important to limit the KSIs that get placed onto the agenda to those that are "likely to be executed" and to limit the re-rating of KSIs.

Efficient blackboard control is critical for applications involving real-time deadlines, but it is important in other situations as well. For example, it is important when there are large numbers of methods available for pursuing hypotheses (as in the DVMT using approximate processing KSs [Decker90]) or when there are numerous direct methods for resolving uncertainty (as in RESUN [Carver90a]). Control efficiency can also be a major concern with interpretation applications in which sensors continuously produce large amounts of data. Each piece of sensor data that is inserted onto the blackboard may trigger the creation of one or more KSIs, but only a fraction of these actions will ever be able to be executed. This can lead to an agenda whose size increases during problem solving and to a large number of KSIs remaining on the agenda, being repeatedly re-rated [Carver90b].

5 Additional Control Mechanisms in Hearsay-II

In this section we will examine several control mechanisms that the HSII implementation added to the basic control model of Section 2.2. These mechanisms represent initial attempts to extend the blackboard control architecture to allow the use of more sophisticated goal-directed control strategies. Thus, they can be seen as precursors of the evolutionary lines that were identified in the preceeding section. For example, the HSII implementation included mechanisms to do some simple subgoaling (the *predict* and *verify* KSs) and to implement special-purpose strategies (via large-grained KSs). We have already seen how HSII used its focus-of-control database and the stimulus and response frames of the KSIs to attempt to assess the global and long-term value of possible actions via the scheduler rating function. HSII also used several special KSs that made global assessments of the work needed for termination: generator and policy KSs, the *word-seq* KS and the *stop* KS. While HSII did not maintain explicit goals identifying the work required for termination, these KSs developed implicit models of where additional work was likely to be effective for meeting the termination criteria. Finally, even though HSII did not do explicit differential diagnosis, it was able to implement a limited form of differential diagnosis through KSI clustering.

5.1 Predict and Verify

The special KSs *predict* and *verify* were one way that HSII extended its goal-directed reasoning capabilities. Effectively, these KSs allowed HSII to do some implicit *subgoaling* to focus low-level processing. *Predict* made predictions about words that could possibly extend a phrase. *Verify* then confirmed or discounted the predictions by directly looking at the acoustic data. Thus, the predictions from potential phrase-level hypotheses are in a sense used to focus low-level (data) processing because they limit the set of interpretations of the data that will be examined. The limitations of this approach to subgoaling will be discussed in Section 6.3.

HSII also foresaw the need for scheduling sequences of KSs: the *predict* and *verify* KSs were rated based on the same criteria so that they were always scheduled together. This was done because the output of *predict* did not prove to be a good estimator of whether this line of reasoning (i.e., phrase hypothesis) should be continued—in contrast to some other path. While the number of words predicted from each phrase did give an estimate of the amount of work that would be required to pursue that phrase, it was not possible to reliably balance the cost against the credibility of the phrase hypotheses using HSII's one pass, linear evaluation function for rating KSs. In other words, this is an example of the problem of computing reliable absolute measures of the expected value of possible actions. Because it was inappropriate for the output of *predict* to stand alone and influence blackboard processing decisions, the decision was effectively delayed until the result from the *verify* KS was available. However, the expected cost of verifying predictions was used in deciding, from a local perspective, which predictions to verify. Predictions could be backward in time or forward in time. The precondition of *verify* used cost information to decide whether to instantiate the *verify* KS to work in both directions or only one direction (on the predictions from a single phrase). This was a reasonable strategy since all other scheduling factors were the same here except for cost.

The joint scheduling of the *predict* and *verify* KSs also involved the issue of efficiency. Instead of having *predict* create (prediction) hypotheses on the blackboard for each of the potential extension words, the list of potential words was passed to *verify* by being attached to an attribute of the hypothesis that was the basis for the prediction (the change in this attribute triggered the *verify* KS). *Verify* then checked each of the words attached to the triggering attribute against lower-level hypotheses to see which were possible and created hypotheses only for these possible words. *Predict* would typically create a large number of words, most of which could be immediately discounted by *verify*. Placing all the intermediate hypotheses on the blackboard would have been inefficient because it would have caused overloading and because creating blackboard hypotheses is typically more costly than creating specialized local representations.

5.2 Large-Grained KSs

While the *predict* and *verify* KSs were kept separate for modularity and simply scheduled together, HSII implemented several specialized strategies through large-grained KSs. For example, later configurations of HSII had low-level KSs that processed all the input data at once in order to force processing to be strictly bottom up to the word level. This was done because it was found that credibility ratings were not reliable enough at the lower levels to support effective opportunistic scheduling. While the solution of large-grained KSs is not general, experience with HSII demonstrates the need for context-specific specialized strategies. The use of highly context-specific control strategies or what we have termed sophisticated control strategies [Carver93] are addressed by both BB1 and the RESUN framework which permit more explicit representations of detailed strategies.

5.3 STOP: Termination

HSII dealt with termination through the special *stop* KS. This KS was triggered by the creation of a highly rated, complete phrase-level hypothesis. The *stop* KS then examined the existing alternative hypotheses and "pruned" those that were unlikely to be able to produce higher-rated answer hypotheses. It did this by looking at the ratings of word hypotheses not covered by the alternative phrases and determined whether these words could possibly improve the alternatives so that they would be better than the current phrase [Lesser77b]. This was a heuristic process, since it was possible that *predict* and *verify*, working top-down, could generate new words in an area that would be more highly rated. *Stop* halts processing when all the potential alternatives have been removed.

Reasoning about termination requires a global view of the state of problem solving. *Stop* achieved a global view by using the focus-of-control database and because it could examine the entire blackboard. One way of viewing the *stop* KS is that it implements a sophisticated global control strategy that determines whether particular hypotheses are worth pursuing (it removes those that aren't). HSII's *stop* KS can also be viewed as a type of control KS (see Section 6.4) since it was effectively making control decisions about whether to proceed with problem solving.

5.4 Generator and Policy KSs

One of the special features of the HSII implementation was its use of *generator* and *policy* KSs—e.g., *mow* and *word-ctl.* Generator KSs were large-grained synthesis KSs that were capable of creating all plausible explanations for hypotheses at some level. However, instead of always creating hypotheses representing *all* the explanations, generator KSs could be controlled so that only a portion of these hypotheses would be created at once. This control was provided by corresponding policy KSs that specified how many hypotheses a generator KS should create and where in the space. Should processing "stagnate," policy KSs may be triggered to change the criteria for generator KSs and retrigger these KSs.

One way to view the generator/policy KS approach is in terms of implementing a strategy of incrementally opening up the search space. Initially the policy KSs will specify thresholds such that only the (a priori) most likely hypotheses are created. If problem-solving conditions suggest that the valid solution may have been excluded by these decisions, the policy KS can lower the threshold or otherwise direct the generator KSs to post hypotheses that were previously considered too unlikely to pursue. The creation of these additional hypotheses will trigger additional actions and thus extend or redirect the search process.

Another view of policy KSs is that they provide a mechanism for implementing a more global search than would be possible with a basic scheduler—i.e., they provide a method for integrating non-local, goal-directed control factors in order to control the combinatorial explosion of hypotheses that might otherwise be created. In fact, in later versions of HSII the policy KSs actually mediated/triggered generator KS actions via special "goal" hypotheses [F. Hayes-Roth77]. Thus, HSII extended the model of KSs so that they are not only controlled by hypotheses, but also by goals. Because of this we can view HSII "goals" as being a precursor of the kinds of goal units found in the goal-directed blackboard architecture (see Section 6.3). Goals restrict the actions and output of KSs—e.g., while a KS could generate a range of outputs in a certain space, goals limit what output it actually covers and when the KS is triggered.

Yet another way of viewing HSII's policy KSs is that they were an early precursor of knowledge source-based control. This is a key feature of the BB1 architecture (see Section 6.4). Policy KSs were triggered not by blackboard events like standard KSs, but rather by the state of the control process—e.g., there are no reasonably highly rated KSIs on the agenda. Unlike the control plans of BB1, policy KSs could only provide a "single shot" at changing control parameters to respond to the situation and were not themselves driven by explicit goals.

5.5 WORD-SEQ

Another large-grained KS used in HSII was *word-seq* (*woseq* in early HSII papers). HSII recognized that the selection of blackboard abstraction levels and associated KSs is often driven by control considerations. Pursuing top-level answers via intermediate-level hypotheses also allows constraints to be incrementally aggregated. This can result in more efficient problem solving. Applying these partial constraints can be significantly cheaper than applying the full constraints and may eliminate large numbers of hypotheses from consideration. In other words, the ability to create intermediate-level hypotheses allows the application of inexpensive constraint knowledge that can significantly reduce the search space for the higher-level interpretations without incurring the cost of applying the full, top-level constraints.

In HSII this approach was taken with the *word-seq* KS. While most of the KSs (and blackboard levels) in HSII represent the application of different types or classes of knowledge, in part, the *word-seq* KS really just represents an *approximate* version of the *parse* KS that creates phrase-level (multi-word sequence) hypotheses from sequences of word-level hypotheses. This is done because the credibility ratings of individual word-level hypotheses is not reliable enough to (opportunistically) direct interpretation to the phrase level and the complete phrase-level knowledge is costly to apply. The *word-seq* KS is able to efficiently generate multi-word "islands" whose credibility ratings are reliable to direct further processing. This allows HSII to efficiently filter the multi-word sequences before the relatively costly knowledge of the *parse* KS was applied. *Word-seq* is more efficient than *parse* because it only does pairwise checking of word sequences (using bit-tables). While this can result in ungrammatical phrases, these will be eliminated by the application of the more complete grammatical knowledge in *parse*.

Word-seq is not just an approximate KS, it is also a large-grained KS that implements a specialized control strategy. *Word-seq* processes the entire set of word-level hypotheses at once (remember, other large-grained KSs process the complete set of data in a strictly bottom-up manner to the word level). This provides an overall view of where the most reliable multi-word "islands" are. In order to do this, the *word-seq* KS must: decide on the length of the sequences to be produced based on ambiguity considerations (among competing multi-word extensions) and make comparisons between alternatives to decide whether additional hypotheses are worth posting. Since *word-seq* is also a generator KS (with associated policy KS *word-seq-ctl*), it may be re-triggered to post "less plausible" islands if processing stagnates.

5.6 KSI Clustering

While explicit differential diagnosis was not used in HSII, the implementation did include a technique for implicitly doing some limited differential diagnosis through "KSI clustering." This involved pursuing sets of similarly rated hypotheses together. Though the similarly rated hypotheses were not necessarily competing alternative hypotheses, the technique was very useful when they were and caused no harm when they weren't. KSI clustering was developed because when similarly rated hypotheses were competing alternatives, HSII's "island driving" strategy would cause whichever hypothesis was first extended to then be pursued to the exclusion of the other alternatives. This is because the system had no way to recognize that one alternative had become more highly rated than another simply because more evidence had been gathered for it—not because there was any evidence against its alternative.

In part, this problem arises because HSII used a very simple representation (single numbers) of the credibility of hypotheses. However, even if the characteristics of the evidence had been better represented, this information would have been difficult to exploit. In conventional agenda management, the rating of each KSI is done independently of rating the other KSIs. This local evaluation procedure limits the ability of the control system to understand the relationships among possible actions. Where explicit relationships among KSIs have been exploited [Lesser89], additional stages of processing beyond the local evaluation have been required (see the discussion of goal relationships in Section 6.3).

5.7 Triggered versus Executable KSs

HSII used another idea for focusing resource usage. In early versions of Hearsay-II, once a KS was triggered, the precondition procedure was placed onto the agenda instead of immediately being executed. This made it possible for the scheduler to control both the execution of actions and the execution of preconditions. Control over precondition execution gives a system more ability to focus its limited resources—if there is a good way to identify unpromising KSIs from their triggers. This could be very effective, for instance, if precondition procedures are relatively costly or if only a small percentage of KSIs are eventually executed. It could also be effective at keeping the agenda from becoming overloaded with KSIs and making the scheduling process inefficient. However, in HSII, precondition scheduling was not ultimately pursued because knowledge was not available to support decisions about the overall value of action versus precondition execution. This is an instance of the general meta-level control issue of deciding whether to take domain or control actions. Here, the issue is whether there is value in taking information gathering actions (that identify new possible domain actions) or whether the system should just proceed with the current set of actions. Balancing control versus domain costs has become an important concern as blackboard systems are used for real-time problem solving (see Section 7).

6 Blackboard Control Architectures

The previous section showed how the HSII implementation included a number of mechanisms to extend the capabilities of the basic HSII control model. In this section we will examine a number of blackboard control architectures that have evolved from that basic HSII model: HASP/SIAP's event-based control, CRYSALIS' hierarchical control, the goal-directed architecture of the Distributed Vehicle Monitoring Testbed (DVMT), the control blackboard architecture (BB1), model-based incremental planning for the DVMT, the channelized, parameterized control loop version of the DVMT, ATOME's hybrid multistage control, CASSANDRA's distributed control, and the RESUN interpretation framework. These particular architectures were chosen because they depart significantly from the HSII model and/or because they have been widely studied. The systems will be presented in roughly chronological order (as listed) because the cross-fertilization of ideas makes it difficult to distinguish clear evolutionary paths.

We will analyze each of the architectures in terms of the evolutionary themes presented in Section 4. The development of more explicit goal-directed control mechanisms has been a major area of concern. The goal-directed architecture of the DVMT was one of the earliest blackboard systems to use explicit goals. BB1, the model-based incremental planner for the DVMT, and RESUN all have planning-based control mechanisms. The use of abstract models of the search space has been a key concern in blackboard systems for sensor interpretation: the goal-directed blackboard architecture, the model-based incremental planner for the DVMT, and RESUN. Support for explicit, context-specific control strategies was a major issue in CRYSALIS, BB1, and RESUN. Efficiency concerns played a role in CRYSALIS, the model-based planner for the DVMT, RESUN, and in real-time extensions to BB1.

6.1 HASP/SIAP: Event-Based Control

The HASP/SIAP project [Nii82, Nii86b] for interpreting sonar signals was an early system based on the blackboard model of HSII.[14] However, instead of the agenda-based scheduler mechanism of HSII, HASP used a control mechanism based on the occurrence of predefined *events*. In HSII, changes to the blackboard were described in terms of a set of *blackboard event types* that then triggered the appropriate KSs (to evaluate their preconditions). HASP took this event-based approach much further, making events the primary basis of control. First, instead of reporting all blackboard changes using a set of primitive blackboard event types, HASP programmers specified the blackboard changes that were of interest to the system by defining their own set of blackboard event types. Second, HASP programmers specified a sequence of KSs that were to be executed for each event type. The main control decision in HASP was which event (instance) to select as the next focus of attention; once an event had been selected as the next focus, the KSs to be executed were predetermined. In effect, the predefined blackboard event types served as the KS preconditions in HASP and there was never any uncertainty about how best to respond to each event. The HASP architecture is shown in Figure 4 and the basic control loop is shown in Figure 5.

Another way in which HASP extended the concept of events, was by supporting three other *categories* of events: clock, expectation, and problem.[15] These events were posted directly by KSs into the appropriate *event lists*. *Clock events* consisted of a time and a list of KSs to be executed at that time. They were used to confirm the expected behavior of signal sources (based on the existing interpretation hypotheses). *Expectation events* represented blackboard events that were expected at some point in the future—e.g., the appearance of data from some vehicle that was thought to be in the area. They were used to periodically trigger the system to check for the expected events.[16] *Problem events* signalled "problems" encountered by the KSs such as information that was missing or was desired by a KS—e.g., if a KS was unable to execute any of its rules. They were used, in effect, to set up "goals" for developing particular hypotheses; an ad hoc approach to doing some limited goal-directed control reasoning.

