AI Magazine Volume 7 Number 3 (1986) (© AAAI)

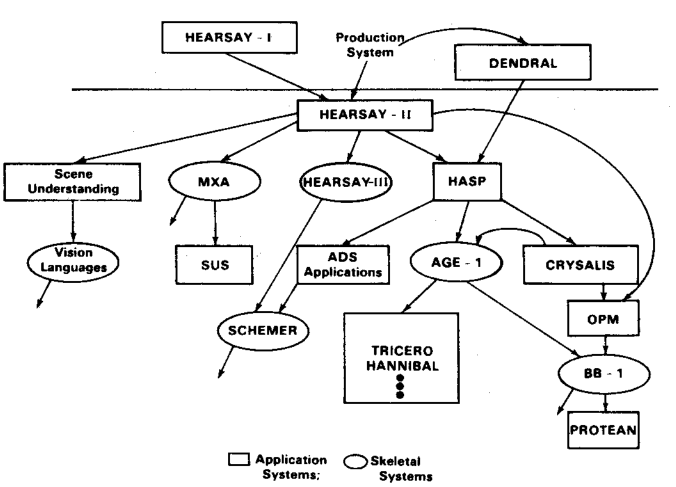
PART TWO

Blackboard Application Systems and a Knowledge Engineering Perspective

H. Penny Nii

Blackboard Application Systems

The application systems described here are presented in chronological order. The design of many of the systems is similar because of similarities in the application tasks, propagation of ideas, or involvement of the same designers. Figure 1 shows a general chronology and intellectual lineage of the various application and skeletal systems. The figure includes some of the better-known and better-documented systems. Only a few of the many application systems are described here; they were chosen because they illustrate different designs and because they contributed new ideas and features to the repertoire of blackboard system architectures. For each application, the task and domain characteristics are described. The description is followed by a summary of the system design in



References: Hearsay-I (Reddy *et al.* 1973); Production Systems (Newell and Simon 1972, Davis and King 1977); Dendral (Lindsay *et al.* 1980); Hearsay-II (Erman *et al.* 1980); Scene Understanding (Nagao and Matsuyama 1979); Vision Language (Shafer *et al.* 1986); MXA and SUS (Lakin and Miles 1984); Hearsay-III (Erman *et al.* 1981); HASP (Nii *et al.* 1982); ADS applications (Spain 1983, McCune and Drazovich 1983); AGE-I (Nii and Aiello 1979); CRYSALIS (Terry 1983); OPM (Hayes-Roth *et al.* 1979); BB1 (Hayes-Roth 1985); TRICERO (Williams *et al.* 1984); HANNIBAL (Brown *et al.* 1982); PROTEAN (Hayes-Roth *et al.* 1987).

Influences Among Blackboard Systems

Figure 1

Abstract

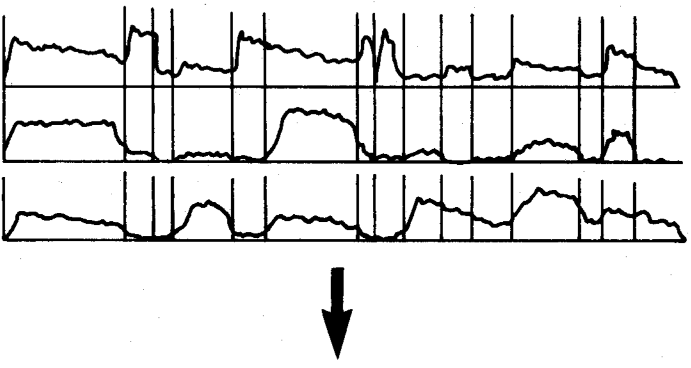
The first blackboard system was the Hearsay-II speech-understanding system (Erman *et al.* 1980), which evolved between 1971 and 1976. Subsequently, many systems have been built that have similar system organization and run-time behavior. The objectives of this document (a part of a retrospective monograph on the AGE Project currently in preparation) are (1) to define what is meant by blackboard systems and (2) to show the richness and diversity of blackboard system designs. In Part 1 we discussed the underlying concept behind all blackboard systems—the blackboard model of problem solving. In order to bridge the gap between the model and working systems, we introduced and discussed the blackboard framework. We also traced the history of ideas and designs of some application systems that helped shape the blackboard model. In Part 2, we describe and contrast existing blackboard systems. Blackboard systems can generally be divided into two categories: application systems and skeletal systems. In application systems, the blackboard system components are integrated into the domain knowledge required to solve the problem at hand. Skeletal systems are devoid of domain knowledge and, as the name implies, consist of the essential system components from which application systems can be built by the addition of knowledge and the specification of control (that is, metaknowledge). Application systems are discussed in the first section, Blackboard Application Systems, and skeletal systems will be discussed elsewhere. In Summary, we summarize the features of the application systems. In Blackboard Systems from a Knowledge Engineering Perspective, the author's perspective on the utility of the blackboard approach to problem solving and knowledge engineering is discussed.

H. Penny Nii is a Senior Research Associate in the Knowledge Systems Laboratory, Computer Science Department, Stanford University

This work was supported by grants from the Advanced Research Projects Agency (N0039-83-C-0136), the National Institutes of Health (5P41 PR-00785), and Boeing Computer Services (W266875).

four parts: the blackboard structure, the knowledge source organization, the control component, and the knowledge-application strategy employed. Unique features in the system are pointed out and discussed within the context of either the application task or its history.

HEARSAY-II



The HEARSAY-II Task

Figure 2

Most of the background information on HEARSAY-II (Erman *et al.* 1980) was covered in Part 1 of this article and is not repeated here. One additional item of historical context is worth noting, however. Various continuous speech understanding projects were brought under one umbrella in the Defense Advanced Research Projects Agency (DARPA) Speech Understanding Project, a five-year project that began in 1971. The goals of the Speech Understanding Project were to design and implement systems that "accept continuous speech from many cooperative speakers of the general American dialect in a quiet room over a good-quality microphone, allowing a slight tuning of the system per speaker, by requiring only natural adaptation by the user, permitting a slightly selected vocabulary of 1,000 words, with a highly artificial syntax ... in a few times real time..." (Newell *et al.* 1973). Hearsay-II was developed at Carnegie-Mellon University for the Speech Understanding Project and successfully met most of these goals.

The Task. The goal of the HEARSAY-II system was to understand speech utterances. To prove that it understood a sentence, it performed the spoken commands. In the earlier HEARSAY-I period, the domain of discourse was chess (for example, bishop moves to king knight five). In the HEARSAY-II era, the task was to answer queries about, and to retrieve documents from, a collection of computer science abstracts in the area of artificial intelligence. For example, the system understood the following types of command:

> "Which abstracts refer to the theory of computation?"
>
> "List those articles."
>
> "What has McCarthy written since nineteen seventy-four?"

The HEARSAY-II system was not restricted to any particular task domain. "Given the syntax and the vocabulary of a language and the semantics of the task, it attempts recognition of the utterance in that language." (Reddy, Erman, & Neely 1973b.) The vocabulary for the document retrieval task consisted of 1,011 words in which each extended form of a root, for example, the plural of a "noun", was counted separately. The grammar defining a legal sentence was context-free and included recursion, and imbedded semantics and pragmatic constraints. For example, in the place of "noun" in conventional grammars, this grammar included such nonterminals as topic, author, year, and publisher. The grammar allowed each word to be followed, on the average, by seventeen other words in the vocabulary.

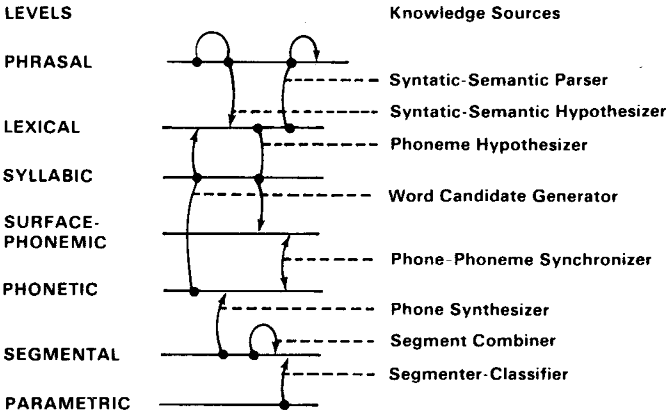
The problem of speech understanding is characterized by error and variability in both the input and the knowledge. "The first source of error is due to deviation between ideal and spoken messages due to inexact production [input], and the second source of error is due to imprecise rules of comprehension [knowledge]." Because of these uncertainties, a direct mapping between the speech signals and a sequence of words making up the uttered sentence is not possible. The HEARSAY designers structured the understanding problem as a search in a space consisting of complete and partial interpretations. These interpretations were organized within an abstraction hierarchy containing signal parameters, segments, phones, phonemes, syllables, words, phrases, and sentence levels. This approach required the use of a diverse set of knowledge that produced large numbers of partial solutions on the many levels. Furthermore, the uncertainties in the knowledge generated many competing, alternative hypothetical interpretations. To avoid a combinatorial explosion, the knowledge sources had to construct partial interpretations by applying constraints at each level of abstraction. For example, one kind of constraint is imposed when an adjacent word is predicted, and the prediction is used to limit subsequent search. The constraints also had to be added in such a way that their accrual reduced the uncertainty inherent in the data and the knowledge sources.

In order to control the combinatorial explosion and to meet the requirement for near real-time understanding, the interpretation process had to be selective in exploiting the most promising hypotheses, both in terms of combining them (for example, combining syllables into words) and in terms of predicting neighboring hypotheses around them (for example, a possible adjective to precede a noun). Thus, the need for incremental problem solving and flexible, opportunistic control were inherent in HEARSAY's task.

The Blackboard Structure. The blackboard was partitioned into six to eight (depending on the configuration) levels of analysis corresponding to the intermediate levels of the decoding process.[1] These levels formed a hierarchy in which the solution-space elements on each level could be described loosely as forming an abstraction of information on its adjacent lower level. One such hierarchy was comprised of, from the lowest to the highest level: parametric, segmental, phonetic, phonemic, syllabic, lexical, phrasal, and conceptual levels (see figure 3). A blackboard element represented a hypothesis. An element at the lexical level, for example, represented a hypothesized word whose validity was supported by a group of syllables on the syllable level. The blackboard could be viewed as a three-dimensional problem space with time (utterance sequence) on the x-axis, information levels containing a hypothesized solution on the y-axis, and alternative solutions on the z-axis. (Lesser *et al.* 1974)

Each hypothesis, no matter which level it belonged to, was constructed using a uniform structure of attribute-value pairs. Some attributes, such as its level name, were required for all levels. The attributes included a validity rating and an estimate of the "truth" of the hypothesis represented as some integer value. The relationships between the hypotheses on different levels were represented by links, forming an AND/OR tree over the entire hierarchy. Alternative solutions were formed by expanding along the OR paths. Because of the uncertainty of the knowledge sources that generated the hypotheses, the blackboard had a potential for containing a large number of alternative hypotheses.

The Knowledge Source Structure. Each knowledge source had two major components: a condition part (often referred to as a precondition) and an action part. Both the condition and the action parts were written as arbitrary SAIL procedures. "The condition component prescribed the situations in which the knowledge sources may contribute to the problem solving activity, and the action component specified what that contribution was and how to integrate it into the current situation." (Erman *et al.* 1980) When executed, the condition part searched the blackboard for hypotheses that were of interest to its corresponding action part; all the relevant hypotheses found during the search were passed on to the action part. Upon activation, the action part processed all the hypotheses passed to it. The tasks of the knowledge sources ranged from classification (classifying acoustic segments into phonetic classes), to recognition (recognizing words) to generation and evaluation of predictions.



HEARSAY-II Blackboard and Knowledge Sources

Figure 3

Control. The control component consisted of a blackboard monitor and a scheduler (see Figure 4). The monitor kept an account of each change made to the blackboard, its primitive change type, and any new hypotheses. Based on the change types and declarative information provided by the condition part of the knowledge sources, the monitor placed pointers to those condition parts that potentially could be executed on a scheduling queue.[2] In addition to the condition parts ready for execution, the scheduling queue held a list of pointers to any action parts ready for execution. These actions parts were called the *invoked knowledge sources.* A knowledge source became invoked when its condition part was satisfied. The condition parts and the invoked knowledge sources on the scheduling queue were called *activities.* The scheduler calculated a priority for each activity at the start of each system cycle and executed the activity with the highest priority in that cycle.

In order to select the most productive activity (the most important and promising with the least amount of processing and memory requirements), the scheduler used

[1] See Lesser & Erman [1977] for a comprehensive discussion on the results of experiments conducted with two different blackboard configurations.