[14] [Nii82] and [Nii86b] differ somewhat in their descriptions of the HASP/SIAP architecture. Our description of the architecture follows [Nii86b]. The AGE blackboard shell [Nii79] used a control scheme that was similar to that used in HASP/SIAP.

[15] The HASP papers are somewhat inconsistent in their terminology. While they label these the four "event categories," they also refer to "expectations" and "problems" rather than expectation events and problem events.

[16] The use of clock and expectation events is a result of differences between the HSII task and the HASP task that dealt with the continuous interpretation of signals. However, an event-based control architecture like HASP's is not required to deal with continuous interpretation. For example, expectations posted to the blackboard could trigger the creation of KSIs, but these KSIs would not be scheduled for execution until an appropriate time (based on information in the stimulus frame of the KSI). Alternatively, users could be allowed to define their own event classes—e.g., events based on time. This approach is used in the GBB blackboard framework [Corkill86, Gallagher88].

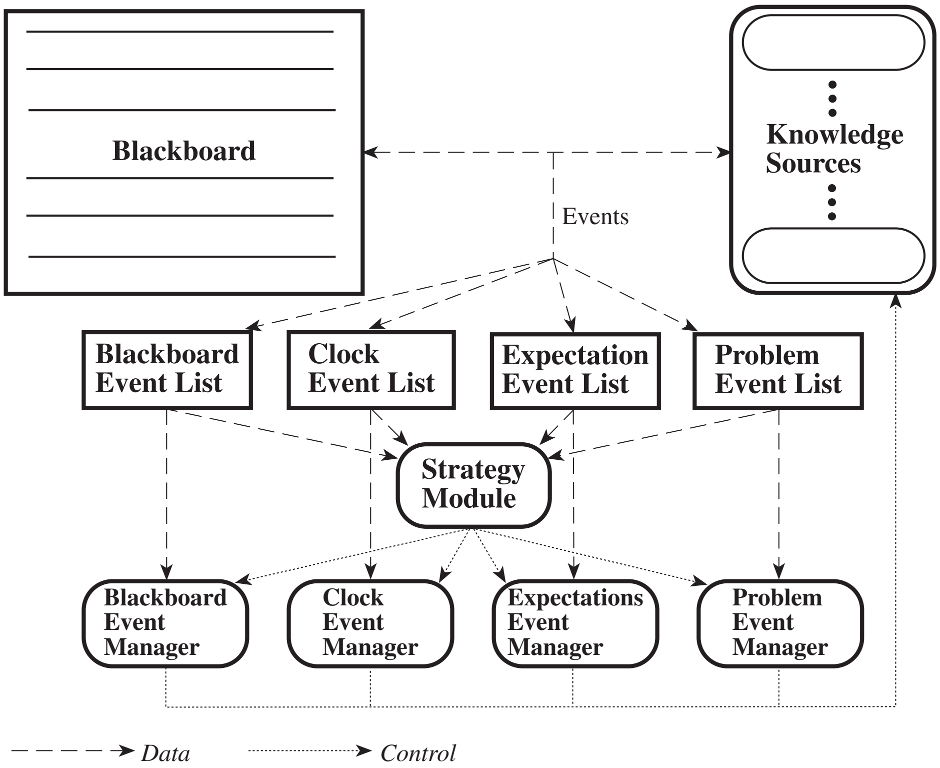

Figure 4: The HASP/SIAP blackboard architecture.
Knowledge sources are rule sets that modify the blackboard and post *events* into the appropriate *event lists*. The *strategy module* selects the event category that should be in focus next. *Event managers* select the particular event (instance) that should be in focus and identify the appropriate knowledge sources to execute using the event focus.

In HASP, the event that would become the next focus of attention was selected through a two-level process. At the top level, the *strategy module* decided what event category to focus on next. Then, the appropriate *event manager* selected the event (instance) that was to become the next focus of attention (selecting it from the appropriate event category list). Once the event was chosen, the KS(s) that were to be executed[17] were predetermined, based on the event type of the chosen event. One way of looking at this architecture is that it was a precursor of the kind of hierarchical control found in CRYSALIS (Section 6.2). [Nii86b] states that, "the hierarchical control in HASP was an attempt to separate the domain-specific knowledge from knowledge about the application of that knowledge." However, it is not clear that HASP offered any advantages over HSII, in which the knowledge about the application of the problem-solving knowledge (the KSs) was held in the scheduler function. Furthermore, HASP's hierarchical architecture was very limited since it had two fixed and specific levels: event category selection and event (instance) selection. The CRYSALIS architecture (in theory) allows any number of levels and these levels can be related (implicitly) to any system goal (task).

[17]KS actions were implemented as sets of *rules* in HASP. This same approach was used in CRYSALIS and is discussed further in Section 6.2.

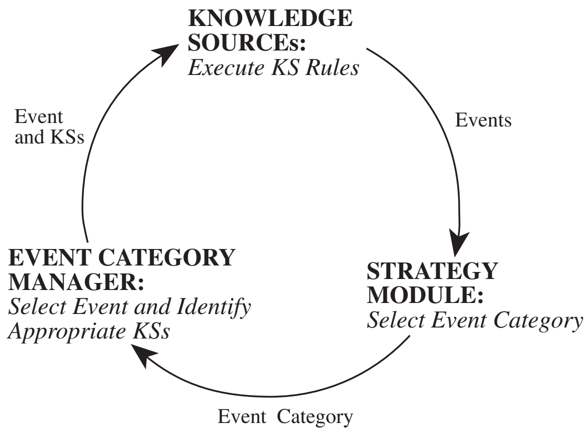

Figure 5: The basic control loop for HASP.

Another problem with the HASP architecture is that it is not clear that its decision hierarchy is appropriate for making opportunistic decisions, in the sense of the selecting the best possible actions in the situation. [Nii86b] states that in selecting the event to focus on next HASP was implicitly selecting the "solution island" to focus on. In fact, [Nii86b] says that "the focus-of-attention problem in HASP was primarily a problem of determining which island to work on next" and the control modules were "biased toward the selection of a solution to be pursued that would have the highest payoff in subsequent processing cycles." However, HASP first chose the event category and only then chose the event (and implicitly, the solution island) to focus on. The problem with this approach is that—in general—the choice of best event category can not be made without knowledge of the (best) instances of each category that are available. In other words, finding the best focus could require a search through all of the possible events. HASP made the event category decision using "priorities as encoded in its knowledge base" [Nii86b]—e.g., clock events get priority if it is time for them to be processed. Clearly, this approach may not yield the best focus—e.g., if all applicable clock events are associated with lower importance/credibility hypotheses than some other event.

Because of these issues, the HASP control architecture can also be viewed as a specialization of the HSII architecture that is efficient for the particular application. HASP did not use an agenda-based control architecture because its focus-of-attention problem was simpler than that of HSII in terms of the level of detail required for focusing. In contrast to HSII, which had to focus on actions (KSIs), once HASP focused on a particular event its actions were determined. Of course, another problem with this event-based approach is that the opportunism of the system may be limited because the designers must have correctly prespecified all the event types (i.e., situations) of interest. There is no chance for KSs that are triggered by general events to be activated in unexpected situations. Likewise, the approach is not very modular. When new event types are defined, existing event types (and their associated KSs and control modules) may have to be redefined if the new events overlap with the existing events. Since HSII rated KSIs based on the characteristics of the stimulus and response frames—not by reference to specific KSs—its control knowledge was largely independent of the particular KSs.

Because HASP involved continuous interpretation of sensor data, it addressed the issue of dealing with large amounts of data. It did this through the use of top-down (model-driven) KSs that looked for data to support important features of the high-level interpretation hypotheses. The arrival of new data does not appear to have been a signalled event (as it would be in an agenda-based system like HSII). Instead, expectations would be used to drive the system to interpret a limited amount of the data as it comes in. HASP could also examine the data looking for counterevidence as well as supporting evidence.

HASP did not make use of any models of the state of problem solving in making its decisions. In part this may be because, unlike HSII, HASP did not concurrently pursue multiple solution paths. In fact, HASP could not represent alternative interpretations (although it could represent potential alternative attribute bindings for hypotheses—i.e., uncertainty about hypothesis attributes). As a result, it was more difficult for the system to revise its interpretations: it backtracked by deleting affected hypotheses and restarting its analysis from the point of change.

6.2 CRYSALIS: Hierarchical Control

The CRYSALIS system for protein crystallography [Engelmore79, Terry88] introduced the idea of hierarchical control for blackboard systems.[18] Instead of the agenda-based scheduler mechanism of HSII, CRYSALIS used a hierarchy of "control knowledge sources" to select the domain/problem-solving KSs to be executed. CRYSALIS had two levels of control knowledge: *strategy* and *task*. The single strategy knowledge source[19] selected a sequence of task knowledge sources to be executed while task-level knowledge sources selected a sequence of domain KSs to be executed. The CRYSALIS architecture is shown in Figure 6 and the basic control loop is shown in Figure 7.

Since KSs were directly selected by higher-level KSs (or by default at the top), CRYSALIS KSs did not have precondition components as in the HSII model. However, all CRYSALIS KSs were implemented as *rule sets*[20] so selection of a KS to be executed really meant selection of a set of rules to be interpreted. In the strategy KS, rule LHSs were conditions on the *features list* (a summary of key features of blackboard hypotheses) and RHSs identified a sequence of one or more task KSs to be executed. In task KSs, rule LHSs involved conditions on the *events list* (blackboard changes were represented as *events* as in HSII) and RHSs identified a sequence of domain KSs to be executed. In domain KSs, rule LHSs could examine the entire blackboard structure while RHSs created and modified blackboard hypotheses.

[18] Another early blackboard-based system that used hierarchical control was VISIONS [Williams77a, Williams77b]. Control (strategic) knowledge is organized into a hierarchical set of modules based in part on the structure of the domain blackboard. For example, the H-L-Strategy level is concerned with selecting the model (interpretation) to pursue next and next lower level elements that select the portion of the model to pursue and the process to be used.

[19] Strategy and task units are referred to as "KSs" in CRYSALIS and they perform the same function as BB1 "control KSs" (Section 6.4). However, the CRYSALIS "control KSs" differ significantly from the BB1 model of control KSs and from the basic HSII model of KSs. In particular, there is a single strategy KS—not multiple KSs—and task KSs neither write onto a shared database nor are they self-activated (opportunistically invoked).

[20] Actually, some of CRYSALIS' domain KSs were represented as procedural code for efficiency.

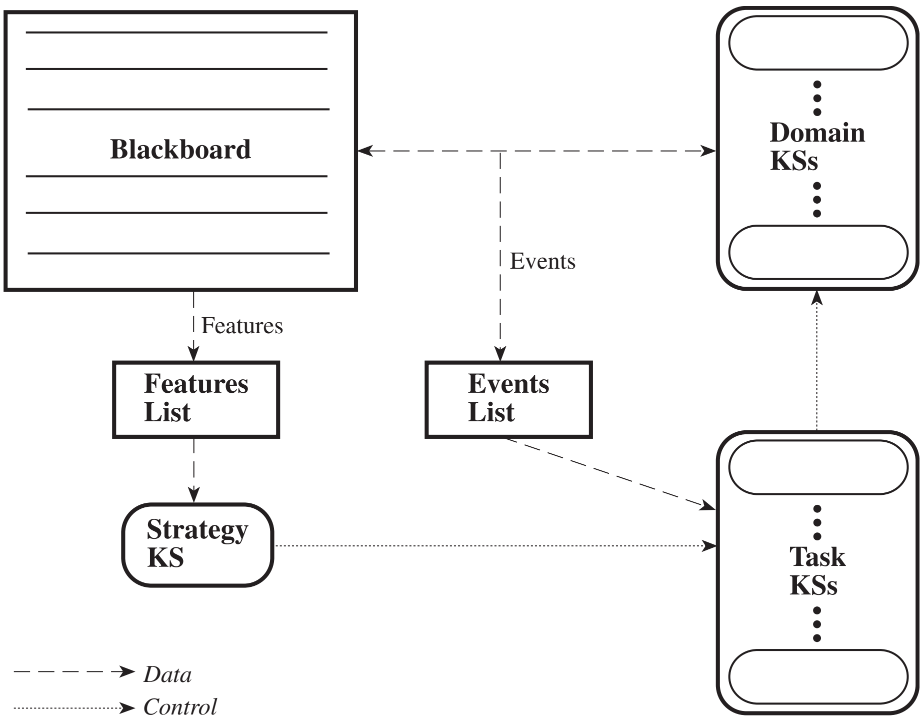

Figure 6: The CRYSALIS blackboard architecture.
All KSs are rule sets. Changes to the blackboard made by the *domain KSs* are described as *events* and the state of blackboard solutions is summarized as a list of *features*. The *strategy KS* selects a blackboard region to focus on and a sequence of task KSs to execute. The *task KSs* select events related to the in-focus region to focus on and a sequence of domain KSs to execute.

CRYSALIS' *features list* played a role similar to HSII's focus-of-control database. It provided an abstract view of the state of blackboard problem solving for use in making decisions about where to focus problem solving. The features list identified key patterns in the hypotheses that suggested it would be useful to pursue particular subgoals (tasks) next. In other words, it was intended to assist in determining where the system was most likely to make further progress—i.e., subgoals with maximum expected value. Differences in the type of information contained in the features list as compared with HSII's focus-of-control database are at least partly related to differences in the search strategies and the relationship between hypotheses and solutions between the two applications.

CRYSALIS' hierarchical control architecture provided more explicit goal-directed control than the agenda-based scheduler of HSII. The strategy KS determined appropriate tasks (effectively, subgoals) to be pursued next while the task KSs selected the best methods (sequences of domain KSs) for accomplishing the task. Another way of looking at the CRYSALIS control mechanism is that the strategy KS provided the coarse focus of the system and the task KSs provided the fine focus of the system: the strategy KS selected the goals to pursue next (i.e., what to try to accomplish with particular hypotheses) and the task KSs then selected actions that best accomplished the goals. HSII focused directly on actions without any representation of system subgoals. The sort of multi-step focusing that is inherent in hierarchical control architectures means that control decisions

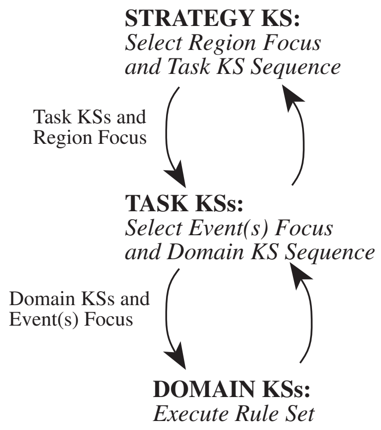

Figure 7: The basic control loop for CRYSALIS.

(action selections) are effectively made through comparative evaluations—as opposed to selecting actions by determining their expected value. For example, the strategy KS only considered (a limited set of) general tasks that might be pursued; it could not determine the expected value of each task since it did not know how the tasks would be carried out.

Of course, one of the advantages of the agenda-based approach is the high degree of opportunism that it supports: the system can switch between alternative potential solutions, subgoals, and methods on every cycle. In providing some simple goal-directed control, the CRYSALIS architecture limited opportunism. This is not to imply that CRYSALIS was not capable of opportunism: the strategy KS was opportunistic in selecting the next task/subgoal to be pursued, task KSs responded to the events list to adapt their methods to the specific situation, and the actions actually taken by the domain KSs depended on the state of the blackboard (acting through the domain KS rule conditions). In addition, both task and domain KSs could include rules that explicitly terminate the KS. However, the architecture did not allow for multiple tasks to be pursued concurrently and, once selected, a task or domain KS was run to completion (or explicit termination). Thus, there was no way to easily switch to some other focus or perform a search for appropriate subgoals or methods to be pursued next. We will see goal-directed mechanisms that provide these capabilities in Sections 6.4 and 6.9.

Part of the reason why the CRYSALIS architecture has less opportunism than HSII is a result of differences in the termination criteria, the representation of potential solutions, and the basic search strategies used in the two systems. CRYSALIS pursued a single line of reasoning at a time and backtracked when necessary. HSII could concurrently pursue multiple lines of reasoning and did not do explicit backtracking because of its ability to represent alternative interpretations in an integrated fashion on its blackboard. In addition, the CRYSALIS application did not require real-time or reactive performance that would have made the ability to rapidly switch focus more critical.

The CRYSALIS control architecture provides the ability to coordinate sequences of actions as in the planning-based control mechanisms that we will examine in Sections 6.5, 6.4, and 6.9.

However, CRYSALIS' hierarchical control mechanism does not provide all the capabilities of these more general planners. For example, its ability to do hierarchical refinement of subgoals (problem reduction) is very limited since tasks are essentially "flat" plans with a restricted language for sequencing domain KSs (though the sequencing of actions taken by each KS can be quite complex as a result of the rule set representation). In addition, CRYSALIS does not create an explicit representation of its goals and subgoals that can be used both in making and understanding control decisions. While each instance of a task KS corresponds to a subgoal of the overall problem, these subgoals and their role in the overall problem are not made explicit.

Because all CRYSALIS KSs are implemented as sets of rules, CRYSALIS can be viewed as a hierarchical production system as well as a blackboard system.[21] There is nothing about the concept of a hierarchical blackboard control architecture that requires KSs to be represented as rules, of course. This representation has both advantages and disadvantages. It makes the possible behavior of a KS more explicit than if the action portion was represented as procedural code. However, the exact behavior of a KS may not clear when a KS involves a number of rules since it may not be obvious which rules are going to be fired. This source of uncertainty is compounded by the fact that the mode of invocation of the rules (i.e., single hit, multiple hit, cyclic) can be changed from the KS's default mode by the invoking KS.

One advantage of this hierarchical approach in which actions are directly identified is that it can be much more efficient: the reasoning required to evaluate and select actions is reduced since only a limited set of actions out of all of the possible actions will be relevant to a particular task. In addition, because tasks identify sequences of actions, this eliminates the need to repeatedly reason about the most important (sub)goal and best action as in the standard HSII agenda-based control model. It may also eliminate the cost of running KS precondition procedures. Tasks specifying sequences of KS invocations effectively represent compiled knowledge about the actions that are possible and appropriate in a particular situation. On the other hand, the rules format does make control reasoning more explicit than in systems that use complex numeric rating functions.

6.3 The Goal-Directed Blackboard Architecture

One response to the problem of integrating goal-directed factors with data-directed, agenda-based control was the development of the *goal-directed blackboard architecture* [Corkill82a, Corkill82b, Lesser89]. The framework was first implemented in the Distributed Vehicle Monitoring Testbed (DVMT) [Lesser83]. The goal-directed blackboard architecture extends the HSII architecture by adding a *goal blackboard* and a *goal processor*. The goal processor instantiates *goals* on the goal blackboard, whose structure mirrors that of the domain blackboard. The goal processor is driven

[21] Another early hierarchical production system architecture was used in NEOMYCIN [Clancey86a, Clancey86b]. Control knowledge in NEOMYCIN was defined in terms of *tasks* that consisted of sequences of (meta-)rules. Meta-rules could invoke either domain rules or other tasks. Like CRYSALIS, each of NEOMYCIN's "tasks" included an *end condition* that could abort the task (or its subtasks) at any time—e.g., if the goal of the task became satisfied or if some prerequisite for doing the task became unsatisfied. Thus, the NEOMYCIN architecture was very similar to the CRYSALIS architecture, but was more flexible as it could have an indeterminate number of control (task) levels. However, NEOMYCIN's architecture was still incapable of the kind of control search necessary for complete opportunism in a hierarchical control architecture (see Section 6.9).

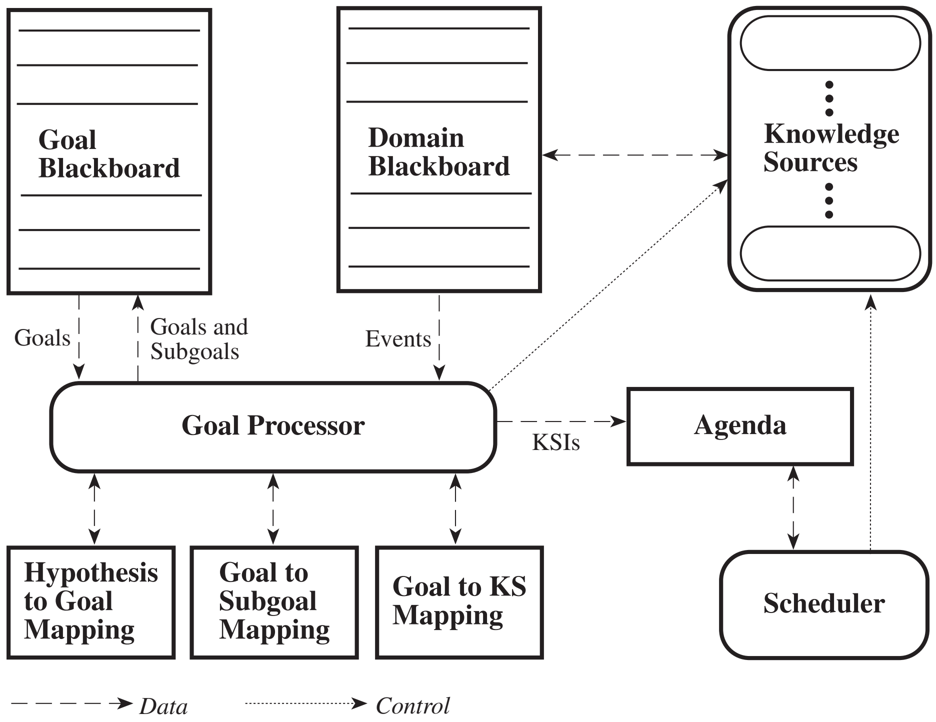

Figure 8: The goal-directed blackboard architecture.
Compared to the agenda-based architecture of HSII (Figure 1), the goal-directed architecture adds the *goal blackboard* and the *goal processor* replaces the blackboard monitor. The goal processor uses the *hypothesis-to-goal*, *goal-to-subgoal,* and *goal-to-KS* mappings to post goals/subgoals and trigger KSs.

by three mapping functions: a hypothesis-to-goal map, a goal-to-subgoal map, and a goal-to-KS map. The goal-directed blackboard architecture is shown in Figure 8 and the basic control loop is shown in Figure 9.

The goal-directed blackboard architecture integrates data-directed and goal-directed reasoning through its explicit goals because these goals can be created in two ways. *Data-directed goals* are created by the goal processor in response to the creation or modification of hypotheses on the domain blackboard, based on the hypothesis-to-goal mapping function. Data-directed goals represent the *ability* to create hypotheses with particular characteristics. *Goal-directed goals* are created in response to the creation of other goals based on the goal-to-subgoal map. This is a form of *subgoaling.* Goal-directed goals can also be created when a KS precondition procedure fails. If the failure is due to the lack of some appropriate hypotheses, the precondition procedure can return information that is used to post goals to direct the creation of these hypotheses. This is a form of *precondition-action backchaining.* Goal-directed goals represent the *desire* to create hypotheses with particular characteristics in order to satisfy other goals.

When a goal is inserted onto the goal blackboard, it may trigger KSs that can achieve the goal—identified by the goal-to-KS mapping function—to have their preconditions checked (the conditions under which goals trigger KSI creation are discussed below). If a KS is determined to be likely to generate hypotheses that would satisfy the goal, an appropriate KSI is added to the agenda. A KSI rating reflects both data-directed and goal-directed factors because it is based on

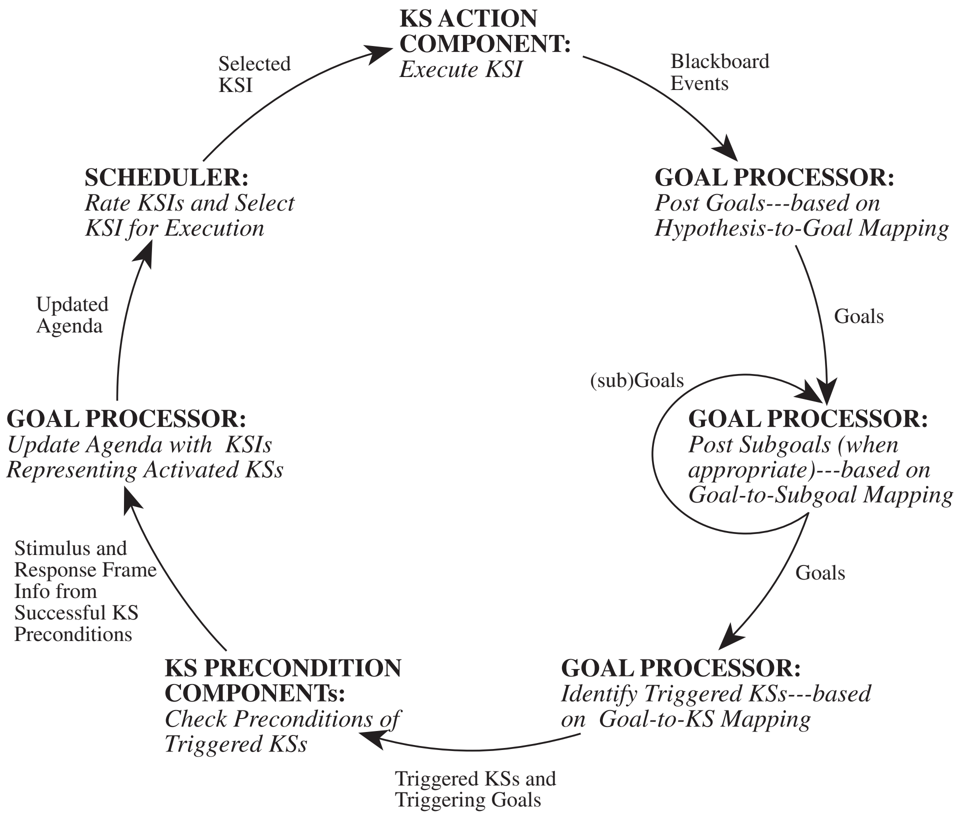

Figure 9: The basic control loop for the goal-directed blackboard architecture.

the rating of the goal that triggered the creation of the KSI (goal and data-directed factors) and on the expected credibility of the output hypotheses of the KSI (data-directed factors). Goal ratings reflect both data-directed and goal-directed factors because a goal is rated based on the ratings of any hypotheses that stimulated the creation of the goal and/or the rating of its supergoal (data and goal-directed goals can be merged—see below). By adjusting the weighting factors in the goal and KSI rating functions, the relative role of data and goal-directed control factors can be modified.

As we have said, goal-directed reasoning techniques can be critical for effective control in complex domains and the ability to understand subgoal/supergoal relationships is a key aspect of goal-directed reasoning. Subgoaling using explicit goals allows the goal-directed blackboard architecture to understand how actions are related to high-level goals (and to each other). The addition of goals allows a system to connect the immediate effects of a KS action with higher-level goals via the subgoal relations. While response frame information allowed HSII to understand what results could be derived from the data, HSII was unable to consider in detail how possible actions related to its goals.

Subgoaling can be crucial for focusing processing at the lower levels in the blackboard. To implement reasoning analogous to subgoaling, HSII used the special *predict* and *verify* KSs. One of the major advantages of using explicit goals is that it gives the system the flexibility to asynchronously, incrementally develop hypotheses because the subgoaling process leaves around a representation of subgoals at the different levels. By contrast, HSII jumped levels in making its predictions and performed the entire subgoaling and verification process in (what was effectively) a single step: the *predict* KS posted predictions at the word level and the *verify* KS jumped right to the data level to immediately confirm the predictions. The use of goals also avoids the issue of treating prediction hypotheses specially (since they do not have supporting data) so that they are not used to provide (invalid) support for hypotheses that are alternatives to the original predicting hypothesis [Corkill82b]. Since the purpose of making predictions is to guide control decisions, placing the "predictions" on a separate goal blackboard makes their role in control clear. One reason that *predict* and *verify* were scheduled as a unit was for efficiency: to avoid placing lots of prediction hypotheses onto the blackboard. The use of goals may help with efficiency because goals can represent abstractions of hypotheses—e.g., a goal can represent a *set* of predicted/desired hypotheses.

While subgoaling is a very powerful technique, it needs to be carefully controlled. The DVMT application makes it possible to use an efficient goal-to-subgoal mapping function, however, subgoaling may require complex and costly computations in other domains (since it involves goal decomposition). Furthermore, the creation of a large number of (sub)goals may degrade control efficiency since it can trigger the creation of an excessive number of KSIs. The basic goal-directed blackboard framework uses several methods to control subgoaling.[22] First, subgoaling is done only from particular (typically the higher) abstraction levels and only when goal ratings are above a specified threshold. Second, when goals are posted, they are analyzed to see if there is "significant overlap" with other goals. When there is, and when the goals are to be satisfied by the same KS, the goals and the KSIs are *merged* into a single KSI with the combined stimuli of the original KSIs. This improves performance by reducing both scheduling and computation overhead.[23] Goal merging also plays a role in limiting the KSIs that result from subgoaling. Instead of triggering KSs whenever a goal is created, KSs are only triggered by data-directed goals (including merged goals that include a data-directed goal). In other words, KSs are triggered only when there is some reason to believe that action may currently be possible.

The potential of subgoaling to overload the agenda is further limited because the goal-to-KS mappings are priority ordered (when multiple KSs are applicable to a goal). Instead of evaluating all of the relevant KSs and possibly instantiating multiple KSIs, the KS preconditions are successively evaluated (by priority) until one is satisfied. This represents an approximate processing strategy since goal satisfaction is viewed from a very local perspective. If a KSI "fails," the goal that it was supposed to satisfy can be re-stimulated in order to produce alternative KSIs to try. However, there is no way to pursue alternative KSIs if the initial KSI "succeeds" in satisfying its immediate goal, but the results of the KSI eventually prove poor in meeting higher-level goals. This is a complex problem since it involves issues of relative satisfaction of goals, understanding how low-level goals are related to the overall system goals, and credit/blame assignment throughout a goal/subgoal structure.

[22] Because of the importance of controlling subgoaling, work was done to extend this architecture through the addition of goal and hypothesis filters (see Section 6.6) and through the use of a planning mechanism (see Section 6.5).

[23] Merging of actions does decrease opportunism. However, in the DVMT, the increase in efficiency more than offset the slight decrease in opportunism that resulted.

The explicit goals of the goal-directed blackboard architecture were the basis for another technique for improving efficiency: KSs were modified so that their output was constrained by the goals that triggered them. Each goal explicitly represents the system's desire to create one or more hypotheses with particular attributes. KSs used the goals to limit the created hypotheses to those that are desired. This in turn limits the goals and KSIs being created to those that are relevant to the lines of reasoning being pursued. This extension of the basic HSII model of KSs can be viewed as a generalization of one of the ideas behind the policy KS and "goal" hypothesis mechanism of HSII. Besides simply re-triggering a KS to post additional hypotheses through lowered thresholds, goals here can specify more characteristics of hypotheses that should be produced by the KSs they trigger.