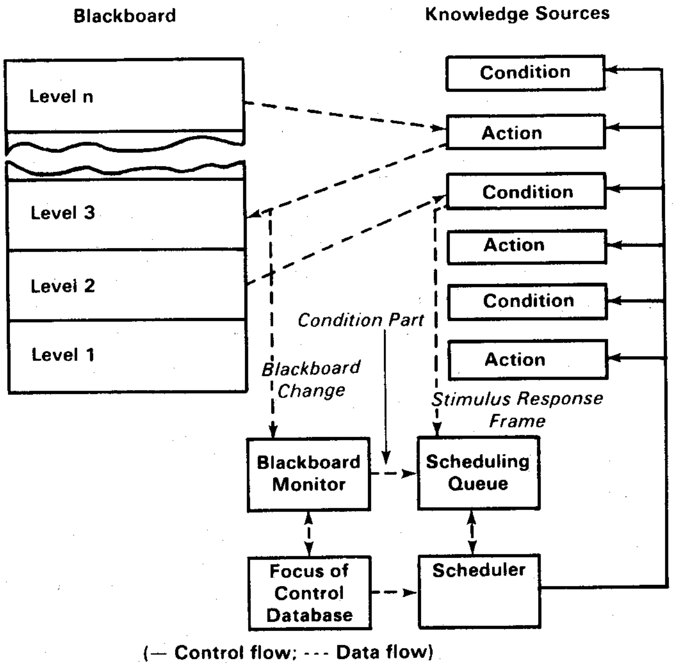
[2] In Figure 4 the "Focus-of-control database" contained a table of primitive change types and the condition parts that could process each change type. The primitive change types possible within the system were predefined and consisted of such items as "new syllable" and "new word created bottom up". This paragraph is based on discussions with Lee Erman.

experimentally derived heuristics to calculate the priority. These heuristics were represented as imbedded procedures within the scheduler. The information needed by the scheduler was provided in part by the condition part of each invoked knowledge source. The condition part provided a *stimulus frame*—a set of hypotheses that satisfied the condition—and a *response frame*—a stylized description of the blackboard changes the knowledge source action-part might produce upon execution. For example, the stimulus frame might indicate a specific set of syllables, and the response frame would indicate an action that would produce a word. The scheduler used the stimulus-response frames and other information on the blackboard to select the next thing to do.

The control component iteratively executed the following basic steps:

1. The scheduler selected from the scheduling queue an activity to be executed.
2. If a condition part was selected and executed and if it was satisfied, a set of stimulus-response frames was put on the scheduling queue together with a pointer to the invoked knowledge source.
3. If an action part was selected and executed, the blackboard was modified. The blackboard monitor posted pointers to the condition parts that could follow up the change on the scheduling queue.

The problem of *focus of attention* was defined, in the context of this architecture as a problem of developing a method which minimized the total number of knowledge source executions and which achieved a relatively low rate of error. The focus of attention problem was viewed as a knowledge scheduling problem as well as a resource-allocation problem.[3] In order to control the problem-solving behavior of the system, the scheduler needed to know the goal of the task and the strategies for knowledge application to be able to evaluate the next best move. Although various general solutions to this problem have been suggested (Hayes-Roth & Lesser 1977), it appears that ultimately one needs a knowledge-based scheduler for the effective utilization of the knowledge sources.[4]



Schematic of HEARSAY-II architecture

Figure 4

Knowledge-Application Strategy. Within the system framework described earlier, HEARSAY-II employed two problem-solving strategies. The first was a bottom-up strategy whereby interpretations were synthesized directly from the data, working up the abstraction hierarchy. For example, a word hypothesis was synthesized from a sequence of phones. The second was a top-down strategy in which alternative sentences were produced from a sentential concept, alternative sequences of words from each sentence, alternative sequences of phones from each word, and so on. The goal of this recursive generation process was to produce a sequence on the parametric level that was consistent with the input data (that is, to generate a hypothetical solution and to test it against the data). Both approaches have the potential for generating a vast number of alternative hypotheses and with it a combinatorially explosive number of knowledge source activations. Problem-solving activity was, therefore, constrained by selecting only a limited subset of invoked knowledge sources for execution. The scheduling module thus played a crucial role within the HEARSAY-II system.

Orthogonal to the top-down and bottom-up approaches, HEARSAY employed a general hypothesize-and-test strategy. A knowledge source would generate hypotheses, and their validity would be evaluated by some other knowledge source. The hypothesis could be generated by a top-down analytic or a bottom-up synthetic approach. Often, a knowledge source generated or tested hypotheses by matching its input data against a "matching prototype" in its knowledge base. For example, a sequence of hypothesized phones on the phone level were matched against a table containing prototypical patterns of phones

[3] If we compare the HEARSAY-II control constructs with those of the blackboard framework discussed in Part 1, they are basically the same. Some aspects of the control in HEARSAY are emphasized more (for example, scheduling) than others.

[4] The current work of Barbara Hayes-Roth [1985] on the BB1 system elaborates this point.

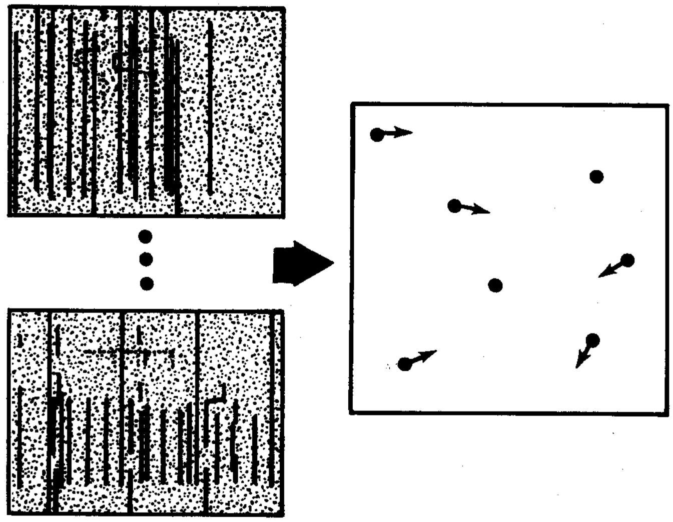
for each word in the vocabulary. A word whose phones satisfied a matching criterion became a word hypothesis for the phones. The validation process involved assigning credibility to the hypothesis based on the consistency of interpretation with the hypotheses on an adjacent level.

At each problem-solving step, any one of the bottom-up synthesis, top-down goal generation, neighborhood prediction, hypothesis generation, and hypothesis evaluation might have been initiated. The decision about whether a knowledge source could contribute to a solution was local to the knowledge source (precondition). The decision about which knowledge source should be executed in which one of many contexts was global to the solution state (the blackboard), and the decision was made by a global scheduler. The scheduler was opportunistic in choosing the next step, and the solution was created one step at a time.

Additional Notes

- The condition parts of the knowledge sources were complex, CPU-intensive procedures that needed to search large areas of the blackboard. Each knowledge source needed to determine what changes had been made since the last time it viewed the blackboard. To keep from firing the condition parts continually, each condition part declared *a priori* the kinds of blackboard changes it was interested in. The condition part, when executed, looked at only the relevant changes since the last cycle. All the changes that could be processed by the action part were passed to it to avoid repetitive executions of the action part.
- The HEARSAY-II system maintained alternative hypotheses. However, the maintenance and the processing of alternatives are always complex and expensive, especially when the system does not provide support for this. In HEARSAY-II the problem was aggravated by an inadequate network structure that did not allow the shared network to be viewed from different perspectives. In the current jargon, it did not have good mechanisms for processing multiple worlds.[5]
- The evidence to support a hypothesis at a given level can be found on lower levels or on higher levels. For example, given a word hypothesis, its validity could be supported by a sequence of syllables or by grammatical constraints. The evidential support is represented by directional links from the evidence to the hypothesis it supports. The link that goes from a higher-level to a lower-level hypothesis represents a "support from above" (that is, the justification for the hypothesis can be found at a higher level). A link that goes in the opposite direction represents support from below (that is, the reason for the hypothesis can be found at a lower level). Although the names of the support mechanisms were first coined in HASP (Nii & Feigenbaum 1978), the bidirectional reasoning mechanisms were first used in the HEARSAY-II system.
- In HEARSAY-II the confidence in a hypothesis generated by a knowledge source was represented by an integer between 1 and 100. The overall confidence in the hypothesis was accumulated by simple addition of the confidence attached to the evidence (that is, supporting hypotheses). When the confidence in a hypothesis was changed, the change was propagated up (if the support was from below) and down (if the support was from above) the entire structure.

HASP/SIAP



HASP/SIAP Task

Figure 5

The HASP project began in 1972 under the sponsorship of (DARPA). The HASP project was terminated in 1975 but was reinstated in 1976 under the name SIAP (Nii *et al.* 1982). At the time, the computational resources needed to maintain a major ocean surveillance system of sensors with conventional methods of statistical signal processing seemed economically unfeasible. It was also a time when artificial intelligence techniques were first being applied to the problem of signal interpretation. The DENDRAL program (Lindsay *et al.* 1980) was achieving significant success and the Speech Understanding Project, of which the HEARSAY Project was a part, was under way. The major objectives of the HASP project were to demonstrate

[5] Currently, there are better techniques for processing and maintaining alternative worlds. However, these techniques have yet to be integrated into blackboard systems.

that artificial intelligence techniques could contribute significantly in addressing the surveillance problem and, further, that the project could be accomplished with reasonable computing resources. HASP was successful in meeting both these objectives.

The Task. The task of the HASP/SIAP system was to develop and maintain a situation board that reflected the activities of platforms (surface ships and submarines) in a region under surveillance. The situation board was developed by interpreting multiple, continuous streams of acoustic signals produced by objects in the region and by integrating intelligence reports with the interpretation.

The acoustic input to the system came in the form of digitized data from multiple hydrophone arrays, each monitoring a part of the region.[6] Each array had multiple hydrophones with some directional resolution. The major sources of acoustic radiation were rotating shafts and propellers and reciprocating machinery on board a platform. The *signature,* or sound spectrum, of a platform under steady operation contained persistent fundamental narrow band frequencies and certain of their harmonics. The front-end signal-processing hardware and software detected energy peaks appearing at various spectral frequencies, and followed these peaks over time. On an analyst's sonograms, the peaks appeared as a collection of dark vertical stripes (see Figure 5). Under ideal conditions, a hydrophone picked up sound energy near its axis. In practice, the terrain of the ocean floor, water temperature, and other platforms interfered, producing signals with very low signal-to-noise ratios. That is, the stripes in the sonogram appeared against a very fuzzy background.

In addition to the acoustic data, intelligence reports were available to HASP. The reports contained information about movements of friendly and hostile platforms with varying degrees of confidence. Routine information on commercial shipping activities was also included in these reports.

As in the speech-understanding problem, the sonar–signal-understanding problem is characterized by a large solution space, a low signal-to-noise ratio, and uncertain knowledge. Unlike the speech problem, the semantics and the syntax are ill defined in the sonar problem. That is, the targets of highest priority, the enemy submarines, are most likely to be ill understood and, at the same time, are trying their best to go undetected. The implications are these: (1) There is no "legal move generator" for the solution space except at the highest level of abstraction. (It is assumed that different types of enemy submarines and their general characteristics are known.) (2) One must rely heavily on the analysts' methods and heuristics in detecting and classifying enemy submarines. (3) In order to find the targets, the analysts accounted for all known entities (primarily surface platforms) and looked for the targets of interest within the unaccounted-for data.[7] The problem is somewhat akin to the following tasks: When there are two people talking at the same time, one in English and another in a relatively unfamiliar language, try to pick up what the non-English speaker is saying. Another task is the cocktail conversation problem in which many people are talking as they move around; the task is to keep track of each person using data from microphones scattered around the room.

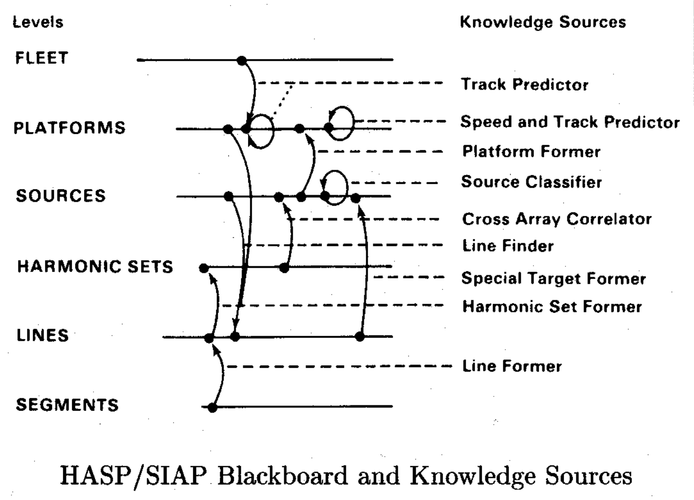
Even given all the difficulties, there were aspects of the problem that made it tractable. The situation unfolded over a relatively long period of time because the platforms moved rather slowly, but the data collection was relatively frequent and from many different locations. This meant that the system was given many chances to interpret the situation with data sets containing slightly different information. For example, two hydrophones might pick up incomplete harmonic sets attributable to the same platform, but they might be fractured in different ways. When combined, they provided more information than from each one separately. There also were many different kinds of knowledge that could be used, bits and pieces, such as in the Koala problem discussed in Part 1. The general strategy employed was to accumulate both positive and negative evidence for a hypothesis element.

The Blackboard Structure. The data structure on the blackboard represented the best understanding of the situation at any given point in time. It was a dynamic entity that evolved over time. Referred to as the *current best hypothesis* (CBH), it was partitioned into an abstraction hierarchy consisting of input segments, lines, harmonic sets, acoustic sources, platforms, and fleet levels (see Figure 6). The signal data arrived on the segments level, and the report data arrived on either the fleets level or the platforms level, depending on the content of the report.

Unlike the HEARSAY-II system in which the "answer" to the problem was the hypothesized sentence on the highest level, HASP's "answer" was the network of partial solutions that spanned the entire blackboard. In other words, partial solutions were considered acceptable, if not desirable, solutions. For example, a partial solution of the form, "There's something out there producing these lines," was acceptable, even though a preferable solution was, "There is a platform of type x, whose engine is

[6] During the first phase of the project, the acoustic input consisted of segments that described signal events. For example, a piece of input might have contained a frequency and indicated it as a beginning of a frequency shift (called a *knee*). Later, five-minute segments produced by a signal-processing front-end system were used as the input data.

[7] This does not guarantee that the targets will be found. For one thing, the targets might be very quiet, and their sound may not be picked up by the hydrophones, or their sound might be overshadowed by noisier platforms. The HASP system conjectured about their existence and their whereabouts from other information.



HASP/SIAP Blackboard and Knowledge Sources

Figure 6

accounted for by the following harmonics and whose propeller seems to be producing the following lines, and no shaft data are currently being received."

The nodes on the blackboard were called *hypothesis elements* rather than hypotheses as they were in HEARSAY-II. The hypothesis elements formed a network, each element representing a meaningful aggregation of lower-level hypothesis elements. No attempt was made to maintain uniformity of attributes across the levels. Each knowledge source knew the relevant vocabulary (*attributes*) associated with those levels in which it was interested. The lines level and the harmonic-sets level used a descriptive vocabulary that dealt primarily with signal characteristics, and the sources level used vocabulary dealing primarily with machinery. Thus, the point of signal-to-symbol transformation can be said to have occurred between the harmonic-sets level and the sources level. Signal information in a hypothesis element on the harmonic-sets level was translated into machinery information in a hypothesis element on the sources level; that is, there was an element-for-element translation between the two levels.

In contrast to HEARSAY-II, each hypothesis element could have alternative values for its attributes but no alternative links. The hierarchy was organized as an AND tree, with a possibility for local alternatives. Although this approach reduced computational time and space, it was awkward for the system to "change its mind" about the solution. In HEARSAY "changing its mind" might only have involved focusing on an alternative structure. In HASP either the affected hypothesis elements had to be reanalyzed (which could result in reorganizing the whole CBH), or the past analyses dealing with the elements in question had to be forgotten and the analysis restarted from the point of departure. The latter approach was used in HASP because the human analysts tended to behave in a similar manner.[8]

In addition to the blackboard, HASP had other globally accessible information generated directly or indirectly by the knowledge sources (refer to Figure 7). This global information was used primarily by the control modules:

- *Event List:* All changes made to the blackboard, together with the types of these changes, were posted on the event list. Each event had a generic "change type" associated with it. An event also had associated with it a particular blackboard node. An event in the event list was selected by a control module to become a focus of attention. The focus of attention, then, had two implicit components: a change type and a blackboard node. (A more detailed discussion can be found in the Control section.)
- *Expectation List:* The Expectation-list contained events (event types and associated hypothesis elements) that were expected to occur in the future. Thus, acoustic signature of platforms reported to be in the region in the intelligence reports were posted on the expectation list. The canonical acoustic signatures of all the known platforms were stored in a static knowledge base.[9] Periodically, the expectation list was searched to see if expected data had arrived.
- *Problem List:* This list contained a description of the various problems the knowledge sources encountered. For example, when no rule fired during the execution of a knowledge source, it might have meant, "I should know, but don't." Such information was useful to the programmers. The most important use of this list, however, was for posting missing or desired information. A knowledge source could post pieces of information, that if available, would increase the confidence in its hypothesis. For example, a knowledge source might indicate that if the dependency relationship was known for a given set of lines, it might be able to identify the platform. In such a case, an operator might provide the information if it was known, or a goal might be set up by a control module to find the information.
- *Clock Event List:* A clock event consisted of a time and associated rules. The rules were to be executed at the designated time. Because behaviors at various levels were known for some types of platforms, knowledge sources tracked the expected and actual behavior by

[8] In the human system there are analysts whose task is to do offline postanalyses. What they learn from the postanalyses is often added to the pool of knowledge about the task. HASP had no counterpart to this activity.

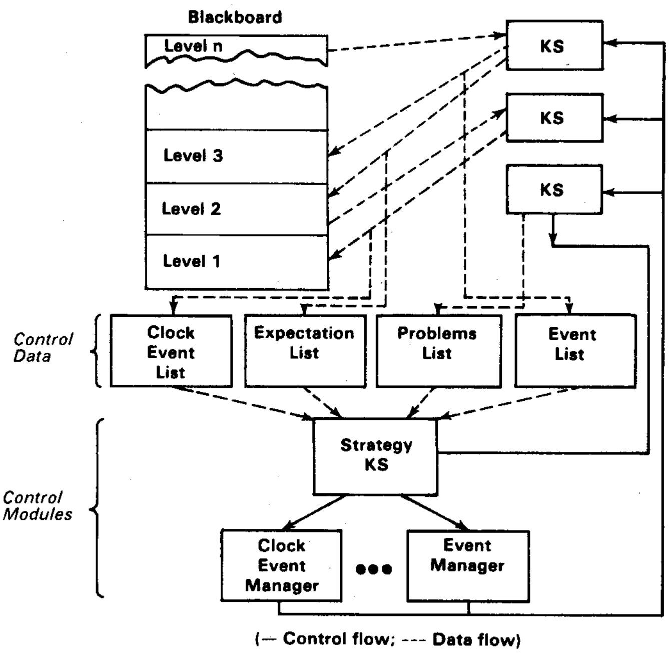
[9] In the entire discussion of blackboard systems, the role and the form of the static knowledge base have been omitted. It is assumed that taxonomies, facts, and definitions are represented in some form. This type of knowledge is awkward to represent as rules and is usually represented as tables, records, property lists, or frames.

this mechanism. The types of behavior known to the system ranged from the temporal characteristics of the sonograms to the physical movement of the platforms.

- *History List:* All the processed events and their context (for example, a blackboard node and its values and the bindings in a rule that made the change) were kept on this list. The history list was used to recount the knowledge-application steps that led to the generation of the CBH.[10] This list was also used by the programmers to ensure that solutions were arrived at by an expected line of reasoning. The designers wanted to detect occurrences of right answers for wrong reasons.

The Knowledge Source Structure. Each knowledge source consisted of a precondition part and an action part. In contrast to the HEARSAY-II knowledge source organization, the precondition part and the action part were contained in one module. The precondition part consisted of a list of pairs of tokens; the pair consisted of a name of an event type and its modifier (new, old, or modified). The modifier indicated the status of the hypothesis element (for example, modified hypothesis element) that was the focus of the events. When an event became "focused," knowledge sources whose precondition contained the event type of the focused event were executed. An event type was one of several predefined categories of changes that could be made in the system. The action part consisted of a set of rules. In this knowledge source organization, the precondition can be viewed as a simple trigger for a set of rules. The detailed test for applicability of knowledge occurred in the condition parts of the rules. The knowledge source could create bindings local to the knowledge source that remained valid for the duration of its execution. The bindings served to "freeze" the context until all the rules in a knowledge source were evaluated.

Control. Each of the control modules in HASP was written in the same form as the domain knowledge sources, that is, as a set of rules. The knowledge sources formed a simple control hierarchy (see the "control modules" in Figure 7). Although the control knowledge sources were logically independent, they were executed in a predefined order. The strategy knowledge source decided which categories of events (that is, clock, problem, expectation, or blackboard) to process next, based on priorities as encoded in its knowledge base. An appropriate event-management knowledge source was executed based on this decision. Once activated, an event manager, in turn, decided which specific event to focus on. The basis of this decision varied with the event manager. For example, the clock-event manager selected events that needed to be processed at a given time, and within those events the priority rested with events dealing with enemy platforms. The knowledge sources associated with the focused event were then executed. The node associated with the focused event served as the context for the knowledge source's execution. For example, the strategy knowledge source might have decided that it was time to process a blackboard event. It would activate the blackboard-event manager. The event manager in turn looked through the event list and selected an appropriate event as the focus of attention. Finally, one or more knowledge sources whose precondition contained the blackboard change type of the focused event were executed. The node associated with the event (that is, the blackboard node containing a change) served as the context for the knowledge sources.



HASP/SIAP System Organization

Figure 7

It was mentioned in Part 1 that the scheduling module and the focus-of-attention mechanisms were simpler in HASP than in HEARSAY-II. This is true in view of the following: Because it was known what blackboard changes were significant for making progress toward a solution, the HASP programmer decided what blackboard changes were to be called "events." That is, only certain changes to the blackboard were called events. For each such change, it was also known what knowledge sources were available for following up on the new information. By making the precondition of a knowledge source be the occurrence of

[10] Because the program processed events in a breadth-first order, and humans had difficulties in following this processing order, the history list was used to construct a text that made it appear as though the processing had been in depth-first order.

specific types of events, the selection of knowledge sources for a given event became a very simple matter. For example, a table of event types and applicable knowledge sources could be used.[11] In this scheme, however, the process of selecting the most promising event became a major issue. The selection of an event is really the selection of a node, which, in turn, is really a selection of a solution island. Thus, the focus-of-attention problem in HASP was primarily a problem of determining which solution island to work on next, rather than a problem of which knowledge source to apply next, as in HEARSAY-II. In HASP the hierarchical control knowledge sources were all biased toward the selection of a solution island to be pursued that would have the highest payoff in subsequent processing cycles. Once the focus-of-attention event was selected the relevant knowledge sources were easily selected and all the knowledge sources were executed in a predetermined but interchangeable order.

The basic actions of the control component were iterations of the following: (1) The strategy knowledge source decided which event category to focus on, that is, clock events, expectations, problems, or blackboard events. (2) The manager of the chosen event category selected a specific event from that category to process next. The event information contained the name of a node to which a change was made and change type associated with that change. The node name and the change type constituted the focus of attention. (3) Based on the change type of the focus-of-attention node, knowledge sources associated with the change type were executed. The node associated with the focus of attention served as the context for the activation of the knowledge sources. (4) The executing knowledge sources produced changes to the blackboard, and the changes were recorded.

To summarize, in HASP there were four categories of events: expectation, clock, problem, and blackboard. Each category of events contained a predetermined set of event types (that is, a set of expectation event types, a set of blackboard event types, and so on). For each event type, the knowledge sources that could process an instance of the event type were also predetermined. The system was openended in that new event types and new knowledge sources could be added without perturbing the existing ones. The major task for the control mechanism was to select the next solution island to be investigated.

Knowledge Application Strategy. As in HEARSAY-II, HASP used several problem solving approaches. A basic generate-and-test method was used to generate hypothesis elements and to test their credibility. Instead of using a *legal move generator* as was the case in HEARSAY-II, where the space of legal solutions was known from the grammar and vocabulary, HASP used a *plausible move generator* based on the heuristics used by the analysts. The construction of higher-level partial solutions from lower-level partial solutions, the determination of their properties, the generation of expectations, and so on, were driven by empirical association rules obtained from an analyst.

Most of the forty to fifty knowledge sources in HASP were engaged in bottom-up processing. Several pieces of data from a lower level were combined to form or update information on a higher level (for example, lines into harmonic sets). Similarly, information on one level was translated into a different vocabulary on another level (for example, harmonic sets into mechanical parts). The data were processed breadth first. That is, all the harmonic sets were formed from lines, and all sources were assigned to harmonic sets, and so on, in a pipeline fashion up the hierarchy.

The most powerful reasoning strategy used in HASP was the top-down, model-driven strategy. The assumption underlying model-driven reasoning is the following: In the interpretation of data, the amount of processing can be reduced by matching carefully selected pieces of data with discriminating or important features of a model (a frame or a script). A successful match tends to confirm the model as an explanatory hypothesis for the data. In a continuous-data interpretation task, the model, combined with periodic confirmatory matches, serves as the "cognitive flywheel" that maintains the ongoing "understanding." In driving a car, for example, our model of the road situation (prototypical highway characteristics, shapes of cars, their range of speed, their normal behavior, and so forth) saves us from having to continually process every bit of data within our visual range. Therefore, we don't "notice" the color of the upholstery of the car in front of us even though that piece of information is often available. The danger with this approach is that data can often match a wrong model for a long time, especially when the discriminating features are not carefully chosen.[12]

With this caveat, a model-driven approach is a very powerful device in interpreting noisy data. In HASP a model-driven approach was used quite extensively and successfully. For example, it was used in determining which lines formed a harmonic set. In fact, the CBH served as a situation model from one time frame to the next. There was an implicit assumption that the current state of affairs was not significantly different from the state a few minutes

[11] In HASP a set of simple rules was used. The condition side contained event types and a few other simple conditions, and the action side contained a sequence of knowledge sources to be executed.

[12] How often have you listened to a person and thought that person was talking about a particular topic before suddenly realizing it was a different topic all the time? (See Aiello 1983 for a simple experiment relevant to this topic.) The same pieces of knowledge from the PUFF (Kunz *et al.* 1978) program were used in data-driven, goal-driven, and model-driven approaches. Although the model-driven approach ran the fastest, extra knowledge had to be added to keep it from making the wrong diagnoses.