One of the problems that was identified in HSII control was the inability to understand the relationships among actions when reasoning about control decisions. In later work with the goal-directed blackboard architecture [Lesser89], the basic architecture was extended to reason about the relationships among goals—and thus the relationships among actions. For example, experiments were run with the DVMT using two mechanisms to minimize redundant activity. One of the mechanisms involved recognizing when sufficient work had been done to verify high-level solution components, and then posting *inhibiting* (sub)goals (a new type of goal) that restricted processing resulting from subsumed subgoals.[24] This mechanism extended the goal blackboard and goal processor, and made use of the fact that the output of a KSI was constrained by the goals that triggered the KS. The other mechanism could recognize when the expected results of one possible action (KSI) subsumed the expected results of another action and could schedule the more comprehensive action—even if this action were rated lower due to the "quality" of the associated data. The scheduling process went from a one pass to a two pass approach: the first stage was the normal rating process, while in the additional stage, the highest rated KSI from the first stage was compared against all other KSIs to see if there were more comprehensive KSIs on the agenda. In effect, this two-pass process allowed for some relative evaluation of the possible actions to deal with the fact that KSI ratings can not perfectly predict the ultimate value of actions. These two mechanisms were able to improve performance in the DVMT experiments. However, their successful application depends on the necessary computations being relatively inexpensive for this application and on the outcome of actions being quite predictable.

Finally, a major limitation of the goal-directed blackboard architecture is that its goals still provide only a limited understanding of the ultimate consequences of actions. In other words, these goals represent the desire to carry out relatively simple actions like synthesizing a particular type of hypothesis or extending an existing hypothesis. The goal-directed blackboard's goals can not represent complex, long-term goals like "resolve the uncertainty in a particular hypothesis" or "take actions to differentiate between these two alternatives." Furthermore, they can not represent the

[24] Note that this mechanism is similar to the HSII's *stop* KS except that it operates via the goal mechanism instead of by directly analyzing hypotheses (this makes it more general since HSII relied on the fact that there was a single solution in each blackboard "region" to recognize "subsumed" actions). Thus, the inhibiting goals mechanism deals in part with termination. However, this mechanism does not address the issue of how to recognize that enough work had been done on a particular solution component—e.g., through the use of an appropriate model of the state of problem solving. This issue was addressed in other DVMT research—see Section 6.5.

role of the various (sub)goals in satisfying the overall system goals. More general goals such as these require sequences of actions to be accomplished. In Section 6.5, we will examine an extension to the goal-directed blackboard model that addresses this limitation by providing long-term goals via planning. Both the BB1 and RESUN systems (Sections 6.4 and 6.9) also address this issue through planning mechanisms.

6.4 BB1: The Control Blackboard Architecture

Hayes-Roth's *control blackboard* architecture [Hayes-Roth85a, Hayes-Roth88]—now usually referred to as *BB1* after the name of a particular implementation—extends the control architecture of HSII through the addition of a "control planning" mechanism. In BB1, the control problem is treated as a problem-solving task in itself. Both the domain problem and the control problem are then solved using a blackboard approach.[25] In order to do this, the BB1 framework adds *control knowledge sources* and a *control blackboard* to the domain KSs and domain blackboard of the HSII model. The BB1 architecture is shown in Figure 10. The basic control loop of BB1 is similar to that of HSII (Figure 2).

Like HSII, BB1 uses an agenda-based approach to control. However, instead of using a single, fixed scheduling function for rating actions, BB1 rating functions are selected via the control planning process. Control KSs *incrementally* construct and refine *control plans* on the control blackboard. Control plans are represented at three basic levels of abstraction: strategy, focus, and heuristic. The *strategy* level represents long-term plans. Strategies may be broken down into sequences of sub-strategies (which may in turn be broken down into sequences of sub-sub-strategies, and so on). The lowest-level (sub-)strategies are broken down into sets of *foci*, each of which effectively represents a goal that the system wants to achieve. A set of *heuristics* is associated with each focus. Heuristics are rating functions that are used to judge a possible action's relevance to the focus (goal). The BB1 scheduler uses the set of active heuristics to rate possible actions.

One of the key advantages of this control mechanism is that the rating functions being used by the scheduler can be dynamically changed. This eliminates the need to try to come up with a single complex scheduling function that produces the desired behavior throughout the entire problem-solving process (as in HSII). Instead, the control planning mechanism allows the user to define strategies that invoke appropriate rating functions for different stages of processing. As each plan "step" is completed, new rating functions are activated. In fact, "goals" need not be instantiated through the hierarchical planning mechanism, but may be directly instantiated by context-specific control KSs. This allows the control decision criteria to be highly responsive to changing problem-solving circumstances (direct posting of goals is discussed further in Section 6.9).

[25] The concept of treating the blackboard control problem as a task that should be solved using the blackboard model actually originated with OPM [Hayes-Roth79] and Hearsay-III [Erman81]. OPM was a blackboard-based opportunistic planner. It included an *executive* blackboard plane whose information could modified by KSs and which was used in making scheduling decisions. Hearsay-III was one of the first attempts to generalize and extend the HSII architecture. Scheduling in Hearsay-III could be based on very complex schemes because the scheduling functions could be changed by *scheduling KSs* that could also record control information on a *scheduling blackboard*. These same ideas form much of the basis of the design of the BB1 architecture. However, BB1 provides more structured methods for influencing control decisions and has been much more widely used than Hearsay-III.

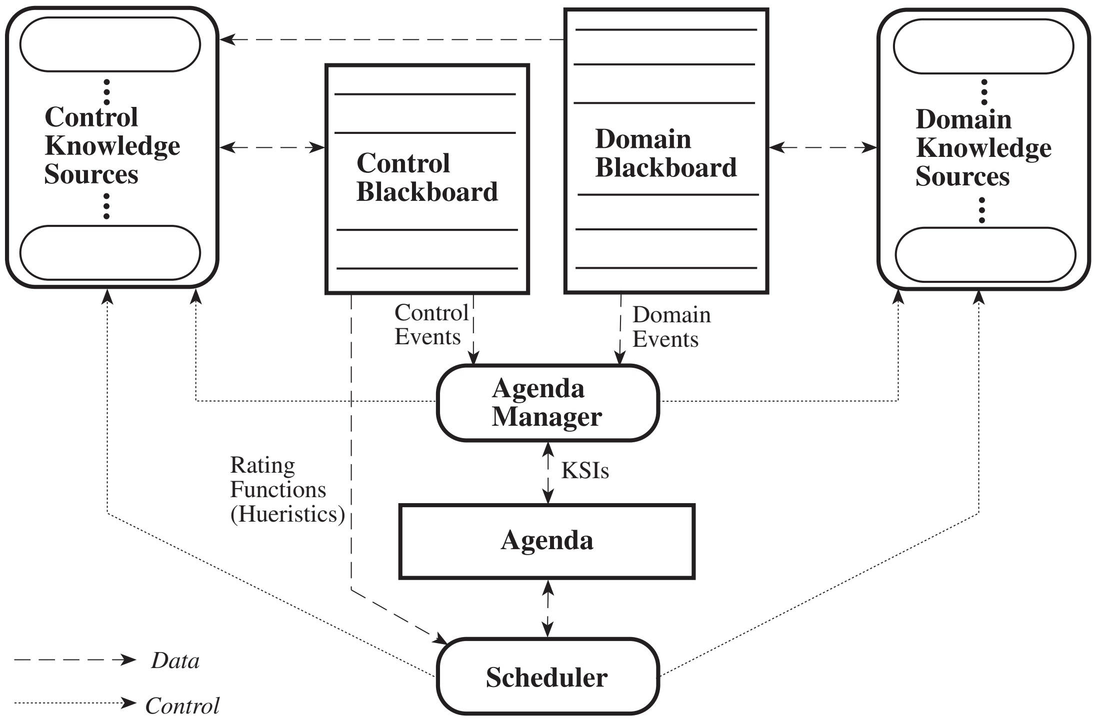

Figure 10: The BB1 blackboard architecture.
Compared to the agenda-based architecture of HSII (Figure 1), BB1 adds the *control blackboard* and *control knowledge sources*. The *agenda manager* plays much the same role as the blackboard monitor, since it determines which (control and/or domain) KSs to trigger following the execution of either a control or domain KS. However, because BB1 maintains its agenda in two parts (triggered KSIs and executable KSIs), the agenda manager is also responsible for checking the preconditions of triggered KSIs on the agenda to see if they should become executable. In addition, the agenda manager must check the *obviation conditions* of the KSIs on the agenda to decide if they should be removed. The scheduler rates the executable KSIs on the agenda using the *heuristics* (rating functions) associated with the currently active *foci* on the control blackboard, selects (and removes) the highest rated one, and invokes the appropriate (control or domain) KS action.

As noted in Section 5, HSII's policy KSs (which responded to the state of problem solving) were an early precursor of KS-based control as used in BB1.

Another way of looking at the BB1 approach is as a method for doing comparative evaluations of possible actions. As we said in Section 6.2, hierarchical control mechanisms inherently select actions through relative evaluations. In BB1, comparative evaluation manifests itself through the fact that rating functions (heuristics) judge the value of possible actions relative to their associated focus (goal)—decisions about what goals should be pursued are made through the hierarchical plan refinement process. An advantage of this kind of approach is that BB1 rating functions typically have to consider a much smaller number of factors than HSII-type rating functions (since they are not trying to judge the value of actions relative to the overall problem-solving goals). Of course, BB1 still maintains an agenda and rates all of the possible actions. By contrast, the control mechanisms of CRYSALIS and RESUN (Section 6.9) effectively partition the agenda as decisions are made at each level in the hierarchical refinement process. Thus, BB1 may be at a disadvantage unless it is easy (cheap) to identify possible actions that are not relevant to each active goal. In its favor, BB1 maintains more complete opportunism than CRYSALIS (once CRYSALIS selects a task, this task is pursued regardless of how "poor" the associated actions are—unless the task terminates itself). If there is no one goal that is most important, BB1 can have multiple active foci and then specify an appropriate combination function to judge the trade-off between the relative importance of the different goals and the relative quality of the actions (to select the action with the greatest expected value). Still, BB1 has no straightforward, explicit way to change goals if all (domain) actions are low rated[26] This capability is provided to some extent by extensions of the BB1 control loop [Collinot89] (see below). The RESUN framework provides the ability to compare actions versus multiple goals and to reconsider goals if all actions are poor—as part of an explicit control search process.

Control KSs are treated just like other KSs in that their KSIs—which represent possible (control) actions—are placed on the same agenda as the normal domain KSIs and are selected via the same scheduler rating functions.[27] In other words, there is no special control planning loop in the BB1 model. BB1 maintains the opportunism of the classic blackboard model because the identification of possible actions is done in a data-directed fashion: all possible actions (KSIs/KSARs) are placed on an agenda as they are triggered. Since both domain and control actions are treated in this manner, both actions and plans (or goals) can be opportunistically instantiated. There has also been work to extend the BB1 control loop for added flexibility. [Collinot89] allows KSs to be triggered not only by the execution of KSIs, but by the creation of KSIs (activation of KSs) and by the selection of KSIs for execution. Effectively, this allows for multiple passes through the agenda in making control decisions so that actions that are "preparatory" for other actions can be identified and executed.

Work on BB1 has emphasized the need for goal-directed control and long-term strategies for meeting goals—i.e., plans. BB1 strategies represent high-level, long-term goals as opposed to the more immediate goals of the goal-directed blackboard architecture. BB1 has also emphasized explicit representation of control strategies via the explicit control plans that it constructs. However, when describing BB1's planning capabilities, it is important to understand that the notion of planning in BB1 is somewhat different from that of a typical AI "planner." BB1 control plans implement their problem-solving strategies through rating functions that select KSIs from the agenda, rather than directly identifying actions. In BB1, the lowest level control "goals" are typically relatively general goals that will influence the selection of several actions—e.g., "work in region x" or "prefer actions of type y." This approach allows for highly opportunistic goal-directed control—e.g., when there are multiple, competing system goals and decisions must depend on the specific data or hypotheses that are available. However, the framework would make it difficult to specify detailed plans should this be desirable; there would be very substantial overhead in using the agenda and ratings functions to accomplish detailed planning. Thus, some control reasoning in BB1 is still implicit—especially since the BB1 model makes it possible to have multiple active

[26] This can be accomplished by ensuring that an appropriate "change focus" control KS has been triggered and that the herustics rate this KSI at the desired threshold to ensure it is selected over any domain actions.

[27] Possible actions are referred to as *knowledge source activation records* or *KSARs* in BB1 rather than KSIs.

strategies and goals. This forces the BB1 scheduler to use implicit reasoning methods (i.e., numeric "combining functions") to resolve goal and strategy conflicts. Another control framework that stresses the importance of explicit control reasoning and does detailed planning for control is the RESUN system (see Section 6.9).

Because the BB1 planner does not do the kind of detailed planning of a conventional planner—i.e., because it does not do goal decomposition to the level of identifying actions to satisfy its subgoals—it is unable to implement certain aspects of goal-directed control through its planner. However, BB1 is able to implement and integrate other types of goal-directed reasoning because it maintains its agenda in two parts: triggered "KSIs" and executable KSIs/KSARs. This makes it possible to include forms of precondition-action backchaining and subgoaling [Johnson89]. Note though, that BB1 can not fully implement subgoaling because it does not leave an explicit record of the relationships between subgoals it needs/desires to pursue (as in the goal-directed blackboard architecture discussed above). Subgoaling is accomplished through special processing of goals that attempts to directly identify relevant actions. In this sense, it is somewhat similar to the way that HSII's *predict* and *verify* KSs were able to accomplish some limited subgoaling ability.

BB1 has not focused on issues involved in modeling the state of problem solving in order to try to understand the relationships among actions and how actions will affect the goal of satisfying the termination criteria. This was one of the major issues in HSII and in work on the DVMT. The lack of effort on this issue is probably a result of the types of problems to which BB1 has been applied. For example, many assembly-arrangement problems do not involve the same kinds of solution uncertainty and termination concerns as interpretation problems. This is because they can accumulate constraints without the need to pursue alternative solution paths because of inconsistency of partial solutions. In other words, while there is control uncertainty because the system may not know what actions would provide the best constraints (i.e., best prune the search space), the system does not have to deal with the range of solution uncertainty issues as part of the process of making control decisions since partial solutions are rarely inconsistent (i.e., they can almost always be combined, producing tighter constraints). In addition, once the system constructs a solution that meets the constraints, the system can terminate—so termination is not a major issue either. Of course, this depends on the complexity of the application.[28]

Because control actions must compete with normal domain actions for resources, this makes it possible for the system to decide whether to reason about how to make control decisions (by executing control KSIs) or whether to just act based on its current strategies. To our knowledge, however, the flexibility that BB1 has to reason about how to balance domain and control actions has not been exploited. Typically, control KSs are given higher priority than domain actions and

[28] [Hayes-Roth86a] states the following in the context of the PROTEAN system: "However, when reasoning about all constituent structures in larger proteins with all available constraints, PROTEAN will need a new strategy. It will have to reason about multiple partial solutions and their relationships to one another. It will have to sequence its constraint-satisfaction operations intelligently to avoid a computationally intractable explosion of hypothesized structures. It will have to reason about alternative protein conformations corresponding to constraints that are not satisfied simultaneously."

are thus scheduled first. In part this is probably because there is no good theory that would allow BB1 to judge the relative value of domain versus control actions.[29]

BB1's agenda/scheduler mechanism has two characteristics that have the potential for making BB1 very inefficient for particular classes of problems. First, the ability to dynamically change ratings functions makes it more likely that KSIs will have to be repeatedly re-rated as compared with a classic blackboard framework. Second, both triggered and executable KSIs (KSARs) must have their preconditions and obviation conditions repeatedly rechecked. This has not been a problem until recently, since BB1 has not been used for applications in which large amounts of data trigger numerous KSIs that remain on the agenda throughout problem solving—as in many interpretation problems. Recent work has addressed these efficiency issues as well as the need for timely/responsive control in real-time applications. In [Collinot90] the standard "best-next" BB1 control cycle is replaced with a "satisficing" control cycle which can limit the time spent reasoning about control decisions and respond to real-time deadlines by not considering (rating) all of the possible actions. In [Hayes-Roth89], not only is this satisficing control cycle used, but intelligent preprocessing of sensor data [Washington89] is used to "shield the reasoning system from data overload."

6.5 Model-Based Incremental Planning

Work by Durfee and Lesser [Durfee86, Durfee87, Durfee88, Durfee91] on an incremental planner for blackboard-based interpretation systems extends the capabilities of the goal-directed blackboard framework (Section 6.3). An abstract model of where there are potential solutions in the search space, the relationships among these potential solutions, and the likely difficulty of constructing them is used to drive the planner. This model allows the system to develop long-term, high-level goals for meeting the termination criteria. The planner makes it possible to coordinate sequences of actions in order to meet the long-term goals. The system was implemented in the Distributed Vehicle Monitoring Testbed and is built on top of the DVMT's goal-directed blackboard framework. The architecture of this planner-based blackboard system is similar to that of the goal-directed blackboard architecture (Figure 8) except for the addition of the planning mechanism which controls the goal processor.

The planner has two components: a clustering mechanism that creates an abstract model of the solution space and a planning mechanism that uses this model to develop plans for working toward "long-term goals" of developing vehicle tracks. The clustering mechanism forms high-level models by examining several types of relationships among the data: temporal relations, spatial relations, event-class relations, and so on. It provides the system with an abstract view of the information about the potential solutions (i.e., vehicle tracks) and identifies those that may be alternatives because they may require the same supporting data. The modeling process effectively represents a rough pass at solving the problem and can be viewed as a kind of *approximate processing* technique [Decker91b]. In this respect, clustering is related to HSII's use of the *word-seq* KS (and

[29] This was the same issue that caused the elimination of the early HSII mechanism that scheduled KS preconditions as well as KS actions (Section 5.7).

associated processing strategy). One key difference is that clustering does not work via the (domain) blackboard, but by directly providing information to the planner.

The planner makes use of this model to create "long-term goals" for creating the potential solutions (tracks) and for resolving uncertainty about the potential alternatives. It does this by first identifying and ordering "intermediate goals." These goals represent the needs for the system to construct particular types of hypotheses that will be needed to create and extend solution hypotheses. The intermediate goals are ordered using domain-independent heuristics. For example, the planner will favor working on those intermediate goals first that best discriminate between alternative solutions since this may eliminate the need to work on data representing alternative potential solutions. Thus, the clustering process provides the system with a view of its search space that the planner exploits to provide an abstract pass at order actions via the ordering of the intermediate goals.

After the intermediate goals have been ordered, the planner determines how to achieve them by identifying appropriate sequences of KSIs on the agenda. This detailed level of planning is done incrementally as each "intermediate goal" actually has to be achieved. KSIs to achieve the intermediate goals are identified and ordered based on models of the KSs that provide rough estimates of their costs and the characteristics of their output. Here the planner controls the subgoaling process in the underlying goal-directed blackboard system. Only subgoals necessary to carry out the plan will be created.

The planning system developed for the DVMT is not general-purpose. Both the solution space modeling process and the planning process are highly specialized to vehicle monitoring and vehicle tracking. This can be seen by the fact that the planner has a very limited notion of goals: goals only refer to creating and extending the *potential tracks* identified by the modeling process. A major reason for this lack of generality may be the simplistic evidence model that was used in the DVMT. The representation of evidence effectively limited the system to resolving its uncertainty by evidence aggregation—i.e., extending vehicle tracks. In fact, a key reason for the performance advantages of the planner-based system over the basic DVMT is that it provides the ability to do some differential diagnosis. This is because the modeling process identifies the potential alternative tracks and, thus, the potential alternative vehicles. Providing differential diagnosis in this way has one major drawback: all of the reasoning about alternatives is done by the control component rather than the evidential reasoning component. In other words, there is no representation within the hypothesis structures of the evidential relationships between alternatives.

6.6 The Channelized, Parameterized Blackboard Architecture

The channelized, parameterized blackboard architecture [Decker89, Decker91b] is an extension of the goal-directed blackboard architecture (Section 6.3) in combination with a modified version of the BB1 architecture (Section 6.4). The impetus for the architecture was research into real-time problem solving in the Distributed Vehicle Monitoring Testbed. In order to use a blackboard system to solve problems involving real-time deadlines, the blackboard execution loop must be efficient (because there is the potential for a large number of KSIs if approximate processing methods are

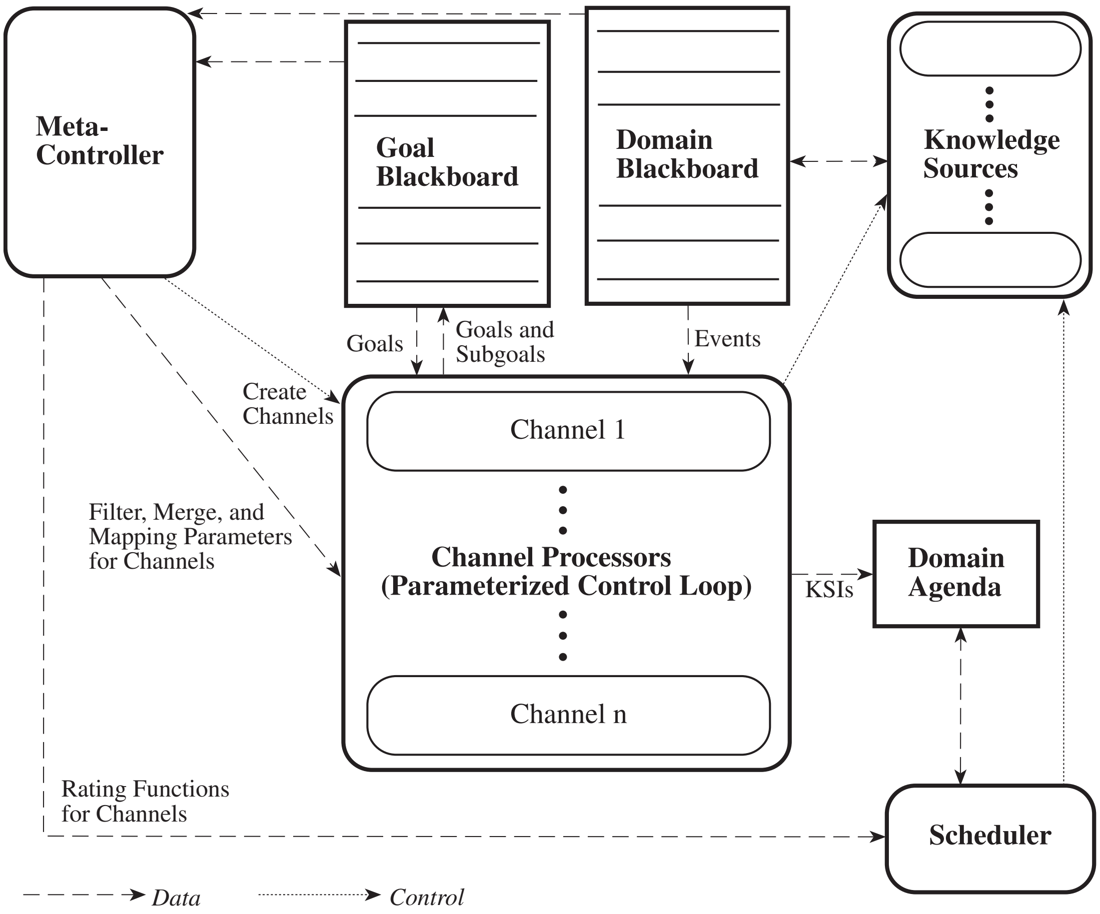

Figure 11: The channelized, parameterized blackboard architecture.
Compared to the goal-directed blackboard architecture (Figure 8), the goal-directed architecture adds the *meta-controller* (see Figure 12) and the *channel processors* replace the goal processor. The channel processors execute the parameterized control loop (Figure 13) that is an extension of the control loop used in the goal-directed blackboard architecture. In addition to the *hypothesis-to-goal*, *goal-to-subgoal,* and *goal-to-KS* mappings done by the goal processor, the channel processors do *hypothesis filtering*, *goal filtering*, goal merging, and KSI merging. The meta-controller is responsible for creating channels; setting the filtering, merging, and mapping parameters for the channels; and determining the rating functions for KSIs associated with each channel.

considered) and predictable (in terms of the time that will be spent satisfying goals and making control decisions). In addition, there must be some representation of both the current and future goals to be pursued (in order to select and schedule actions that are appropriate to satisfy the goals and meet real-time deadlines). There are three main elements of the channelized, parameterized blackboard architecture: the *parameterized control loop*, a *meta-controller*, and *multiple execution*

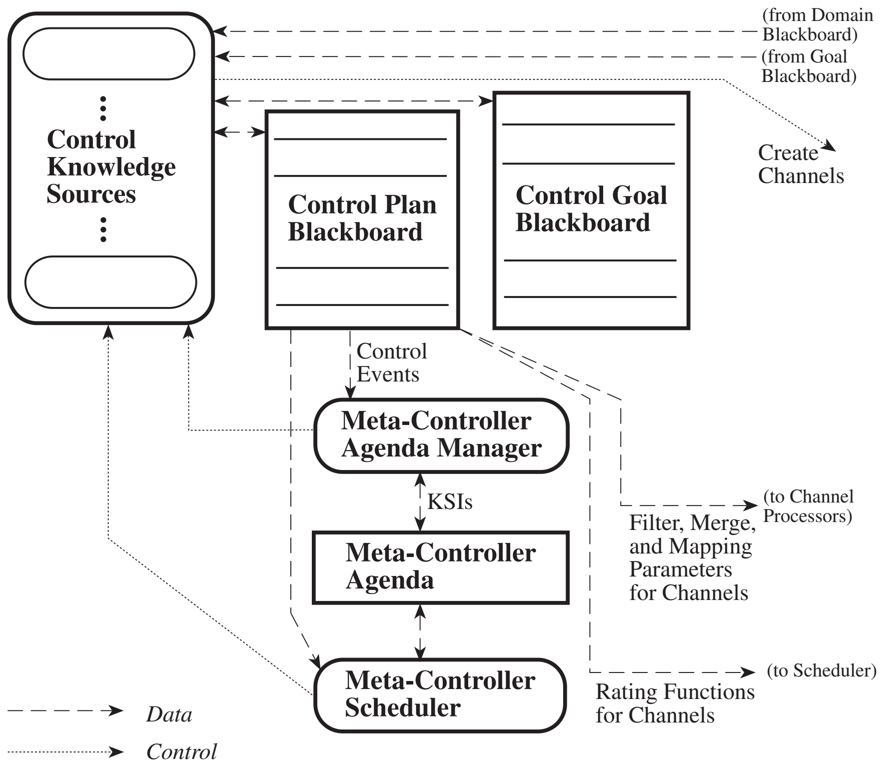

Figure 12: The meta-controller of the channelized, parameterized blackboard architecture.
The meta-controller is based on the BB1 architecture (Figure 10). It extends BB1 by adding a *control goal blackboard* that maintains explicit goals which drive the control planning process and by allowing its control plan *foci* to set the control loop parameters for the channels as well as select *heuristics*. Unlike BB1, the control loop for the meta-controller (which selects control KSs) is completely separate from the (channelized, parameterized) control loop that selects domain KSs.

channels.[30] The overall architecture is shown in Figure 11 and the architecture of the BB1-based meta-controller is shown in Figure 12.

The parameterized control loop extends the control cycle of the goal-directed blackboard architecture so that more control can be exercised over what KSIs appear on the agenda. Comparing the basic parameterized control loop shown in Figure 13 with the goal-directed blackboard control loop (Figure 9), the major additions are the *filter* and *merge* steps: filter hypotheses, filter goals, merge goals, and merge KSIs. Filtering is done before the (hypothesis-to-goal and goal-to-KS) mapping steps to limit the hypotheses and goals/subgoals that are pursued through each mapping. For example, hypothesis filtering might allow through only hypotheses of certain classes or

[30] For real-time problem solving, the system also includes a *real-time scheduler* that augments the standard blackboard agenda scheduler. It schedules groups of KSIs that are associated with particular tasks (goals) and monitors their progress to ensure that all deadlines are met. The ability to reason about the time actions are expected to take is critical for real-time problem solving since the value of an action is strongly dependent on when its results become available. The channelized, parameterized architecture version of the DVMT was one of the first blackboard systems to make (and use) explicit predictions about the time necessary to execute actions and satisfy goals. The real-time scheduler will not be discussed here since we are not discussing real-time problem solving in any detail in this paper (though real-time problem solving is discussed as an emerging line of research in Section 7.)

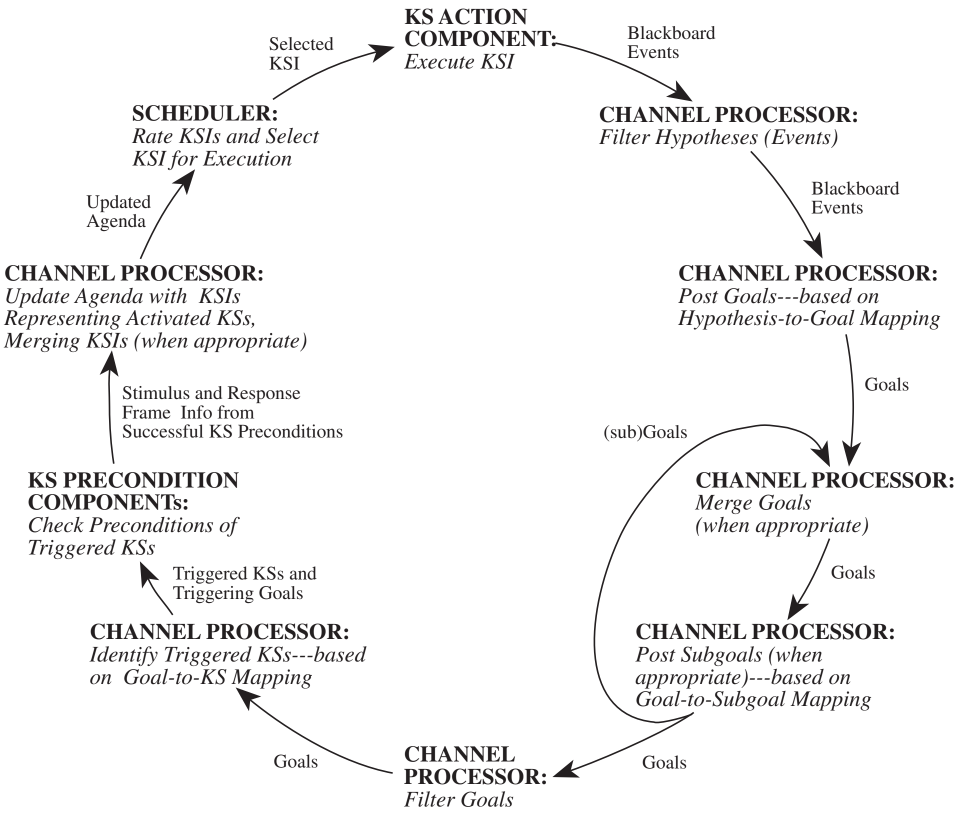

Figure 13: The parameterized control loop for the channelized, parameterized blackboard architecture.

hypotheses associated with particular blackboard regions. Merging also serves to limit the KSIs that are created by combining goal and KSI units that lead to duplication of effort.