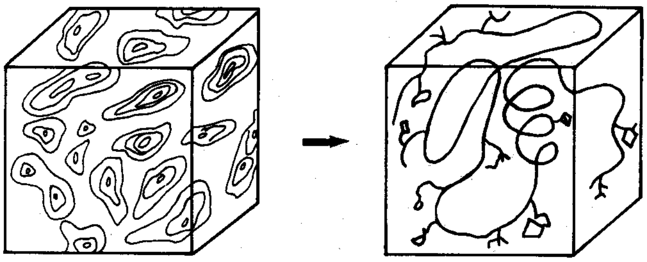
earlier. To make this assumption work, HASP focused on finding counterevidence for a hypothesis as much as on finding supporting evidence.

Within an abstraction hierarchy, model-driven reasoning is usually a top-down process. For example, if a platform type is "known" with support from above (for example, reports) or with support from below (for example, data), then the facts about the platform type can serve as a model. From this model, we can hypothesize the platform's range of speed, its sound-producing machinery and the machinery's acoustic signature, the platform's travel patterns, and so on. Pieces of data that can support the model-based hypothesis are sought in the signal data. As more supporting evidence is found, confidence in pursuing the model is increased. In this sense, the model serves as a constraint in the search process.

Additional Notes

- Major differences in the design of the HEARSAY-II and HASP systems are summarized below:
  - A knowledge source was written as a set of two procedures in HEARSAY and as a set of rules in HASP.
  - Each knowledge source in HEARSAY consisted of two procedures: the condition part and the action part. In HASP the precondition part was a list of tokens, and the precondition and the action parts were in one module.
  - In selecting a focus of attention, HEARSAY was concerned with selecting the next knowledge source to execute, and HASP was concerned with selecting the next solution island to pursue.
  - HEARSAY used a central scheduler to select its focus of attention; HASP partitioned the scheduling task and used a hierarchy of control knowledge sources to select the focus of attention.
  - In HEARSAY, a subset of knowledge sources was chosen for execution from a list of all applicable (invoked) knowledge sources. HASP executed all knowledge sources applicable to a focused event. However, not all the changes to the blackboard became focused events.
  - HASP was designed to interpret continuous, multiple streams of data. HEARSAY interpreted single-speech utterance.
- The hierarchical control in HASP was an attempt to separate the domain-specific knowledge from knowledge about the application of that knowledge. It was the first attempt at such an organization and was rather simplistic. In the CRYSALIS system (described next), the hierarchy of control knowledge sources was organized differently.
- In HASP the control-related information was made globally accessible. It was also decided to represent control functions in rule form. The grouping of control-related rules into control knowledge sources was an obvious next step. However, by not integrating the control information into the blackboard structure, the control rules had to be expressed and processed differently from the domain knowledge sources. The BB1 system (Hayes-Roth 1985) corrects this awkward representation problem.

CRYSALIS



The CRYSALIS Task

Figure 8

The CRYSALIS system (Terry 1983) contributed to the repertoire of blackboard system designs in two ways. First, it introduced the use of multiple hierarchies on the blackboard—*the blackboard panels.* Second, it addressed the control problem from the perspective of rule-based systems.[13]

The CRYSALIS project began in the spring of 1976 under the sponsorship of the National Science Foundation. It was a joint project between protein crystallographers at the University of California at San Diego and computer scientists in the Heuristic Programming Project at Stanford University. It was undertaken because the computer scientist thought that a blackboard approach "appeared to be appropriate" for this difficult task. The objective of the project was to build a system that determined the structures of proteins given their amino acid sequence and X-ray diffraction data for the protein crystals. The project did not reach its goal of building a system to construct complete stereo models of proteins. It did, however, succeed in a few cases in mapping more than 75% of the amino

[13] In this sense, the intellectual lineage of the control component of the CRYSALIS system is closely tied to MYCIN-like systems and applications written in OPS. One of the the major control issues in rule-based systems is to separate control information from domain rules and thus to make explicit the implied or built-in sequence of rule executions.

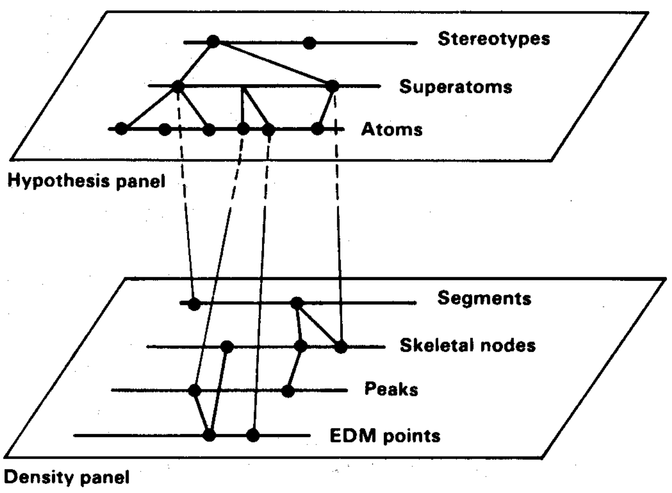
acid residues in the data. (As is seen later, this is equivalent to finding partial solutions on the middle level of the blackboard hierarchy.) Basically, this was a problem that usually took crystallographers months to solve and proved to be too difficult to solve completely. In retrospect, one can argue that this problem violated many of the criteria for choosing appropriate application problems: lack of experts from whom knowledge could be extracted and tested, almost no theoretical knowledge about the relation between the structures and the functions of proteins, and an impoverished state of knowledge about reasoning in three-space. Nonetheless, the CRYSALIS system did solve a significant part of the application problem and did contribute to the evolution of blackboard systems.

The Task. The task of the CRYSALIS system was to infer the three-dimensional structure of protein molecules. A protein structure was derived from an interpretation of the electron density map of the protein, which, in turn, is derived from X-ray diffraction data gathered from the crystallized protein. Traditionally, the protein crystallographer represents the explanation of the electron density map in a ball-and-stick molecular model fashioned from metal parts. These parts are strung together to form a model that conforms to the density map and is also consistent with protein chemistry and stereo chemical constraints.

The *electron density map* is derived from diffraction patterns produced by placing a protein crystal in an X-ray beam. It records the density of electrons in the protein molecule sampled at various points on a three-dimensional lattice. The resolution of the electron density map is typically poor, and the locations of individual atoms are generally not identifiable.

In addition to the electron density map, CRYSALIS was provided with the amino acid sequence of the protein. These data could also be errorful, with the sequence being incomplete or out of order. Nevertheless, given the amino acid sequence, the problem is narrowed to a task of determining the folding of the chain of amino acid residues and peptide bonds consistent with the distribution of the electron density.

Other data were available to CRYSALIS which wwere not used directly by human model builders. They were produced by mathematical algorithms that abstracted (or reduced) the electron density data by keying in on different features of the density data (the equivalent of the low-level-signal processing algorithms used in the speech- and other signal-understanding tasks). One reduced data set consisted of density peaks above some threshold and their locations; another consisted of connected peaks and regions called *skeletons;* and a third consisted of segments of the molecular skeleton. In some sense, these data were pieces of solutions produced by three different knowledge sources and were useful as intermediate solutions on the blackboard. However, the initial attempt to construct a hierarchy that included the various data and a target molecular model proved unsatisfactory. First, the data representation did not integrate well with the abstraction hierarchy of the molecular model, which consisted of atoms, superatoms, and secondary-structures levels. Second, the algorithms to generate the intermediate data were not designed to be used incrementally; that is, they could only work on the data for the entire molecule, not on small regions of the data. In order to organize all the data in a rational way, two hierarchical data structures were created for the blackboard: one to represent the bits and pieces of stereo structures conjectured during problem solving and the other to hold the data produced by the mathematical algorithms (see Figure 9).



The CRYSALIS Blackboard Panels

Figure 9

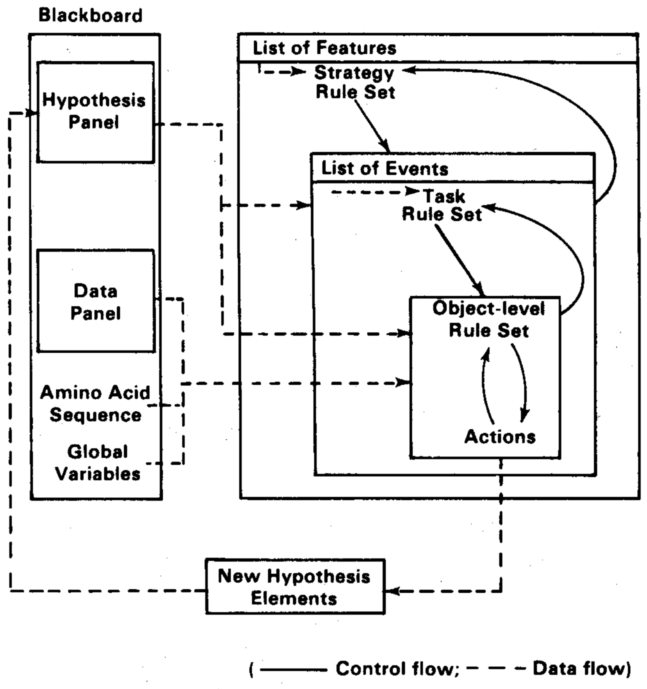
The Blackboard Structure. The blackboard contained two abstraction hierarchies called *blackboard panels.* The density panel contained four levels: the raw electron density map, the peaks, the skeleton, and segments. The information on this panel was produced by signal-processing algorithms prior to the interpretation process. The hypothesis panel contained atom, superatoms (amino acid residues and peptides), and stereotype (for example, alpha-helixes) levels. The objective was to place each atom of the protein molecule in the three-space represented on the hypothesis panel. The solutions were built by generating partial solutions on the hypothesis panel derived from data at any level in the density panel.

The Knowledge Source Structure. Each knowledge source consisted of a set of rules. Unlike the knowledge sources we have seen thus far, there were no preconditions associated with the knowledge sources. A precondition of a knowledge source served to inform the control module when it had something to contribute during the problem-solving process. The knowledge sources in CRYSALIS were not designed to be self-selecting.

Control. The CRYSALIS system used a three-tiered control structure. All the control modules were uniformly represented as knowledge sources. The overall control of the system was assigned to the strategy-level knowledge source. A set of strategy rules governed the choice of the next task to be performed on particular regions of the blackboard. A task was represented as a task-level knowledge source. Rules in the task knowledge source decided which object-level knowledge sources to execute within the context of a given strategy.

The strategy knowledge source had access to a summary of the solution state called a feature list, which recorded the state of the solution by regions. The strategy knowledge source decided upon which region to work and, based on the characteristic features of that region, selected and executed a task-level knowledge source. The selected task knowledge source executed a sequence of object-level knowledge sources based on the recent changes in the chosen region as recorded on the event list. The event list contained a list of changes made on the hypothesis panel (see Figure 10). After the execution of the object-level knowledge sources, control returned to the task knowledge source. The task knowledge source updated the feature list at this point and executed another sequence of object-level knowledge sources if the situation warranted. After the task-level knowledge source was finished, it returned control to the strategy knowledge source. The strategy knowledge source selected the next region on which to work and the appropriate task-level knowledge sources to reinitiate the processing. Only the object-level knowledge sources were allowed to modify the hypothesis panel, and no modifications were made to the density panel.

One can view the organization of the control component in one of two ways: (1) as a nested activation of the knowledge sources or (2) as an organization in which the precondition of each knowledge source was held within its immediate higher-level knowledge source. The higher-level knowledge source acted as the manager of the lower-level knowledge sources. In either case, the opportunistic application of the knowledge sources was less evident in CRYSALIS than in other systems. First, a large region was selected (by strategy), then a series of specific nodes within that region were selected (by task). Finally, a series of predetermined knowledge sources were executed to process each selected node. A task knowledge source remained in control until all the possible processing in a given region was exhausted. Once processed, a region was never revisited. The focus of attention consisted of subdividing a given region to find a solution island to process next.



CRYSALIS System Organization

Figure 10

In summary, the basic actions of the control component were the following:

1. The strategy knowledge source focused on a region of the blackboard based on information in the feature list and executed the appropriate task knowledge source.
2. The task knowledge source selected a specific place in the region and used it as a context for a sequence of object-level knowledge sources. The task knowledge source updated the feature list before returning control to the strategy knowledge source.
3. An object knowledge source modified the blackboard and returned control back to the task knowledge source.

Knowledge Application Strategy. The behavior of the CRYSALIS system was strongly island driven, or, more specifically, was directed at region growing on the blackboard. The following is a possible problem-solving scenario: Look for an electron-dense region that might indicate the presence of a tryptophan element (a large, ring-containing amino acid). Look in the amino acid sequence for the occurrences of tryptophan. Look to see if an adjacent region might be one of the neighbors of a tryptophan element in the the sequence. If the amino acid adjacent to the tryptophan matches the region in the close proximity of the tryptophan data, continue growing the region using the sequence as a guideline. When unable to continue with the region-growing process, look for another region to grow.