The filtering, merging, and mapping are parameterized so that the overall behavior of the loop can be dynamically modified. The parameters are set by the meta-controller, which is based on an extended version of the BB1 control planning architecture. In BB1, the selection of actions is affected only by the active rating functions (the heuristics associated with the active control plan foci). The meta-controller control plan foci not only have associated rating functions, they can also modify the parameters for the parameterized control loop.[31] The meta-controller planning cycle is run to quiescence (all possible control reasoning is done) after each (domain) KSI execution, to set the parameters for the parameterized control loop. Note that the meta-controller execution cycle is separate from the the parameterized control loop cycle, so the selection and execution of control KSs is separate from the selection and execution of domain KSs here—unlike BB1.

Another way in which the meta-controller extends the BB1 architecture is its use of explicit *control goals* that are maintained on a *control goal blackboard*. These control goals drive the control planning process: the top-level, system goal stimulates the creation an appropriate strategy control plan that in turn posts appropriate (sub)goals that then stimulate the creation of substrategies and

[31] This idea is similar to the BB1 extension in which a preprocessor is controlled to limit the amount of data that makes it to the blackboard (Section 6.4).

so on. Control goals make explicit the purpose of the strategies and foci. They also allow *expected* high-level goals to be represented so that the system can estimate the resource requirements of the current and future steps of each task. The BB1-based planner was modified so that it was not strictly incremental, but would post expected future sub-strategies/foci ("goals") of the high-level strategies. Note that the control goals are completely separate from the (domain) goals that were part of the goal-directed blackboard architecture and that are still used in the parameterized control loop. Control goals represent high-level, long-term goals of the system while domain goals represent detailed and immediate goals that are more directly associated with possible actions.

While the architecture described so far has the power to control the KSIs that make it onto the agenda, it has one serious limitation. Systems often need to concurrently pursue multiple high-level goals—i.e., they have multiple tasks whose problem-solving actions must be interleaved. For example, a vehicle monitoring system may have to track multiple vehicles simultaneously and must interleave work on each of the vehicles in order not to miss important data. The problem is that each of these tasks may require different settings of the control loop parameters for optimal problem solving. With a single control loop, either tasks can not be pursued concurrently or else sub-optimal parameter settings must be used to cover all the tasks, reducing the effectiveness of the parameterized architecture. To solve this problem, the parameterized control loop architecture was extended through the notion of *multiple execution channels*: multiple sets of control loop parameter settings that can be associated with individual tasks. This allows the same level of control over the processing for each task, regardless of how many tasks are being concurrently pursued. Channels are created dynamically as needed by the meta-controller, using separate channels for problem solving associated with (mainly) independent goals (typically, a channel is created for each active control plan focus). For example, separate channels would be created for tracking each of several distinct vehicles. The decision about which KSI to execute next is still made by the scheduler: the KSIs from all of the active channels are placed onto a single agenda and rated using rating functions provided by the channel with which each KSI is associated. This may result in actions from different channels being interleaved.

The channelized, parameterized blackboard architecture allows the system to exercise control over the KSIs that make it onto the agenda. This can lead to more efficient blackboard control if the KSIs can be limited to truly useful and desirable actions. It also makes execution of tasks more predictable since it more tightly constrains the data that will be processed for task, the type of problem-solving knowledge to be used, etc., and so makes timing and output quality more predictable. In part, this can be viewed as an issue of controlling the amount of opportunism (versus predictability) that is permitted with each task. It can also be viewed as enabling more explicit and direct implementation of sophisticated control strategies since the possible actions on the agenda can be limited to those relevant to the selected control plans. By contrast, BB1 (which also has explicit control plans) may have to have more complex rating functions since they must discriminate among larger numbers of possible actions. Finally, while the architecture makes the execution of tasks more predictable and the control cycle more efficient, it did not address the issues of having a predictable control cycle—i.e., there were not fixed deadlines for making control decisions as in the satisficing control cycle extension of BB1.

6.7 ATOME: Hybrid Multistage Control

In the ATOME blackboard framework [Laasri89], selection of domain KSs to be executed is accomplished with a control mechanism derived from that of CRYSALIS (Section 6.2). The designers of ATOME were concerned with improving the efficiency of blackboard systems without sacrificing the flexibility of HSII-like control mechanisms. They felt that the hierarchical control architecture of CRYSALIS could improve the efficiency of making control decisions, but that it compromised the ability to be highly opportunistic when this was appropriate. As a result, ATOME extends the CRYSALIS architecture by providing the ability to solve subproblems (tasks) either by directly identifying KSs to be executed or by applying an agenda-based mechanism for selecting KSs. The decision about which type of task control to use is made dynamically, according to which approach is judged to be most appropriate given the current state of problem solving.[32] Because this allows the use of both hierarchical and agenda-based control and because the decision about the appropriate type of control is made for successive stages of problem solving (i.e., for each new task), ATOME is referred to as a *hybrid multistage* control architecture. The ATOME architecture is similar to that of CRYSALIS.

Just as in CRYSALIS, ATOME uses two levels of "control KSs" and represents its KSs as sets of rules. The strategy KS selects the task KSs to be executed while task KSs select the domain KSs[33] to be executed. In the strategy KS, rule LHSs are conditions on *blackboard summaries* and RHSs list one or more task KSs to be *sequentially* executed. Blackboard summaries are abstractions of the state of the (domain) blackboard that attempt to give a global view of the relative quality of developing solutions. Blackboard summaries are equivalent to CRYSALIS' features list and are analogous to HSII's focus-of-control database.

In task KSs, rule LHSs refer to blackboard events and RHSs identify domain KSs to be executed. Selected domain KSs may be executed either sequentially or opportunistically depending on the *type* of the task KS. There are three types of task KSs: event-driven, rule-driven, and opportunistic. In event and rule-driven tasks, domain KSs are directly identified and sequentially executed.[34] In opportunistic tasks, the control is basically that of a standard agenda-based approach: events trigger KSs which wait for their preconditions to become satisfied, these executable KSs post KSIs on an agenda, and a scheduling function (instantiated by the strategy unit) selects the KSI to execute next.

[32] It is not clear what factors ATOME's designers have found to be relevant in making this decision.

[33] Domain/problem-solving KSs are referred to as *specialist* KSs in ATOME.

[34] Event-driven and rule-driven tasks differ in how their rules examine and are triggered by the events list. In event-driven tasks, the "highest-priority" (i.e., most recent) event is taken and the set of task rules scanned to find a relevant rule to fire. In rule-driven tasks, the set of rules is scanned looking for the first rule to fire (and rule LHSs may search the entire events list). CRYSALIS also allowed flexibility in specifying how to search for valid rules. Note that this choice is independent of the choice about the mode in which KS rules are fired—i.e., single hit, multiple hit, or cyclically.

From the general description of ATOME, it might appear that it provides a much cleaner mechanism to implement the kinds of special-purpose strategies that HSII implemented through large-grained KSs (e.g., working on all the data in a data-directed fashion to the word level) or through special scheduling mechanisms (e.g., always scheduling the *predict* and *verify* KSs together). In other words, it seems that ATOME could be used to develop a blackboard system that would typically work highly opportunistically (through an agenda), but would sometimes directly schedule sequences of events to accomplish particular tasks. However, because ATOME tasks can not be suspended or interleaved and because ATOME agendas are local to particular tasks, this does not appear possible.

6.8 CASSANDRA: Distributed Blackboard Control

The CASSANDRA architecture [Craig89] is a significant modification of the blackboard model rather than an alternative blackboard control architecture. It is discussed here because it was originally presented as addressing perceived limitations of the blackboard model and because certain aspects of the architecture have profound implications for control.[35] [Craig89] states that the blackboard model does not provide sufficient modularity or flexibility for many problems. To achieve greater overall modularity, in CASSANDRA the database and control mechanisms (as well as the problem-solving knowledge) are modularly structured. The principal component of CASSANDRA is the *level manager* (LM). Each LM includes its own local database of partial solutions, its own set of KSs, and its own local control mechanism. Because LMs are intended to correspond to the abstraction levels in a conventional blackboard system, the local database of an LM is essentially a single level of a standard blackboard database and the local KSs are those KSs that would be triggered by events associated with that level (it is not clear how KSs that may be triggered by events at multiple levels are handled). *Communication channels* define the relations among the LMs (levels): information can be transferred between two LMs only by sending messages over a channel that links the LMs. Thus, if a KS is triggered by an event in its local database but its precondition requires information about another level, the local LM must send a request to the LM associated with the other level (over a channel between the LMs) and the KS must wait to receive a response before completing its precondition check.

Control is more modular in CASSANDRA than in conventional blackboard architectures since control decisions are made locally in the LMs, each of which can have control strategies that are appropriate for its particular level. This increased control modularity is not without cost, however, since CASSANDRA's distributed architecture complicates the process of achieving effective global

[35] In private communications, CASSANDRA's developer has stated that he "always considered CASSANDRA to be more of a distributed architecture than a blackboard derivative" and that he no longer considers "any connection with the blackboard architecture as being particularly relevant." An investigation of blackboard-based architectures for distributed problem solving is beyond the scope of this paper (though see Section 7). Nevertheless, viewing CASSANDRA as an alternative *centralized* problem-solving architecture serves to illustrate the trade-offs between increased modularity and effective global control.

control.[36] To understand why this is the case, it is useful to recognize that the architecture is really a special case of one class of distributed problem-solving architectures [Durfee89]. As such, it inherits the issues and the open problems of such systems; in particular, the problem of achieving *global coherence* through local control [Lesser91a]. In fact, [Craig89] states that "global control in completely distributed problem solving systems is a difficult and open problem, so the reader is warned not to expect any solutions." The two test systems cited in [Craig89] that were implemented with CASSANDRA used very simple global control strategies to sequence the execution of the LMs (they used very simple local control strategies as well). By making global, goal-directed control more difficult, CASSANDRA goes against the direction of evolution shown in the other blackboard control architectures presented here. While there are many applications for which distributed problem-solving is appropriate, in most (centralized) blackboard-based systems the blackboard levels (and their associated operations) typically have too small a grain size to make the overhead of distributed problem solving worthwhile.[37]

Control modularity is a worthwhile goal, but it does not require a distributed problem-solving architecture. Hierarchical blackboard control mechanisms (e.g., CRYSALIS and ATOME) can also support the use of different strategies for different levels, and planners (e.g., RESUN) can support control strategies that are specific to every plan and subgoal. In fact, it is not at all clear that it is appropriate to associate control strategies with blackboard levels; control strategies are typically concerned with how to best achieve system goals and subgoals, which are often not directly connected with one particular blackboard level. Another problem with the CASSANDRA architecture is its lack of support for opportunistic problem solving. CASSANDRA does not support opportunistic problem solving largely because it does not support global control, and opportunism involves selecting the action with the maximum (expected) global value (which is difficult to judge from the limited perspective of each LM). In addition, CASSANDRA makes opportunistic control difficult because the relationships among the solution paths are not easily understood since the system no longer has an integrated view of its hypotheses.[38] Furthermore, because preconditions and actions involving multiple levels cannot be directly executed, there can be arbitrary (possibly infinite) delays in carrying out these operations. This can affect opportunism by limiting the actions that appear to be possible at any point in time.

[36] Global control in CASSANDRA involves both the selection of the LM to execute next (in a uniprocessor system) plus mechanisms to influence local control decisions to achieve global system goals (e.g., by developing and distributing information that gives the local controllers a more global view of problem solving).

[37] Local controllers (agents) must be able to handle many issues besides action selection: the possibility of deadlock, arbitrary delays when getting information from other levels, the choice of whether to take local actions or respond to communications from other LMs, and so on.

[38] CASSANDRA is actually quite inflexible because of its insistence that the levels be independent. Many applications (e.g., sensor interpretation problems [Carver90a]) have well-defined evidential relationships between the levels. CASSANDRA makes it very costly to "trace links" among hypotheses in order to update credibility ratings or identify hypotheses with certain substructure characteristics. [Craig89] says that there may only be a single LM, but then the CASSANDRA architecture offers little guidance in structuring the single "local" database or in implementing control.

6.9 RESUN: Planning to Resolve Sources of Uncertainty

Among the more recent blackboard control architectures is a blackboard-based interpretation framework that we have developed called RESUN [Carver90a, Carver90b, Carver93]. The main goals in developing the RESUN system were to extend the range of methods that interpretation systems could use to resolve uncertainty and to support the implementation of sophisticated control strategies—i.e., strategies that involve large amounts of highly context-specific knowledge [Carver93]. In particular, we wanted to be able to make extensive use of direct methods for resolving hypothesis uncertainty such as differential diagnosis (see Section 3.4). To accomplish these goals, the conventional blackboard representation of hypotheses was extended and the agenda mechanism was abandoned for an incremental control planner. However, the opportunistic capabilities of agenda-based blackboard systems were maintained by augmenting the planner with a *refocusing* mechanism, which allows planner focusing decisions to be reconsidered in response to changed problem-solving situations.

In order to use direct methods for resolving uncertainty, a system must be able to understand the *reasons* why its hypotheses are uncertain. RESUN's representation of hypotheses maintains detailed information about the reasons hypotheses are uncertain and about the evidential relations between alternative hypotheses. The key feature of the representation is its use of explicit, symbolic statements of the *sources of uncertainty* in the evidence for interpretation hypotheses. The symbolic source of uncertainty statements (SOUs) are based on a model of interpretation as a process of *abductive inference*. For example, an interpretation hypothesis may be uncertain because its supporting data might have alternative explanations or it may be uncertain because its evidence is incomplete. As interpretation inferences are made in RESUN, SOUs are attached to the hypotheses to represent their current sources of uncertainty.

Control of all blackboard actions is handled by a planning mechanism that was designed to maintain the benefits of planners (e.g., coordination of sequences of actions), while being sufficiently opportunistic and reactive to deal with the uncertainty and the dynamically changing situations that are encountered in many interpretation applications. The basic control loop of the RESUN planner is shown in Figure 14. In addition to the refocusing mechanism (discussed below), the RESUN planner has a number of key features: it is *script-based* (it uses plan schemas), it interleaves planning and execution (planning is *incremental*), plans can invoke explicit *information gathering actions* to examine the current state of problem solving, and context-specific focusing knowledge is applied to control the planner's search. Note that unlike BB1 (Section 6.4), the RESUN planner does detailed planning to directly identify the actions to be carried out. Actions are represented as *primitive* control plans and a RESUN application would include primitive control plan schemas that correspond to the knowledge sources in a conventional blackboard system.