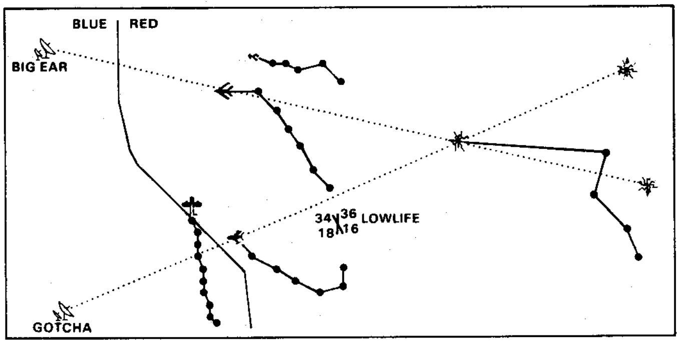
Given this problem-solving scenario, one can see the appropriateness of the CRYSALIS control scheme. The task-level knowledge sources, with names such as point-to-point trace, outward trace, split-group toehold, and so on, knew which object-level knowledge sources to call in order to accomplish their goals (in the scenario, the goal of the task is to extend the hypothesized region). The strategy knowledge source moved from one region to another, with each region demanding possibly different task goals. This type of nested processing was reflected in the hierarchy of the knowledge sources.

Within a region of interest, the selection of which node to process next was opportunistic. This was the only place that opportunism was exercised. The region selection followed the shape of the skeleton. The selection of the knowledge sources, both the task knowledge sources and object knowledge sources within each task knowledge source, was built in.

Additional Notes

- The quality of the density map affected the reliability of the knowledge sources. Each knowledge source had associated with it weights that could be adjusted to reflect the data quality. Furthermore, the rules were weighted according to an importance criterion. The weight on a hypothesis was a combination of the weights that reflected the data quality and the importance of the rule which generated the hypothesis.
- In HEARSAY-II and HASP, knowledge sources were self-selecting. That is, the precondition of knowledge sources determined whether the knowledge sources were appropriate in a given solution state. In CRYSALIS there was no counterpart to the precondition of knowledge sources.[14]
- As in HASP, the hypothesis panel in CRYSALIS was called the Current Best Hypothesis, and the nodes in the panel were called the hypothesis elements. In HASP, the CBH represented the situation board created, updated, and used by the analysts for further interpretation of the signal data. In CRYSALIS, the CBH represented the partial protein model built up to any given point in the model-building process. The hypothesis network on the blackboard represents a network of partial solutions, which is not necessarily the same as intermediate results. Whereas intermediate results often cannot stand on their own in the middle of a problem-solving process, partial solutions are often meaningful and useful on their own. Thus, if the CRYSALIS processing were to be interrupted and the Hypothesis panel examined, there would be solution islands that are acceptable solutions.

TRICERO



The TRICERO Task

Figure 11

The TRICERO system (Williams, Brown, & Barnes 1984; Williams 1985) represents an extension of the blackboard system into the area of distributed computing.[15] There are many possible ways to design a blackboard system that utilize multiple, communicating computers. To design a multiprocessor blackboard system, either the blackboard model or the blackboard framework can be used as a design foundation. What is chosen as the starting point will have a significant effect on the nature of the concurrency in the resulting system.[16]

[14] One wonders if CRYSALIS is truly a blackboard system in a strict sense, because it violates one part of the definitions of the blackboard model—knowledge sources respond to changes on the blackboard. As mentioned earlier, the control component of blackboard systems has disparate designs. However, the knowledge sources should, in order to maintain their independence, indicate the condition under which they can contribute to the problem-solving process. Because the knowledge sources in the CRYSALIS system were not designed so, I feel that the CRYSALIS control is a hybrid between a blackboard system and a rule-based system. It is a blackboardlike system.

[15] The TRICERO system was designed by Harold Brown of the Knowledge Systems Laboratory, Stanford University. It was built by programmers at ESL and Teknowledge. The TRICERO system was written using the AGE skeletal system (Nii & Aiello 1979). The distributed-system aspects of TRICERO were simulated.

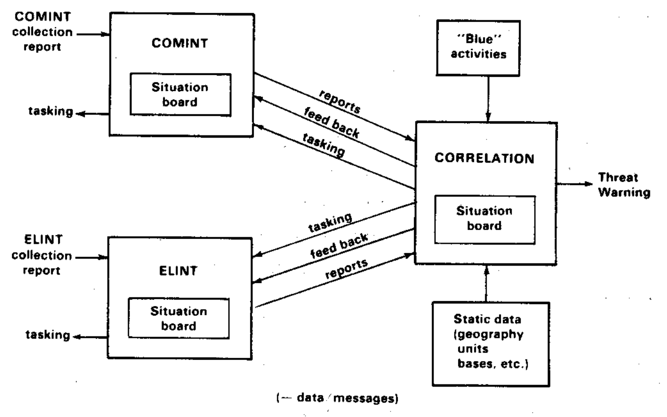
[16] Two parallel blackboard systems are currently being built at the Heuristic Programming Project. The design of one system is based on a fresh interpretation of the blackboard model. The system is designed to work within the context of a large number (100s to 1000s) of

Several possible ways exist for using multiple processors. The first is to partition the solution space on the blackboard into loosely coupled regions (for example, subregions of the ocean, parts of a sentence, pieces of the protein structure, and so on). For each of these partitions, create a copy of a blackboard system. For example, in HASP one might have a complete blackboard system for each sensor array. Because the arrays have overlapping coverage, the systems would have to coordinate their problem-solving activities. A system will notify an "adjacent" system if a platform is moving into that system's area, for example. In other words, the application problem can be partitioned into loosely coupled subproblems that need coordination. Research on this type of system is being conducted at the University of Massachusetts under the direction of Victor Lesser (Lesser & Corkill 1983).

A second way to use multiple processors is to place the blackboard data in a shared memory and distribute the knowledge sources on different processors (see Aiello [1986] and Ensor and Gabbe [1985] for examples). This distribution results in the parallel execution of the knowledge sources. If the knowledge sources are represented as rules, their condition parts can be evaluated in parallel. The action parts can also be executed concurrently with the evaluation of the condition parts in a pipeline fashion. The PSM project at Carnegie-Mellon University is targeted as a parallel rule execution system (Forgy *et al.* 1984).

Third, a more direct use of multiple processors can be accomplished by partitioning the problem into independent subproblems, where each subproblem is solved on a separate processor. For example, in the interpretation and fusion of multiple types of data, each type of data might be interpreted on different systems. Each system will have a different set of knowledge sources and different blackboard organization. The results from the data analysis systems will be fused by another blackboard system. The TRICERO system is an example of this type of system.[17]

The Task. The objective of the TRICERO system is to monitor a region of airspace for aircraft activities. The system consists of three subsystems organized in a hierarchy (two levels at this point), much like the human management organization for which the system was built (see Figure 12). On the lower level are the ELINT and COMINT subsystems that respectively interpret passive radar and voice communication data. The correlation system that integrates the reports from ELINT and COMINT and other data resides at a higher level. This hierarchical organization of blackboard systems emulates the various activities involved in signal understanding. These activities are signal detection, parameter estimation, collection analysis, correlation, and overall interpretation. As one progresses from one activity to another, information in the data is abstracted and reduced. TRICERO analyzed two types of collection data and correlated the analyzed data. Each data type was analyzed independently using different blackboard data organizations and different knowledge sources (see Figure 12).



A Distributed Blackboard System, the TRICERO Control

Figure 12

The Blackboard Structure. The ELINT blackboard consisted of three levels: observation, emitter, and cluster. The input data arrived at the observation level. These data were tagged with the collection time and the site at which they were collected. Each node on the emitter level kept a history of detections from a site having the same identification tag. The history represented radar emissions believed to be emanating from one source. The identification tag could be in error, whereby different sources could have the same identification tag, or one source could have multiple tags. The radar emissions detected at different sites were merged into a hypothetical platform (or a number of platforms "seen" as one platform) on the cluster level. Each level used descriptive vocabulary appropriate to that level: the platform types and speed history on the cluster level and the collection site and signal quality on the observation level, for example. The blackboard data

processor-memory pairs with high-bandwidth communication (Rice 1986). In the other system, designed as an extension to a serial skeletal system, parallel constructs are made available to the user. The design of this system is targeted for multiprocessor, shared-memory systems (Aiello 1986).

[17] The problems of designing and building blackboard systems capable of concurrent problem solving, distributed problem solving, and parallel computations are distinct from those of serial blackboard systems and are not discussed in this document. The TRICERO system is discussed here, because it does not fall into any of the earlier categories. It is a variant of a distributed computing system that can be considered a direct extension of the serial systems. See Lesser & Corkill [1983] for distinctions between distributed processing systems and distributed problem-solving systems.

structure in the COMINT and correlation subsystems were structured in similar fashions using abstraction levels appropriate to interpreting their data.

The Knowledge Source Structure. The knowledge sources were structured according to the specification in the AGE (Nii & Aiello 1979) skeletal system. Each knowledge source had a precondition part and an action part. The precondition part was a list of tokens, each representing a type of change that could be made on the blackboard. The action part consisted of a set of rules. The rules in each knowledge source could be processed as a multiple hit, in which all rules whose condition sides were satisfied were executed, or as a single hit, in which only the first rule whose condition side was satisfied was executed. There was no conflict-resolution process of the type found in OPS systems.

Control. Each of the independent subsystems in the TRICERO system used a subset of control components available in AGE. A globally accessible event list recorded the changes to the blackboard. At each control cycle, one event on an event list was chosen as a focus of attention. The choice of the event on which to focus was based on a predetermined priority of event types. Once an event (an event type and a node) was selected, it was matched against the event-type tokens in the precondition of the knowledge sources. Those knowledge sources whose preconditions contained the event-type token matching the focused event type were executed according to a predetermined priority of knowledge sources.

The TRICERO system augmented the AGE control component to handle the communication among the subsystems. Each subsystem could send messages to designated subsystems. The receipt of a message by a subsystem was treated as an event focused on a special node on the blackboard. This construct allowed the subsystems to treat reports from other subsystems just like any other event.

The basic actions of the control component can be described in two parts:

1. Between subsystems
   a. The simulation of the distributed computation consisted of round-robin execution of the three subsystems – ELINT, COMINT, and Correlation.
   b. Each subsystem sent report messages to designated subsystems. The receipt of a message was treated as an event with appropriate modification of the recipient's blackboard and event list.
2. Within a subsystem
   a. A control module selected a focus event using a list of event priorities. An event contained information about the event type of the change made to the blackboard, that is, the node on which the change was made, the knowledge source and the specific rule that made the change, and the actual change.
   b. Based on the focused event, knowledge sources whose precondition list contained the event were chosen for execution.
   c. The rules in the activated knowledge sources were evaluated and executed according to the rule-processing method associated with the knowledge source. Modifications to the blackboard by the rules were events and caused the event to be put on the event list.

Knowledge-Application Strategy. Most of the knowledge sources engaged in bottom-up processing. They combined information on one level to generate a hypothesis on a higher level. The reports from the correlation subsystem to ELINT and COMINT dealt primarily with information on the higher level (for example, platform identification) that overrode the analysis done by the lower-level subsystem. In such cases, the processing in these subsystems became top down. The reports from ELINT and COMINT were treated as input to the higher-level Correlation subsystem.

Additional Notes

- The partitioning of the overall task into subsystems in TRICERO was accomplished by assigning the analysis of the more abstract information to the correlation subsystem and the analysis of information closer to signal data to ELINT and COMINT. As mentioned in Part 1, the knowledge sources that span the various levels of the blackboard hierarchy are logically independent. Thus, the need for coordination among the subsystems is substantially reduced when the problem is partitioned into subsystems along carefully chosen levels of analysis.
- As with the other systems described, the TRICERO data were noisy and the knowledge sources uncertain. The radar data, for example, contained "ghosts," detections of nonexisting objects. The ELINT subsystem handled the existence of this type of error by delaying the analysis until several contiguous detections had occurred. By doing so, it avoided the creation of hypothesis nodes that later needed to be deleted.

  The issues relating to the deletions of nodes on the blackboard are quite complex. Suppose in TRICERO that a node on the cluster level (an object that represents a platform or a group of platforms) is to be deleted. What does it mean? Has the platform disappeared? Unless it somehow disintegrated, a platform cannot disappear into thin air. Was there an error in interpreting the radar data to begin with? Often, there are "ghost tracks," a characteristic of which is that the tracks disappear after a short duration. However, suppose the platform disappearance was not due to ghost tracks but to an error in reasoning. Unraveling the reasoning steps that led to the hypothesis—or backtracking—and retrying often do not help. The system does not know any more than it did when the erroneous hypothesis was generated. Suppose the platform node is just deleted. What do we do about the network of evidence that supports the existence of the platform? Unfortunately, there is no systematic way of handling node deletions. In HASP the nodes were never deleted. The nodes in error were ignored, and analysis continued ignoring past errors. In TRICERO node creation was delayed until there was strong supporting evidence for the existence of an object represented by the node. When an error occurred, the hypothesis network was restructured according to domain heuristics.

- In TRICERO the confidence assigned to the hypothesis elements was expressed in symbolic form. The vocabulary expressing the confidence consisted of "possible," "probable," "positive," and "was positive." The confidence level was changed according to heuristic criteria.
- TRICERO was one of the first blackboard systems implemented on a computer system with a bit-map display (see Figure 11 for a display output). The situation board, symbolically represented on the blackboard of the Correlation subsystem, was displayed in terms of objects in an airspace and the objects' past behavior. The graphic-display routines were written as procedural knowledge sources and were executed when certain events (changes on the blackboard) occurred that warranted display updates. There might be some argument about the conceptual consistency of this approach, because interfacing is usually not considered a part of problem solving. However, this engineering solution that integrated the display routines with the problem-solving components worked very well. An effective display interface requires knowledge about what is appropriate to display when. A knowledge-based control of displays and display updates is easily implemented using the knowledge source organization.

Other Blackboard Systems

The four application systems discussed thus far transformed signal data into symbolic forms "natural" to the task domain. The signal-to-symbol transformation occurred for the purposes of understanding the context in which the signals were present. There are other blackboard systems that deal with similar application problems. Unfortunately, many of these systems are either proprietary or classified. The descriptions of these systems lack technical details, and we were not able to include them for discussion. We have included references to articles describing some of these systems (Lakin & Miles 1984; McCune & Drazovich 1983; Spain 1983).

We now turn our attention to two blackboard application systems that have been built to address different types of tasks. A brief description of the task is followed by some notable features of these systems.

OPM

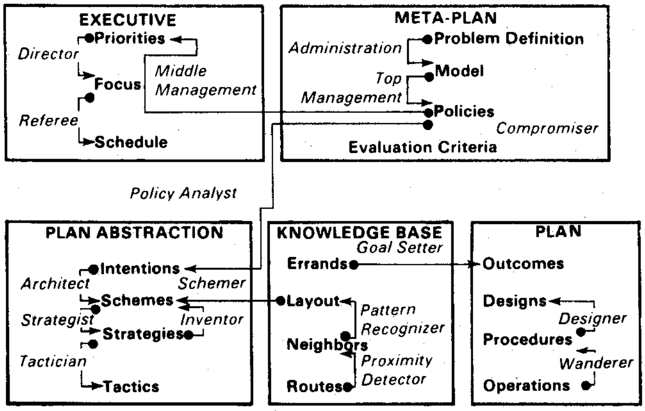
The OPM system (Hayes-Roth *et al.* 1979) differs from the systems described so far in that it is a simulation of a model of human cognitive processes in planning. The cognitive model was reflected in the architecture of the OPM system. Instead of generating a new plan, it replicated the planning process of human subjects. The fact that the system seemed to successfully model many subjects' planning processes indicated the validity of the model. It also attested "to the utility of the blackboard model as a general model of cognition." (Hayes-Roth *et al.* 1979)

In addition to the cognitive-modeling aspect, OPM demonstrated the generality of the blackboard model in the kinds of tasks it could address. The applications we have discussed so far dealt with interpretation tasks that are basically analytic in nature. The processing is primarily bottom up. The planning task is primarily generative in nature. It starts at the top with a goal to be achieved, and the planning process then produces lower-level sequences of actions to be performed. As we saw in both HEARSAY-II and HASP, top-down strategies were combined with bottom-up strategies to interpret noisy data, but nonetheless, the interpretation process was strongly data driven. The utility of the blackboard model for planning tasks opened up the possibility of building blackboard systems for many different classes of application problems.

The Task. The objective of OPM was to simulate human errand-planning protocols. "The planner begins with a list of desired errands and a map of a town in which she or he must perform the errands. The errands differ implicitly in importance and the amount of time required to perform them. The planner also has prescribed starting and finishing times and locations. Ordinarily, the available time does not permit performance of all of the errands. Given these requirements, the planner decides which errands to perform, how much time to allocate for each errand, in what order to perform the errands, and by what routes to travel between successive errands." (Hayes-Roth *et al.* 1979)

The Blackboard Structure. The blackboard was partitioned into five planning panels, called *planes* containing, conceptually different categories of decisions. Each panel contained several levels of abstraction found in the planning space. The five panels were (see Figure 13).

- *Metaplan:* Decisions on this panel indicated what the planner intended to do during the planning process.



OPM Blackboard and Knowledge Sources

Figure 13

For example, on the policies level, a knowledge source specified general criteria to impose on the problem solution, such as "the plan must be efficient" or "minimize certain risks."

- *Plan Abstraction:* Decisions on this panel characterized desired attributes of potential plans. These abstract decisions served as heuristic aids to the planning process, suggesting potentially useful qualities of the planned actions.
- *Knowledge Base:* This panel recorded observations and computations about relationships in the world, that the planner generated while planning. This knowledge supported two types of planning functions—the analysis of the current state of affairs and the analysis of the likely consequences of hypothesized actions. For example, at the errand level, the planner might have computed the time required to perform all of the currently intended errands in order that the planner might evaluate the plan's gross feasibility.
- *Plan:* Decisions on this panel indicated actions that the planner actually intended to take. Decisions at each level within this panel specified a more refined plan than those at the adjacent higher level. For example, the outcomes level indicated what the planner intended to accomplish by executing the final plan, whereas the procedures level specified specific sequences of actions (errands).
- *Executive:* This panel contained information related to control. The knowledge sources on this panel decided which of the invoked specialists were to be executed. The decisions were based on information on the different levels representing different types of "executive decisions." For example, priority decisions indicated a preference for allocating processing activity to certain areas of the planning blackboard before others. Schedule decisions indicated which of the invoked specialists satisfying higher-level decisions to execute next.

The Knowledge Source Structure. The knowledge sources were called *specialists.* Each specialist consisted of two components, condition and action, as in HEARSAY-II. The condition part consisted of a trigger and a test. The *trigger* provided a quick preliminary test of a specialist's relevance for any focused node. The *test* specified all other prerequisites of applicability. For a given focus node, the triggers of all specialists were checked. Test parts were evaluated for those specialists whose triggers were satisfied. A specialist became "invoked," as in HEARSAY-II, when both the trigger and the test parts of the condition were satisfied.

The specialists in OPM were written as procedures, as in HEARSAY-II. However, the procedures were much smaller and represented a smaller grain of knowledge.

Control. The OPM system had four global data structures: a map on which the errands occurred, the blackboard that contained the partial solutions, an agenda that held a list of invoked specialists which needed to be scheduled for execution; and an event list that recorded the history of changes to the blackboard.

The basic actions of the control component consisted of three phases that were repeated:

1. During the invocation phase. the test part of the specialists was evaluated. Specialists whose tests were satisfied became "invoked." The program terminated when there were no invoked specialists.
2. In the scheduling phase, one of the invoked specialists was recommended for execution. The basis of the recommendation was recency of invocation and current focus. The focus node was the most recently added or modified node on the blackboard. A specialist was chosen whose action, if executed, would occur in the region of the focused node.
3. In the execution phase the scheduled specialist was executed. The program immediately evaluated the trigger of all specialists against the *focus node* (the one just changed) and added those specialists whose triggers were satisfied on the agenda.

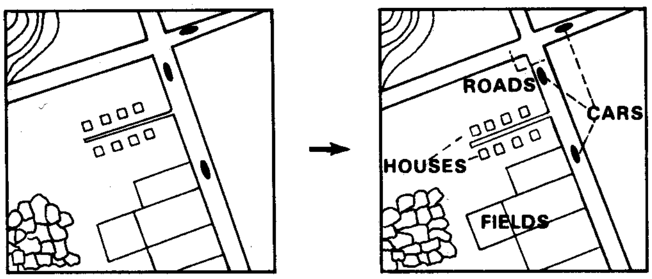
Knowledge-Application Strategy. The objective of the system was to enable the simulation of diverse problem-solving behavior exhibited by human subjects while planning (Hayes-Roth *et al.* 1979). As can been seen from the design of the control modules, however, the basic strategy was, for psychological reasons, to follow up on the most recent actions and in the geographic proximity of those actions. The psychological reason is not explained.

However, one can surmise that at least for the errands task if a person decides to do an errand in one place, then that person will do all the errands in the same area.

Additional Notes

- The OPM design reflects the first step taken to separate and make independent the problem of control. A scheduler of the HEARSAY-II variety was encoded as knowledge sources. Each of these control knowledge sources had a specialized scheduling policy; for example, go to the next closest place, or do the next most important errand. The information needed and generated by the control knowledge sources was stored on a special blackboard panel—the *executive panel.* The executive panel was isolated from the other panels in that changes on the executive panel were not treated as events which affected the triggering of other knowledge sources; changes only affected the selection of the knowledge sources to be executed.
- The map data resided outside of the blackboard data structure. It might have been interesting to organize the map on a blackboard panel in a similar manner to the CRYSALIS' density panel. One could then have modeled the generation of abstract plans and strategies from abstracted maps.

Scene Understanding.



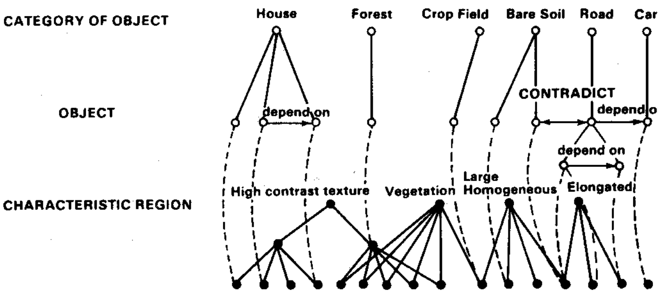
The Scene Understanding Task

Figure 14

As mentioned in Part 1, one stimulus for the HEARSAY project was a desire to integrate syntactic and semantic information into the understanding of speech utterances. The notion of semantically driven vision programs had also existed for some time. A program to interpret complex aerial photographs developed by Makoto Nagao and Takashi Mastuyama (Nagao, Matsuyama, & Mori 1977), was the first to address this problem using the blackboard approach.

The Task. Given a large aerial photograph of a complex suburban area taken at low altitude, the program is to identify and label objects in the photograph (see Figure 14). The variety of objects to be identified includes, among other things, cars, houses, rivers, and roads.

The Blackboard Structure. The blackboard was organized into an abstraction hierarchy consisting of elementary regions, characteristic regions, and object levels (see Figure 15). The lowest level, the elementary regions level, contained regions segmented according to multispectral properties. The attributes of each elementary region were its average grey level, size, location, and basic shape, together with pointers to the digitized picture indicating its relative position. A combination of elementary regions formed an object on the characteristic regions level. On this level, seven characteristic features were extracted: large homogeneous regions, elongated regions, shadow regions, shadow-making regions, water, vegetation, and high contrast-texture regions. The objects from this level were classified into one of the domain objects the system knew about—cars, houses, rivers, roads, crop fields, and so on.



Blackboard and Knowledge Sources

Figure 15

The Knowledge Source Structure. A knowledge source was represented as a single rule. Thus, the condition part of the rule served as the precondition of knowledge sources. Each rule held enough knowledge to recognize one object. Because there are many different ways to recognize an object, there were multiple rules (knowledge sources) for the recognition of a specific type of object.

Control. All knowledge sources whose condition sides were satisfied by the data in the blackboard were executed, as in HASP. Thus, no complex scheduling module was needed. However, because knowledge sources might interpret a region differently (for example, as a crop field or grassland), the system incorporated a mechanism to resolve this type of contradiction. Called the *conflict resolution mechanism*, it calculated the "reliability value" of each region interpretation and retained only the most reliable interpretations.

Because applying each object-detection knowledge source to every region would be computationally expensive, the knowledge sources were applied only to those regions with certain characteristics. These characteristics were precisely the attributes of the objects on the characteristic-region level. Thus, the preconditions of the knowledge sources served as a filter that kept the knowledge sources from processing each region. The authors called this process the *focusing mechanism*. It emulates the way in which a human first globally surveys a scene to find prominent features that attract interest before doing detailed analysis.

Knowledge-Application Strategy. Each object category in the system was specified *a priori* and had associated with it knowledge sources that could recognize or reject an object in the category. For example, there was a "house expert" that, given a region, could recognize it as a house or could reject the possibility of it being a house. The knowledge sources were further specialized into data-driven knowledge sources and model-driven knowledge sources. Data-driven knowledge sources were capable of combining objects on the characteristic-region level into identifiable domain objects. The model-driven knowledge sources interpreted regions on the characteristic-region level based on already identified objects. For example, cars could be easily identified once a road had been identified.

The object-recognition process thus worked both bottom up and top down. Aside from the model-driven aspect, the system employed another form of top-down processing not found in the systems discussed thus far. If the object-detection knowledge sources were unable to detect an object (that is, unable to label a region as an object), the system reapplied segmentation subsystems in order to split the region or to merge it with adjoining regions. The newly formed region was then processed again by object-detection knowledge sources.