In RESUN, sensor interpretation is modeled as an incremental process of gathering evidence to resolve *particular sources of uncertainty* in the interpretation hypotheses. Control plans invoke information gathering actions that examine hypotheses to determine the current SOUs associated with the hypotheses. The SOU information is used to post goals to resolve uncertainty. These goals drive the system to select methods (plans) that are appropriate for the current sources of

```
Initialize current-focus-points to the top-level control plan instance
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)
  case on type-of(focus):
    plan instance Focus on multiple-valued variable bindings to select plan instances.
                  Expand selected plan instances to next subgoals.
                  Focus on subgoals to select subgoals.
                  Match subgoals to control plans.
                  Focus on matching plans to select new plan instances for next-focus-points.
    primitive     Execute 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 in-progress plan instance.
```

Figure 14: The basic RESUN control planning loop.

uncertainty in the hypotheses. The selected methods are refined to identify and execute actions that implement the methods. This general process is repeated until the termination criteria are satisfied.

Termination is an important issue for many interpretation systems and was a key concern in the design of RESUN. For example, in an application like vehicle monitoring, "the solution" can have an indeterminate number of (mainly) independent components (vehicle hypotheses). This means that a vehicle monitoring system must not only resolve uncertainty about the correctness of hypotheses representing potential vehicles, it must also be sufficiently certain that there are not additional—as yet unidentified—vehicles. Furthermore, it will often have to do this without being able to examine all of the data or all of the potential interpretations of the data. In RESUN, the overall interpretation process is driven by a high-level model of the state of problem solving that represents the need to resolve uncertainty in order to meet the termination criteria. For example, the high-level model identifies potential solution hypotheses that are not sufficiently certain to meet the termination criteria, regions of interest that have not been examined thoroughly enough to exclude the possibility of additional solution components, and data that has not yet been ruled out (as supporting a solution hypothesis). The entries in this model effectively represent unsatisfied system goals. They are used to post high-level goals that drive the actions of the planner. Thus, RESUN's actions are explicitly directed by the system termination criteria.

The main innovation in our planner is its *refocusing mechanism*. This mechanism can be used to postpone focusing decisions when there is insufficient information about the particular situation to make a conclusive choice. For example, the planner may find that there are two methods (plans) that may be used to satisfy some (sub)goal that it is pursuing. However, the planner may not be able to immediately decide which method has the higher expected value since that can depend on how each method will actually be implemented given the current situation. Refocusing allows the planner to postpone the decision between the two plans, partially refine the plans based on the current situation, and then reconsider this decision when there is sufficient information to judge the relative value.

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.

Because of the search process inherent in constructing plans, a planning-based approach to control results in a view of control as both a search for the correct solutions and a search for the best methods to use to determine the solutions. Thus, what we have effectively done is to extend the (explicit) search paradigm from just finding the solutions to also finding the control decision (finding how to find the solutions). This comes out clearly in the postponed focusing example mentioned above. In addition, the method search view of control leads us to view opportunism as arising naturally from explicit method search instead of as a 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.

Another result of planning-based control is that the process of making control decisions implicitly involves search. Thus, instead of trying to engineer complex ratings functions that directly select the best KSI (action) to execute next, our planning and refocusing mechanism allows us to make a series of less complex search decisions to select the next action. This approach is exactly analogous to the use of search processes in solving complex AI problems [Rich91]. Making control decisions via search is useful for blackboard-based problem solving given the difficulty of estimating the absolute expected value of possible actions because it facilitates comparative evaluations. In effect, each planner focusing decision partitions the agenda so that the number of possible actions considered by lower level decisions is greatly reduced.

It may also be of interest to note that the RESUN notion of opportunism differs somewhat from that in BB1. In BB1, control KSs are allowed to opportunistically post new "subgoals." 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 planner is intended to embody the overall system 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.

RESUN is the most goal-directed system presented in this paper since it is based on a planner that does detailed planning and produces explicit subgoals. In fact, RESUN takes a very different approach to control from the other blackboard architectures discussed here. Most of the other architectures have started with an agenda mechanism and then added goal-directed control mechanisms to modulate the data-directed behavior of the basic agenda approach. Those that did not use a (global) agenda (CRYSALIS and CASSANDRA) had limited capabilities to support opportunistic problem solving. RESUN starts with a goal-directed mechanism and uses refocusing to modulate the behavior of the planner to achieve the data-directed control necessary for opportunism.

One of the 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: what the purpose of the decision is in terms of the goals and subgoals to which it pertains, what the relationships among the various decision alternatives are, etc. Having detailed and explicit context for each decision facilitates the implementation of sophisticated control strategies. This is because context information can be used to structure the control knowledge into *modular*, context-specific heuristics. In RESUN, focusing heuristics are associated with particular plans, subgoals, or plan variables and they are free to examine the instantiated planning structure. Thus, while the RESUN architecture does not maintain the modularity between knowledge sources and control as in a classic blackboard system, it does make the control knowledge very modular. This makes it easier to encode and modify complex strategy knowledge than with conventional blackboard scheduler functions.

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, many interpretation problems involve passive sensors that continuously generate data. If all this data is automatically placed onto the blackboard to trigger the set of possible actions that can process the data, the blackboard and the agenda can quickly become overloaded. One solution is to use a controllable preprocessor to limit the data that actually gets placed onto the blackboard as in BB1. A highly goal-directed system like RESUN can directly control the amount of data that undergoes (any) processing. RESUN applications have been written that use goal-directed strategies to limit the data that must be examined to meet the termination criteria. Interpretation applications may also involve active sensors whose operation may be controlled by the interpretation system. Because RESUN produces detailed goals, it can actively direct data gathering by invoking sensors when appropriate and by selecting sensor parameters that can best meet its goals.

7 Emerging Directions

In the previous section we examined a number of important blackboard control architectures. Because the field is so wide, however, we do not have space to discuss all developments. In this section, we will briefly mention other lines of research in blackboard control, paying particular attention to recently emerging directions that we believe will be important in the future development of blackboard systems.

One area of research that we have not discussed is parallel blackboard architectures: blackboard systems that allow "the concurrent execution of knowledge sources (KSs) and control components in a shared address space" [Corkill89]. The blackboard model was developed with parallel execution of KSs using multiprocessor machines in mind [Lesser75] and there were early simulations of parallel HSII systems [Fennell77]. This direction has not been widely pursued until recently, perhaps because of concern with the kinds of issues we have discussed here and the lack of wide availability of multiprocessor machines. Much of the early research on parallel blackboard architectures focused on parallel processing of multiple data streams in which there were very weak interactions among the actions associated with each stream. More recent work has begun to deal with tasks that involve significant search and in which parallel activities interact (e.g., [Decker91a]). This complicates the control problem and has led to research on control heuristics that taken into account the possibility of interdependencies among parallel KS executions when scheduling activities on the agenda. A number of recent papers on parallel blackboard architectures and a comprehensive overview can be found in, respectively, [Jagannathan89] and [Velthuijsen92].

Another area of research that we have not discussed is distributed blackboard architectures: systems that involve problem solving via "communication of blackboard data among autonomous blackboard subsystems" [Corkill89]. This has been an area of considerable research and in fact has directly motivated some of the control architectures previously discussed. The blackboard architecture is appropriate for agents in cooperative distributed problem-solving (CDPS) systems [Lesser80, Durfee89] because the blackboard model supports asynchronous and opportunistic problem solving involving uncertain and incomplete information. Research in CDPS using blackboard-based agents has also shown the importance of developing sophisticated blackboard (agent) control architectures: "...the better the local control strategy is in understanding its own search space the easier it is to construct meta-information [that can be used to coordinate agent activities]...local control must [in turn] have a certain level of sophistication to exploit the meta-information [received from other agents]..." [Lesser91a]. Material on distributed blackboard systems can be found in [Bond88, Velthuijsen92] and a distributed version of RESUN (Section 6.9) is described in [Carver91b].

One of the most important recent areas of research in blackboard systems concerns the use of blackboard systems for real-time problem solving. As blackboard systems are applied to a wider range of real-world problems, they must deal with hard real-time constraints imposed by the environment and they must interact with larger systems within which they are embedded. The major theme in this work is that systems must be able to explicitly consider computation time and other resources required by potential actions when making control decisions. This means that systems must be able to predict resource usage and reason about trade-offs between the cost and the effectiveness of potential domain and control actions. For example, real-time systems may have to select *approximate methods* that do not provide the best possible solution because of hard real-time constraints [Decker91b]. The determination of what approximations (if any) are necessary may require a system to estimate whether there are sufficient resources to meet future deadlines by generating a schedule based on its predicted tasks [Garvey93]. Real-time systems may also have to consider the resources involved in making control decisions. For example, the satisficing control cycle for BB1 [Hayes-Roth89] does not necessarily result in the selection of the best possible action, but it does guarantee selection of the best possible action given the resources (time) available to make the decision to respond. Embedded systems must be able to respond to changing goals imposed from outside. A number of recent papers on real-time blackboard problem solving can be found in [Jagannathan89]. Further, real-time control issues are closely related to the very important meta-level control issue of how to dynamically balance domain actions against control actions.

Another issue that has received increased attention is expanding the set of methods available for blackboard systems to respond to uncertainty and inconsistency. As we discussed, the standard blackboard strategy for resolving uncertainty about potential solutions is hypothesize and test. By contrast, when the sources of uncertainty—e.g., inconsistencies—can be determined, the "constraint satisfaction" process can be directed toward gathering constraints that can immediately resolve the inconsistency. For example, we saw this approach in the RESUN interpretation framework (Section 6.9) where extensive use could be made of differential diagnosis methods to resolve interpretation uncertainty. The RESUN framework is also the basis for the IPUS signal understanding system [Lesser91b]. In IPUS inconsistencies among high-level models of the data can drive the system to reprocess the data using alternative signal processing algorithms so that it can reformulate its view of the data. Explicit resolution of uncertainty can also be seen in recent work on negotiated search [Lander92] in which agents recognize conflicts represented by the situation on the blackboard and negotiate directly with each other to resolve the conflicts.

The basic model for control in IPUS is discrepancy detection, diagnosis to ascertain the reason for the discrepancy, followed by replanning to adjust system parameters to gather evidence to resolve the discrepancy. We see this as a powerful meta-level control strategy, which can be used to adaptively adjust the control strategies of a system to the particular characteristics of problem situations. Though originally formulated in [Hudlicka84], to date such an approach has been used only in IPUS. Likewise, to our knowledge there has been little work on learning control strategies (e.g., [Hayes-Roth85b]). As systems become more complex and are applied to long-term (continuous) tasks, we expect that system control strategies must be able to adapt to problem characteristics.

Another emerging direction in blackboard systems research is the use of high-level task-specific languages. One example of this is the Accord language for arrangement-assembly tasks [Hayes-Roth86b, Hayes-Roth87]. With Accord the system-builder can specify both control decisions and domain actions in a high-level language that is specific to the task of arrangement-assembly, but not specific to any particular domain. This is useful because it facilitates the sharing of control strategies among diverse applications that share similar task areas. Because the control plans are specified in a high-level language they can be adapted to different domains simply by defining the appropriate terms for each particular domain. In fact, some work has been done investigating the automated learning of control plans by analogy to existing control plans for similar tasks in different domains [Johnson88]. Another useful effect of high-level languages is that they improve the explanation capability of a system. Explanations of why actions were taken can be provided in a high-level English-like language, rather than in terms of low-level domain-specific heuristics.

Finally, we feel that it will be important to more rigorously evaluate the effectiveness of available blackboard control architectures and to relate the conclusions to specific characteristics of application tasks. While there has been some experimental evaluation of blackboard architectures (e.g., [Garvey87, Hewett92, Lesser77a]), most of this work has involved a single implementation and/or a single task so it is difficult to draw general conclusions. This situation may change as a result of recent experimental work that uses abstract simulations [Decker91c] and the development of formal models that can characterize the complex search spaces that can occur in blackboard-based problem solving [Whitehair92].

In summary, these emerging trends are all based on being able to: explicitly represent and symbolic reasoning about the higher level goals of the system, resources requirements of potential actions and the availability of resources in making control decisions, use both simple and complex abstract models to analyze the state of problem solving, and fully exploit the integrated view of problem solving afforded by the blackboard.

8 Conclusion

In this paper we have explored the evolution of control in blackboard systems. Though we have not emphasized the more general applicability of these ideas to other problem-solving paradigms, we still feel that the issues addressed here are valuable for AI application domains involving complex multi-dimensional search, in which control knowledge is as important to successful problem solving as domain knowledge. The discussion took place within the context of early work on the Hearsay-II speech understanding system. The appeal of the HSII model was that it made it possible to have great flexibility in choosing problem-solving activities. On the other hand, it can be very difficult to exploit this flexibility using the basic HSII architecture. The difficulty arises, in part, from many of the same features that are responsible for the blackboard model's flexibility—e.g., its ability to represent multiple competing, cooperating and independent lines of reasoning at different levels of abstraction. These features complicate the process of accurately estimating the value of possible actions in terms of how the action will reduce the amount of computational effort required to terminate problem solving.

From this perspective, many of the architectural mechanisms that we have examined have been concerned with understanding the long-term and global effects of possible actions relative to meeting the overall goals of the system. We discuss the evolution of blackboard control mechanisms from two main perspectives: one, the development of architectures that allow the specification and application of sophisticated goal-directed control strategies; and the other, the development of abstract and approximate models to assess the non-local effects of actions and to reason about termination. We expect further evolution of these mechanisms to provide the ability to explicitly reason about resource useage, such as the expected execution time of possible actions. This will allow systems to respond to hard real-time deadlines, but also to more accurately estimate the utility of actions and the overhead associated with control activities. Other architectural mechanisms were concerned with efficiency issues.

There still do not exist appropriate models of the blackboard control problem that would allow us to formally analyze the various control architectures and there is a dearth of empirical data that can be used to do comparative analysis. For these reasons, we have approached the analysis of different control architectures from a qualitative perspective and where we have had some insight, tried to relate characteristics of the task domain to the effectiveness of the alternative architectures. It is our hope that in the future these analyses can be more rigorously done using a combination of simulations, empirical data, and formal models. We hope that through such analysis will come a deeper understanding of the relationship between the architectural characteristics of a blackboard system and the key characteristics of the problem domain (i.e., the characteristics of the search space).

In summary, we feel that significant progress has been made in the understanding of how to effectively control blackboard problem solving in complex domains. However, as indicated in the emerging research directions, there is still exciting research to be done.

Acknowledgments

We wish to thank the following people for their thoughtful comments, which greatly improved this paper: Daniel Corkill, Keith Decker, Lee Erman, Alan Garvey, Barbara Hayes-Roth, Frank Klassner, Brigitte Laasri, Hassan Laasri, Susan Lander, and Bob Whitehair.

This work was supported by the Office of Naval Research under contracts N00014-86-K-0764 and N00014-92-J-1450, and by Rome Labs under contract F30602-91-C-0038. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

[Bond88] Alan Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, 1988.

[Carver90a] Norman Carver and Victor Lesser, *Control for Interpretation: Planning to Resolve Sources of Uncertainty*, Technical Report 90-53, Computer and Information Science Department, University of Massachusetts, 1990.

[Carver90b] Norman Carver, *Sophisticated Control for Interpretation: Planning to Resolve Sources of Uncertainty*, Ph.D. Thesis, Computer and Information Science Department, University of Massachusetts, 1990.

[Carver91a] Norman Carver and Victor Lesser, "A New Framework for Sensor Interpretation: Planning to Resolve Sources of Uncertainty," *Proceedings of AAAI-91*, 724–731, 1991.

[Carver91b] Norman Carver, Zarko Cvetanovic, and Victor Lesser, "Sophisticated Cooperation in FA/C Distributed Problem Solving Systems," *Proceedings of AAAI-91*, 191–198, 1991 (also available as Technical Report 91-23, Computer and Information Science Department, University of Massachusetts).