Additional Notes. The integration of symbolic reasoning and numeric (or algorithmic) computation is known to be important, especially the feedback of information from the symbolic side to the numeric side. A push from the symbolic side basically amounts to top-down processing, whether it be for a model-driven analysis or for a goal-directed analysis. Nagao and Matsuyama's system is the first documented system we could find that actually accomplished the integration. In blackboard systems, the integration of symbolic and numeric processes would appear simple—one only needs to treat a numeric algorithm as another knowledge source. One reason for more systems not having this integration, for example in HASP or CRYSALIS, is that the integration must be planned from the beginning and the mathematical algorithms written to fit the plan. Neither usually happens.[18] For example, the algorithms that would have been useful in CRYSALIS were not written to serve as knowledge sources. One requirement for a knowledge source is its ability to deal with partial solutions or data. Most algorithms are designed to process the whole data. In addition, for the algorithms to be used effectively, other knowledge sources must be able to set parameters in the algorithm. For example, a knowledge source might ask an algorithm to increase the data-sampling rate to see if the information content in the algorithms' output can be increased or improved. The effective utilization of numeric algorithms within a blackboard system is a knowledge-based task.

Summary

A definition of the blackboard model and a summary of blackboard system designs, in the form of the blackboard framework, were provided in Part 1. The first part of this article reviewed some of the older application systems. Although a problem-solving model can help in the general organization of domain knowledge and reasoning strategy, the blueprint of the architecture is drawn by the characteristics of the specific task at hand. Details of the task determine the specific choice of knowledge representation and reasoning methods. It is possible, therefore, that there are as many blackboard architectures as there are applications. Figure 16, which summarizes a small set of blackboard systems, indicates variations in the characteristics of signal-understanding problems and variations in the blackboard systems designed to solve the problems. To what degree the differences in the design, especially in the design of the control component, can be attributed to the differences in the problem is hard to determine at this point. At the same time, the figure indicates that complex problems can be solved using the blackboard problem-solving organization and that the basic organization allows for a wide range of variations in system designs.

Blackboard Systems from a Knowledge Engineering Perspective

There are many blackboard or blackboardlike systems being developed that still need to be analyzed.[19] Instead of a conclusion, this section contains one knowledge engineer's observations about the advantages and drawbacks of using a blackboard approach for building expert systems.

[18] This is not surprising. There is a "cultural" gap between people who build numeric algorithms and those involved in symbolic reasoning. Each group would like to solve a given problem within their own discipline. However, there is a great deal to be gained by combining the strength of both sides.

[19] We solicit builders of blackboard systems to provide data so this table can be expanded.

	HEARSAY-II	HASP	CRYSALIS	TRICERO	OPM	ACAP*
Application Characteristics						
Task	speech understanding	sonar signal understanding and information fusion	electron-density map interpretation	military signal understanding and information fusion	errand planning	image understanding
Continuous data? (evolutionary solution)	no	yes	no	yes	no	no
Signal-to-noise ratio	moderate/high	low/moderate	low	moderate/high	N.A.	moderate/high
Uses solution-space generator?	yes	no	no	no	no	no
Data reliability	moderate	moderate	high	moderate	high	moderate
Partial solution evaluation?	yes	no	yes	no	yes	no
Belief revision?	no	yes	no	yes	yes	yes
Multiple hypotheses?	yes	no	no	no	no	no

All the application tasks have (1) big factorable solution space, (2) interactive subproblems, (3) multiple lines of reasoning, (4) heterogeneous domain vocabulary, (5) no fixed sequence of subproblems to be solved. (Expert-system characteristics from Hayes-Roth, et al., 1983, p. 91)

	HEARSAY-II	HASP	CRYSALIS	TRICERO	OPM	ACAP*
System Characteristics						
Number of panels and levels	1/7	1/6	2/3	3/4†	5/4†	1/5
Knowledge source condition (form)	procedure	event name	none (imbedded in control modules)	list of event names	procedure	condition part of a rule
Knowledge source body (form)	procedure	set of rules	set of rules	set of rules	procedure	action part of a rule
What determines an event?	monitor	each KS	control KS	each KS	monitor	monitor
Focus of attention	a KS	a blackboard node	a blackboard node and a KS	a blackboard node	a KS	a KS and blackboard data
What determines the focus of attention?	scheduler	event manager	control KS	priority manager	scheduler	[predetermined]
Primary processing strategy	generate and test with constraints bottom-up generation and test; top-down and middle-out constraint generation	inductive synthesis combined with model-derived expectations bottom-up synthesis and top-down expectation generation	model-guided synthesis and region growing model generation from auxiliary data and middle-out region growing	inductive synthesis and correlation bottom-up synthesis and high level correlation	generate and test with refinement top-down plan generation and bottom-up refinement	feature extraction and model-driven feature match bottom-up feature extraction and top-down match

*Analysis of complex aerial photograph

†Average number of levels

Figure 16. Summary Characteristics of the Application Systems

[This table was developed by Harold Brown, Bob Engelmore, and Penny Nii]

When should the use of a blackboard model be considered by a knowledge engineer? A general guideline is that a blackboard approach is useful for complex, ill-structured problems.

Complex Problems. Simon (1969) defines a complex system as "one made up of a large numbers of parts that interact in a nonsimple way. In such systems, the whole is more than the sum of the parts, [in the sense that] given the properties of parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole." In order to understand complexity, we describe complex systems in terms of subsystems and relationships between the subsystems that are less complex. Often, this description takes the form of a hierarchy.

In software engineering, there are techniques and methodologies that foster hierarchic problem decomposition, and most complex programs are organized according to some form of hierarchy. Usually, the hierarchy is organized along functional decompositions of the task to be performed. This organization has the advantage of allowing common functions to be shared by many subsystems. In blackboard systems, a problem is decomposed to maximize the independence of the subsystems. Functionally, subsystems that generate and fill the solution space are separated from subsystems which determine their utility. All of these subsystems (knowledge sources) can be organized into a control hierarchy, but the emphasis is on limited interactions between the subsystems. This allows for maximal flexibility in the software development phases as well as during the problem-solving phase.

The medium of interactions among the subsystems (the blackboard) is also organized hierarchically. The hierarchical organization of the solution space on the blackboard has pragmatic advantages in designing a solution to a problem. First, the hierarchical structure allows for the integration of diverse concepts and associated vocabularies. For example, in HASP the concept of a "platform" was defined with properties such as type, speed, and location; the concept of "signal" was defined with properties such as frequency, intensity, and bandwidth. Second, the abstraction of information reduces the computational need in two ways: (1) The manipulations of abstract entities involve manipulation of smaller numbers of entities than manipulations of their detailed counterparts and (2) abstractions can store information that would otherwise need to be recomputed from the detailed counterparts. Put another way, the hierarchical structure of the blackboard provides a favorable trade-off between storage space and computational time. For example, in CRYSALIS, reasoning with an amino acid as an object in its own right is easier and faster than reasoning at the level of its atomic constituents, even though extra storage is needed to represent the amino-acid object.

Ill-Structured Problems. Ill-structured problems (Newell 1969) are characterized by poorly defined goals and an absence of a predetermined decision path from the initial state to a goal. Often, there is a lack of well-defined criteria for determining whether a solution is acceptable.[20] Being ill structured is sometimes intrinsic to a problem; for example, sculpt a masterpiece. At other times, a problem is ill structured because it is ill defined or ill-understood—for example, assess the merits of one's financial investments in the light of the proposed tax reforms.[21] Although Newell's discussions about ill structuredness occur within the context of weak problem-solving methods, he notes that a human's ability to solve an ill-structured problem might be due to the problem solver's ability to "recognize the essential connection or form of the solution" or due to the fact that "the problem solver always [has] available some distinctions that apply to every situation (Newell 1969)." In short, many ill-structured problems might be solved by applying knowledge, especially knowledge in the form of empirical associations, or expertise.

Many of the current expert systems deal quite well with ill-structured problems. What further aid can the blackboard approach provide? First, the blackboard approach requires no *a priori* determined reasoning path. Because ill-structured problems often do not have a predetermined decision path to a solution, the selection of what to do next must be made while the problem is in the process of being solved. The incremental and opportunistic problem-solving approach in blackboard systems provides the capability to do precisely that. Second, from a knowledge engineering viewpoint, vague information and knowledge, which characterize ill-structured problems, need to be made concrete in the process of finding a solution to the problem. The blackboard model is an excellent tool for this knowledge engineering activity. (The blackboard model as a problem-formulation tool is discussed in the next section.) The blackboard approach is also an excellent tool for exploratory programming, a useful technique for developing complex and ill-structured problems and is discussed in the section on the use of the blackboard model as a development tool.

Although useful for many complex, ill-structured problems, blackboard systems are generally expensive to build and to run. It would be foolish to apply the blackboard approach when lower-cost methods will suffice. For example, classification problems (Clancey 1985) can in principle be solved using the blackboard method, but there

[20] In many expert systems, the acceptability of a solution is determined by a panel of human experts who might disagree among themselves.

[21] It appears that some problems are more ill structured than others. Art making has traditionally been considered a very ill-structured task. However, the AARON program makes "freehand" drawings. (Cohen 1979) It's interesting that AARON is a blackboard-like system.

are lower-cost approaches to the problem. Determining the appropriate problem-solving methodology for an application problem is itself a difficult problem and is not one of the topics of this paper. The reader is referred to Hayes-Roth, Waterman, & Lenat (1983); Weiss & Kulikowski (1984); and Kline & Davis (1985) for some guidance. Generally, the occurrence of some combination of the following characteristics in a problem makes it an appropriate candidate for the blackboard approach:

- A large solution space
- Noisy and unreliable data
- A variety of input data and a need to integrate diverse information
- The need for many independent or semi-independent pieces of knowledge to cooperate in forming a solution
- the need to use multiple reasoning methods (for example, backward and forward reasoning)
- The need for multiple lines of reasoning
- the need for an evolutionary solution

The Blackboard Model As a Problem Formulation Tool

During the preliminary knowledge engineering phase, the goal of the knowledge engineer is to understand the task domain and the objectives of the proposed system. A knowledge engineer needs a set of conceptual models for organizing knowledge and reasoning. During the initial interactions with an expert, a knowledge engineer tries to find an appropriate conceptual model for the task while trying to understand the domain and the nature of the task. Often, the understanding of the task occurs with the help of a conceptual model. Because the information provided by the expert is rarely organized to fit a particular problem-solving model, the knowledge engineer initially needs a model with flexible methods for representing and applying pieces of knowledge.[22] Many of the issues faced by a knowledge engineer are the same as those faced by a traditional software engineer. As a knowledge engineering tool, the blackboard approach is useful because it provides some organizational principles that are both powerful and flexible.

Partitioning the Knowledge. To make complex problems manageable, they often need to be decomposed into loosely coupled subproblems. As mentioned earlier, one useful decomposition tool is the partitioning of the solution space into a hierarchy. Hypothesizing the objects and their relationships is accomplished by applying knowledge. If the blackboard hierarchy is organized correctly, then the knowledge sources that operate between the levels in the hierarchy should be more or less self-contained. That is, a knowledge source should function much like a specialist, requiring little information from levels other than the one in which it is expert. For example, a lexicographer should not need much information from a phoneticist. Put another way, the system should have the behavioral characteristics of a nearly decomposable system, as described by Simon (1977)—there should be less communication among the subsystems (knowledge sources in this case) than communication within each subsystem.

Separating Knowledge and the Uses of Knowledge. A piece of knowledge can be used for many purposes. For example, knowledge about statistical methods can be used for processing speech, sonar, X-ray, radar, and visual signals. Whether a particular method, for example a least squares method, is useful depends on what it is to be used for, on the goals of the application problem, and on the specific situation that arises while solving the problem. Thus, it is useful for a knowledge engineer to have a model that explicitly separates knowledge and the when, where, and how of applying the knowledge. The blackboard organization encourages the designer to make these separations. In addition, in a blackboard system decisions as to when, where, and how a piece of knowledge is to be applied are made dynamically. The knowledge engineer can thus design a problem-solving strategy or set of strategies that best exploits the state of the solution. The separation also allows the knowledge to be expressed, at least in principle, in a "pure" form, unencumbered by information on how or when or where it is to be used. If the application of knowledge is in itself a complex task, then the blackboard model allows for the control to be encoded as knowledge sources that specialize in reasoning about knowledge application. This capability provides another dimension of organizational flexibility.

Errorful Data. Both noisy input data and unreliable knowledge can result in erroneous partial solutions on the blackboard. Traditionally, uncertainty in the knowledge has been solved by assigning credibility weights to the knowledge and to the information it generates. The blackboard approach provides an additional methodology for coping with errorful data. The basic idea is to establish independent evidential support and reasoning paths to a solution. Many different kinds of data can be used to generate a hypothesis; for example, both intelligence reports and signal data were used in HASP. In addition, both HASP and TRICERO, whose input data were frequent observations of the same scene, exploited information redundancy in the data. In addition to the use of diverse sources of information, different expertise (competing knowledge sources) can be used. In HEARSAY-II both the syntactic-semantic hypothesizer and word-candidate

[22] Often, it is useful to use the blackboard model as a conceptual tool for understanding the task. Once the task is well understood, it can be reformulated for a simpler and less expensive problem-solving method.

generator were experts in generating the next word in a sentence. In HEARSAY-II, HASP, and other blackboard systems, hypothesized partial solutions had reasoned supports from above and supports from below to compensate for erroneous data and uncertain knowledge. The ability to use different resources, whether they be additional data or knowledge, to address the data error problem is important. The advantage of the blackboard approach is that organizationally, at least, it does not preclude the use of any of these resources.