[Carver92] Norman Carver and Victor Lesser, "Blackboard Systems for Knowledge-Based Signal Understanding," in *Symbolic and Knowledge-Based Signal Processing*, Alan Oppenheim and Hamid Nawab, editors, Prentice Hall, 205–250, 1992.

[Carver93] Norman Carver and Victor Lesser, "A Planner for the Control of Problem Solving Systems," to appear in *IEEE Transactions on Systems, Man, and Cybernetics*, Special Issue on Planning, Scheduling and Control, 1993.

[Clancey86a] W. Clancey and C. Bock, *Representing Control Knowledge as Abstract Tasks and Metarules*, Technical Report KSL 85-16, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1986.

[Clancey86b] W. Clancey, "From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons," *AI Magazine*, vol. 7, no. 3, pp. 40–60, 1986.

[Collinot89] Anne Collinot, "Revising the BB1 Basic Control Loop to Control the Behavior of Knowledge Sources," in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence Baum, editors, Academic Press, 27–41, 1989.

[Collinot90] Anne Collinot and Barbara Hayes-Roth, "Real-Time Control of Reasoning: Experiments with Two Control Models," *Proceedings of DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Morgan Kaufmann, 263–270, 1990 (also available as Technical Report KSL 90-17, Knowledge Systems Laboratory, Stanford University, 1990).

[Corkill82a] Daniel Corkill, Victor Lesser, and Eva Hudlicka, "Unifying Data-Directed and Goal-Directed Control: An Example and Experiments," *Proceedings of AAAI-82*, 143–147, 1982.

[Corkill82b] Daniel Corkill, *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*, Ph.D. Thesis and Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, 1982.

[Corkill86] Daniel Corkill, Kevin Gallagher, and Kelly Murray, "GBB: Generic Blackboard Development System," *Proceedings of AAAI-86*, 1008–1014, 1986 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 503–517, 1988).

[Corkill89] Daniel Corkill, "Advanced Architectures: Concurrency and Parallelism," in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence Baum, editors, Academic Press, 77–83, 1989.

[Craig89] Ian Craig, *The CASSANDRA Architecture: Distributed Control in a Blackboard System*, Ellis Horwood, 1989.

[Davis80] Randall Davis, "Meta-Rules: Reasoning about Control," *Artificial Intelligence*, vol. 15, 179–222, 1980.

[Decker89] Keith Decker, Marty Humphrey, and Victor Lesser, *Experimenting with Control in the DVMT*, Technical Report 89-85, Computer and Information Science Department, University of Massachusetts, 1989.

[Decker90] Keith Decker, Victor Lesser, and Robert Whitehair, "Extending a Blackboard Architecture for Approximate Processing," *The Journal of Real-Time Systems*, vol. 2, 47–79, 1990 (also available as Technical Report 89-115, Computer and Information Science Department, University of Massachusetts).

[Decker91a] Keith Decker, Alan Garvey, Marty Humphrey, and Victor Lesser, "Effects of Parallelism on Blackboard System Scheduling," *Proceedings of IJCAI-91*, 15–21, 1991.

[Decker91b] Keith Decker, Alan Garvey, Marty Humphrey, and Victor Lesser, "Real-Time Control of Approximate Processing," to appear *International Journal of Pattern Recognition and Artificial Intelligence*, 1992 (also available as Technical Report 91-50, Computer and Information Science Department, University of Massachusetts).

[Decker91c] Keith Decker and Victor Lesser, *Analyzing a Quantitative Coordination Relationship*, Technical Report 91-83, Computer Science Department, University of Massachusetts, 1991 (to appear in *Group Decision and Negotiation*, 1993.).

[Durfee86] Edmund Durfee and Victor Lesser, "Incremental Planning to Control a Blackboard-Based Problem Solver," *Proceedings of AAAI-86*, 58–64, 1986.

[Durfee87] Edmund Durfee, *A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network*, Ph.D. Thesis and Technical Report 87-84, Department of Computer and Information Science, University of Massachusetts, 1987.

[Durfee88] Edmund Durfee and Victor Lesser, "Incremental Planning to Control a Time-Constrained, Blackboard-Based Problem Solver," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, no. 5, 647–662, 1988.

[Durfee89] Edmund Durfee, Victor Lesser, and Daniel Corkill, "Cooperative Distributed Problem Solving," in *The Handbook of Artificial Intelligence, Volume 4*, Avron Barr, Paul Cohen, and Edward Feigenbaum, editors, Addison Wesley, 85–147, 1989.

[Durfee91] Edmund Durfee and Victor Lesser, "Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 5, 1167–1183, 1991.

[Engelmore79] Robert Engelmore and Allan Terry, "Structure and Function of the CRYSALIS System," *Proceedings of IJCAI-79*, 250–256, 1979.

[Engelmore88] Robert Engelmore and Tony Morgan, editors, *Blackboard Systems*, Addison-Wesley, 1988.

[Erman80] Lee Erman, Frederick Hayes-Roth, Victor Lesser, and D. Raj Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, vol. 12, no. 2, 213–253, 1980 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 31–86, 31–86, 1988).

[Erman81] Lee Erman, Philip London, and Stephen Fickas, "The Design and an Example Use of Hearsay-III," *Proceedings of IJCAI-81*, 409–415, 1981 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 281–295, 1988).

[Fennell77] Richard Fennell and Victor Lesser, "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay-II," *IEEE Transactions on Computers*, vol. 26, 98–111, 1977.

[Gallagher88] Kevin Gallagher, Daniel Corkill, and Philip Johnson, *GBB Reference Manual*, Technical Report 88-66, Computer and Information Science Department, University of Massachusetts, 1988.

[Garvey87] Alan Garvey, Craig Cornelius, and Barbara Hayes-Roth, "Computational Costs versus Benefits of Control Reasoning," *Proceedings of AAAI-87*, 110–115, 1987.

[Garvey93] Alan Garvey and Victor Lesser, "Design-to-time Real-Time Scheduling," to appear in *IEEE Transactions on Systems, Man, and Cybernetics*, Special Issue on Planning, Scheduling and Control, 1993 (also available as Technical Report 91-72, Computer and Information Science Department, University of Massachusetts, 1991).

[Hayes-Roth79] Barbara Hayes-Roth, Frederick Hayes-Roth, Stan Rosenschein, and Stephen Cammarata, "Modeling Planning as an Incremental, Opportunistic Process," *Proceedings of IJCAI-79*, 375–383, 1979 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 1988).

[Hayes-Roth85a] Barbara Hayes-Roth, "A Blackboard Architecture for Control," *Artificial Intelligence*, volume 26, 251–321, 1985.

[Hayes-Roth85b] Barbara Hayes-Roth and Micheal Hewett, *Learning Control Heuristics in a Blackboard Control Environment*, Technical Report HPP-85-2, Computer Science Department, Stanford University, 1985.

[Hayes-Roth86a] Barbara Hayes-Roth, Bruce Buchanan, Oliver Lichtarge, Micheal Hewett, Russ Altman, James Brinkley, Craig Cornelius, Bruce Duncan, and Oleg Jardetzky, "PROTEAN: Deriving Protein Structure from Constraints," *Proceedings of AAAI-86*, 904–909, 1986 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 1988).

[Hayes-Roth86b] Barbara Hayes-Roth, M. Vaughn Johnson, Alan Garvey, and Micheal Hewett, "Application of the BB1 Blackboard Architecture to Arrangement-Assembly Tasks," *International Journal for Artificial Intelligence in Engineering*, vol. 1, no. 2, 85–94, 1986.

[Hayes-Roth87] Barbara Hayes-Roth, Alan Garvey, M. Vaughn Johnson, and Micheal Hewett, *A Layered Environment for Reasoning about Action*, Technical Report KSL 86-38, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1987.

[Hayes-Roth88] Barbara Hayes-Roth and Micheal Hewett "BB1: An Implementation of the Blackboard Control Architecture," in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 297–313, 1988.

[Hayes-Roth89] Barbara Hayes-Roth, Richard Washington, Rattikorn Hewett, and Micheal Hewett, "Intelligent Monitoring and Control," *Proceedings of IJCAI-89*, 243–249, 1989.

[F. Hayes-Roth77] Frederick Hayes-Roth and Victor Lesser, "Focus of Attention in the Hearsay-II Speech Understanding System," *Proceedings of IJCAI-77*, 27–35, 1977.

[Hewett92] Micheal Hewett and Rattikorn Hewett, "A Language and Architecture for Efficient Blackboard Systems," submitted to IEEE CAIA-93, 1992.

[Hudlicka84] Eva Hudlicka and Victor Lesser, "Meta-Level Control Through Fault Detection and Diagnosis," *Proceedings of AAAI-84,* 153–161, 1984.

[Jagannathan89] V. Jagannathan, Rajendra Dodhiawala, and Lawrence Baum, editors, *Blackboard Architectures and Applications*, Academic Press, 1989.

[Johnson88] M. Vaughn Johnson, Jr. and Barbara Hayes-Roth, *Learning to Solve Problems by Analogy*, Technical Report KSL 88-01, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1988.

[Johnson89] M. Vaughn Johnson, Jr. and Barbara Hayes-Roth, "Simultaneous Dynamic Integration of Diverse Reasoning Methods," in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence Baum, editors, Academic Press, 57–73, 1989.

[Laasri89] Hassan Laasri and Brigitte Maitre, "Flexibility and Efficiency in Blackboard Systems: Studies and Achievements in ATOME," in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence Baum, editors, Academic Press, 309–322, 1989.

[Lakin88] W.L. Lakin, J.A.H. Miles, and C.D. Byrne, "Intelligent Data Fusion for Naval Command and Control," in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 443–458, 1988.

[Lander92] Susan Lander and Victor Lesser, "Negotiated Search: Organizing Cooperative Search Among Heterogeneous Expert Agents," *Proceedings of the Fifth International Symposium on Artificial Intelligence, Applications in Manufacturing and Robotics*, 351–358, 1992.

[Lesser75] Victor Lesser, Richard Fennell, Lee Erman, and D. Raj Reddy, "Organization of the Hearsay-II Speech Understanding System," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-23, 11-24, 1975.

[Lesser77a] Victor Lesser, Frederick Hayes-Roth, Mark Birnbaum, and Robert Cronk, "Selection of Word Islands in the Hearsay-II Speech Understanding System," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 791–794, 1977.

[Lesser77b] Victor Lesser and Lee Erman, "A Retrospective View of the Hearsay-II Architecture," *Proceedings of IJCAI-77*, 790–800, 1977 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 87–121, 1988).

[Lesser80] Victor Lesser and Lee Erman, "Distributed Interpretation: A Model and Experiment," *IEEE Transactions on Computers*, vol. 29, No. 12, 1144–1163, 1980.

[Lesser83] Victor Lesser and Daniel Corkill, "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks," *AI Magazine*, vol. 4, no. 3, 15–33, 1983 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 1988).

[Lesser89] Victor Lesser, Daniel Corkill, Robert Whitehair, and Joseph Hernandez, "Focus of Control Through Goal Relationships," *Proceedings of IJCAI-89*, 497–503, 1989 (an extended version of this paper is in *Blackboard Architectures and Applications*, V. Jagannathan, Rajendra Dodhiawala, and Lawrence Baum, editors, Academic Press, 1989).

[Lesser91a] Victor Lesser, "A Retrospective View of FA/C Distributed Problem Solving," *IEEE Transactions on Systems, Man, and Cybernetics*, special issue on distributed artificial intelligence, vol. 21, no. 6, 1347–1362, 1991.

[Lesser91b] Victor Lesser, Hamid Nawab, et al., *Integrated Signal Processing and Signal Understanding*, Technical Report 91-34, Computer and Information Science Department, University of Massachusetts, 1991.

[Maksym83] Joseph Maksym, Anthony Bonner, C. Ann Dent, and Gavin Hemphill, "Machine Analysis of Acoustical Signals," in *Issues in Acoustic Signal/Image Processing and Recognition*, C. H. Chen, editor, Springer-Verlag, 95–112, 1983.

[Nii78] H. Penny Nii and Edward Feigenbaum, "Rule-Based Understanding of Signals," in *Pattern-Directed Inference Systems*, D. A. Waterman and Frederick Hayes-Roth, editors, Academic Press, 483–501, 1978.

[Nii79] H. Penny Nii and Nelleke Aiello, "AGE (Attempt to GEneralize): A Knowledge-Based Program for Building Knowledge-Based Programs," *Proceedings of IJCAI-79*, 645–655, 1979 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 1988).

[Nii82] H. Penny Nii, Edward Feigenbaum, John Anton, and A. J. Rockmore, "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *The AI Magazine*, vol. 3, no. 1, 23–35, 1982 (also in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 1988).

[Nii86a] H. Penny Nii, "The Blackboard Model of Problem Solving," *The AI Magazine*, vol. 7, no. 2, 38–53, 1986.

[Nii86b] H. Penny Nii, "Blackboard Systems Part Two: Blackboard Application Systems," *The AI Magazine*, vol. 7, no. 3, 82–106, 1986.

[Nirenburg89] Sergei Nirenburg, Victor Lesser, and Eric Nyburg, "Controlling a Language Generation Planner," *Proceedings of IJCAI-89*, 1524–1530, 1989.

[Pearl88] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufman, 1988.

[Pearson88] Glen Pearson, "Mission Planning within the Framework of the Blackboard Model," in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 433–442, 1988.

[Reddy73a] D. Raj Reddy, Lee Erman, and R. Neely, "A model and a system for machine recognition of speech," *IEEE Transactions on Audio and Electroacoustics*, vol. AU-21, 229–238, 1973.

[Reddy73b] D. Raj Reddy, Lee Erman, Richard Fennell, and R. Neely, "The Hearsay speech understanding system: An example of the recognition process," *Proceedings of IJCAI-73*, 185–193, 1973.

[Rich91] Elaine Rich and Kevin Knight, *Artificial Intelligence*, second edition, McGraw-Hill, 1991.

[Smith85] Stephen Smith and Peng Si Ow, "The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks," *Proceedings of IJCAI-85*, 1013–1015, 1985.

[Srihari87] Sargur Srihari, Ching-Huei Wang, Paul Palumbo, and Jonathan Hull, "Recognizing Address Blackboard on Mail Pieces," *AI Magazine*, vol. 8, no. 4, 25–40, 1987.

[Terry88] Allan Terry, "Using Explicit Strategic Knowledge to Control Expert Systems," in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 159–188, 1988.

[Velthuijsen92] Hugo Velthuijsen, *The Nature and Applicability of the Blackboard Architecture*, PTT Research, 1992.

[Washington89] Richard Washington and Barbara Hayes-Roth, "Input Data Management in Real-Time AI Systems," *Proceedings of IJCAI-89*, 250–255, 1989.

[Whitehair92] Robert Whitehair and Victor Lesser, *A Framework for the Analysis of Sophisticated Control*, to appear as a technical report, Computer Science Department, University of Massachusetts, 1992.

[Williams88] Mark Williams, "Hierarchical Multi-expert Signal Understanding," in *Blackboard Systems*, Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 387–415, 1988.

[Williams77a] Thomas Williams and John Lowrance, *Model-Building in the VISIONS High Level System*, Technical Report 77-1, Computer and Information Science Department, University of Massachusetts, 1977.

[Williams77b] Thomas Williams, John Lowrance, Allen Hanson, and Edward Riseman, "Model-Building in the VISIONS System," *Proceedings of IJCAI-77*, 644–645, 1977.

[Woods77] W. A. Woods, "Shortfall and Density Scoring Strategies for Speech Understanding Control," *Proceedings of IJCAI-77*, 18–26, 1977.