The Blackboard Model as a System Development Tool

In traditional programming practices, system building consists of determining the requirements, designing a system, and implementing and testing a program. A programmer works from detailed system specifications. Almost all current software engineering techniques (for example, structured programming) are designed to ensure that the implementation strictly follows the specifications. The test phase consists of checking that the program performs according to the specifications. One aspect of dealing with complex and ill-structured problems is that there is often no *a priori* specification for a system. An additional difficulty with designing and implementing expert systems is that at least two parties are constantly shifting their points of view: the domain expert and the knowledge engineer. As the knowledge in the program accumulates and the problem becomes clearer, the knowledge engineer might find better ways to represent and process the knowledge. The resulting behavior of the program may inspire the expert to shift his view of the problem, creating further problems for the knowledge engineer to solve. Consequently, a knowledge engineer needs to engage in exploratory programming. *Exploratory programming* as defined by Beau Sheil, is "a conscious intertwining of system design and implementation."[23] (Sheil 1983).

Waterman (1985) notes that expert system building is accomplished in developmental stages ranging from research prototype to demonstration prototype to field prototype and so on until a fielded system is evolved. That is, expert systems are also developed incrementally. At each development stage, each of the system building phases (design, implement, and test) is repeated. The test phase is different from the traditional approach in that what is being tested is the overall acceptability of the behavior of the system, and not the adherence to specification. At each development phase, the expert system might have to be redesigned and rebuilt.

In light of a need for exploratory and incremental system development, what is desired is a robust system organization that allows for modifications and additions at each development stage with minimum perturbation to extant structures and code. The HEARSAY-II and HASP systems went through various changes without major overhauls. In HEARSAY-II, various configurations of knowledge sources and blackboard levels were tried, parallel constructs were added to run on multiprocessors; and various scheduling schemes were experimented with. The HASP system was converted from an interpreted system to a compiled system, the sonar data were changed from one form to another; and new levels on the blackboard and knowledge sources were added to extend the scope of the task. Each change in these systems required very little work relative to the magnitude of the changes. The robustness of blackboard systems lies primarily in the organization of the systems which tends to localize changes. Appropriate modularization of the solution space and knowledge into modules that require little intermodule communication ensures that changes are locally confined. Changes in the problem-solving behavior (knowledge-application strategy) are confined to the control component. Additions and modifications to the knowledge base involve additions of new knowledge sources or are confined to additions and modifications within the existing knowledge sources.

The blackboard organization is also being used as a development tool for many programs for several closely related reasons.[24] First, the blackboard is used primarily as a means of communication between disparate processing modules. The information being communicated is based on the task semantics. Thus, if a signal processor produces a radar track in terms of a vector and a covariance, the information is placed on the blackboard in terms of coordinates, a heading, a speed, and an error estimate. This form of information is process and data-structure independent and can be used by other knowledge sources.

Second, the blackboard approach allows the postponement of design decisions. For example, in developing a complex, robot navigation system with multiple sensors, a system is needed that allows experimentation with different sensors which interact differently with each other. Until an appropriate combination of sensors is found, the system needs to be open ended.

Third, the blackboard approach provides freedom from message-passing constraints. The message-passing paradigm requires a recipient of a message as well as a sender. Often, the recipient is not known, or the recipient might have been deleted. In the blackboard approach, the message-sending module places the information on the blackboard, and the developer of the module is freed from

[23] Sheil's article contains an excellent discussion on the necessity and the dimensions of exploratory programming.

[24] This is a partial summary of discussions held at the Blackboard Workshop on 12–13 June 1986 at Carnegie-Mellon University. The attendees were primarily representatives from vision and robotics projects. It appears that for these projects at this point in time the blackboard approach is being used primarily as a software engineering tool.

worrying about other modules.

Not all of this is good news. If the initial organization of the blackboard data (and indirectly the organization of the knowledge sources) is wrong, then modifications result in a rapid deterioration of the system structure. Where can things go wrong? Given a task domain, there are many ways to partition its solution space into a hierarchy. If the task (a goal to be accomplished within the domain) is misunderstood, then one can end up with an inappropriate hierarchy on the blackboard. A knowledge engineer will be building a system with a wrong model of the world. Unfortunately, it often takes a long time to discover that one's perspective on the problem is wrong. To aggravate the situation, a clever programmer can go a long way to make things work, even knowing that things will get progressively worse as more knowledge is added. In such a situation, changes occur too late, making a major overhaul inevitable. This type of headache can sometimes be avoided by employing an incremental development strategy, which is easily accomplished with the blackboard system organization.

Blackboard systems can be built in a top-down fashion similar to a standard software engineering technique. A small amount of knowledge can be encoded for each knowledge source, and a simple control component can be constructed to test the overall behavior of the system. This is particularly easy to do if the knowledge sources contain rules because it is easy to implement a few rules for each knowledge source. This approach is in contrast to an approach in which one knowledge source is completed before the next knowledge source is developed.[25] An advantage of this top-down approach is that gross errors, misunderstandings, or inadequacies in the implementation can be discovered quite early. The approach is analogous to the progression of prototypes leading to a fieldable system that was discussed earlier. The increments of development are much smaller, and it permits better internal control of the development process.

The Blackboard Model as a Research Tool

The blackboard model can be a useful tool for conducting research in applied artificial intelligence. The solution space of the application problem and the domain knowledge can be partitioned in many ways and a variety of reasoning strategies can be experimented with. The blackboard model itself is also the focus of some current research projects:

- BB1: The work on the BB1 skeletal system addresses the problem of rationalizing the control component. The problem of when and how to apply domain knowledge is viewed as a separate task. This control task is organized as a separate blackboard system. To solve a problem, BB1 alternates between formulating and executing the problem-solving strategy on one blackboard system and applying the domain knowledge in another (Hayes-Roth 1985).
- Distributed Problem Solving: This research addresses the problems of obtaining a solution and maintaining coherent problem-solving behavior in distributed systems that have a common goal. One of the major research issues in distributed knowledge-based systems is the distribution of control—how much and what kinds of global control are required, and how much control can remain local. This and other issues are explored within the context of three application domains: distributed interpretation, distributed network traffic-light control, and distributed planning (Lesser & Corkill 1983).
- Concurrent Problem Solving: This research explores the feasibility of using the blackboard model as a basis for developing frameworks for concurrent problem solving. Two experimental frameworks are being developed to explore parallel constructs, one for shared-memory, multiprocessor systems and one for distributed-memory, multi- processor systems. Major issues are: finding and expressing parallelism in application problems, determining the optimal grain size of data and knowledge for maximal processing speed; and, as in the distributed problem-solving problem, determining the balance between local and global controls (Nii 1986).

In addition to this research, it might be interesting and useful to explore the utility of the blackboard model in the area of design and machine learning. Designing, whether designing a piece of jewelry or a very large scale integration circuit, is an opportunistic process and is accomplished at various levels of abstraction. Design problems also have large solution spaces and require many, diverse cooperating sources of knowledge. Design seems a good candidate for a blackboard application. The major difficulty with design problems might lie in our limited understanding of how to represent and reason about spatial relationships.

In the case of machine learning, the blackboard model might serve as a model of learning. The learning process can be viewed as incremental (acquiring a piece of knowledge at a time), opportunistic (recognizing when something is worth remembering), and hierarchical (knowledge ranging from detailed empirical association rules to general rules). A learning system might consist of two blackboard systems. One system, which solves a problem, would consist of the standard blackboard configuration. The second system, which learns from the behavior of the first system, would consist of a blackboard containing a hypothe-

[25] I have often seen development of a knowledge source that proved unnecessary or unimportant when it was used in conjunction with other knowledge sources. One might argue that a wrong system decomposition was chosen or that the interactions among the subsystems were not fully explored. More often than not, however, the perspective on the problem and on the problem-solving strategy change once all the knowledge sources are in place.

sis about how the problem is being solved, and its knowledge sources would consist of diverse methods of learning. Whether the learning subsystem performed well or not can be tested by executing the acquired knowledge on the problem-solving blackboard. This particular approach would require extensions to the current blackboard organization. These extentions pose little difficulty, however, because the power of the blackboard system organization lies in its modularity, flexibility, and robustness.

Acknowledgements

I have been waiting a long time for someone to write a comprehensive article on blackboard systems. I finally decided to give it a try, and it turned out to be a bigger job than I expected. The skeletal blackboard systems still need to be reviewed. Without the help of many friends and colleagues, this article would not be, and I would like to thank them all here. Ed Feigenbaum has been supporting a variety of research related to blackboard systems for many years. Without his support, many of the systems mentioned in this paper would not have been built. Bob Engelmore appeared every Tuesday morning at 8:30 to check on my progress in writing, to comment on my approach, and to discuss various aspects of blackboard systems. The many hours of discussion with Harold Brown on wide-ranging topics clarified my thoughts about the blackboard model and blackboard systems. In addition to Bob and Harold, James Rice, John Delaney, and Peter Friedland helped make the article readable. I also want to thank Herbert Simon, who gave me pointers to the earlier work, and Lee Erman, who patiently explained the details of the HEARSAY-II design. Although I have tried to be neutral with respect to the many aspects of blackboard systems, sometimes this was a difficult goal to achieve. Any biases, misrepresentations, and associated blame are mine alone.

References

Aiello, N. 1983. "A Comparative Study of Control Strategies for Expert System: AGE Implementation of Three Variations of PUFF." In *Proceedings of the National Conference on Artificial Intelligence.* Menlo Park, California: American Association for Artificial Intelligence, 1–4,

Aiello, N. 1986. *User-Directed Control of Parallelism: The CAGE System.* Tech. Rep. KSL Report 86-31, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Brown, H. & J. Buckman. 1982. *Final Report on HANNIBAL.* Tech. Rep., ESL, Inc.

Clancey, W. J. 1985. *Heuristic Classification.* Tech. Rep. KSL-85-5, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Davis, R. Davis & J. King. 1977. "An Overview of Production Systems." In E. W. Elcock & D. Michie eds., *Machine Intelligence 8: Machine Representation of Knowledge.* New York: John Wiley.

Ensor, J. R. & J. D. Gabbe. 1985. "Transactional Blackboards." *Proceedings of the 9th International Joint Conference on Artificial Intelligence.* Los Altos, California: William Kaufmann, Inc., 340–344.

Erman, L. D., F. Hayes-Roth, V. R. Lesser, & D. Raj Reddy. 1980. "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty." *ACM Computing Survey* 12: 213–253.

Erman, L. D., P. E. London, & S. F. Fickas. 1981. "The Design and an Example Use of HEARSAY-III." In *Proceedings of the 7th International Joint Conference on Artificial Intelligence.* Los Altos, California: William Kaufmann, Inc., 409–415.

Forgy, C., A. Gupta, A. Newell & R. Wedig. 1984. "Initial Assessment of Architectures for Production Systems." *Proceedings of the National Conference on Artificial Intelligence.* Menlo Park, California: American Association for Artificial Intelligence, 116–119.

Hayes-Roth, F., & V. R. Lesser. 1976. *Focus of Attention in a Distributed Logic Speech Understanding System.* Tech. Rep., Computer Science Department, Carnegie-Mellon University.

Hayes-Roth, F., D. Waterman, & D. Lenat eds. 1983. *Building Expert Systems.* Reading, Massachusetts: Addison-Wesley.

Hayes-Roth, B. 1985. "Blackboard Architecture for Control." *Journal of Artificial Intelligence* 26: 251–321.

Hayes-Roth, B. *et al.* 1987. "Elucidating Protein Structure from Constraints in PROTEAN." In R. Engelmore & A. Morgan eds. *Blackboard Systems: Applications and Frameworks.* Addison-Wesley Publishers, Ltd., in preparation.

Hayes-Roth, B., F. Hayes-Roth, F. Rosenschein, & S. Cammarata. 1979. "Modelling Planning as an Incremental, Opportunistic Process." *Proceedings of the 6th International Joint Conference on Artificial Intelligence.* Los Altos, California: William Kaufmann, Inc., 375–383.

Kunz J, R. Fallat, D. McClung, J. Osborn, B. Votteri, H. P. Nii, J. S. Aikins, L. Fagan, & E. A. Feigenbaum. 1978. *A Physiological Rules Based System for Interpreting Pulmonary Function Test Results.* Tech. Rep. HPP-78-19, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Lakin, W. L., & J. A. H. Miles. 1984. *A Blackboard System for Multi-Sensor Fusion.* Tech. Rep., ASWE, Portsdown, Portsmouth, England.

Lesser, V. R., R. D. Fennell, L D. Erman, & D. Raj Reddy. 1974. "Organization of the HEARSAY-II Speech Understanding System." In *IEEE Symposium on Speech Recognition, Contributed Papers.* Computer Science Department, Carnegie-Mellon University: IEEE Group on Acoustics, Speech and Signal Processing, 11-M2–21-M2.

Lesser, V. R., & L. D. Erman. 1977. "The Retrospective View of the HEARSAY-II Architecture." *Proceedings of the 5th International Joint Conference on Artificial Intelligence.* Los Altos, California: William Kaufmann, Inc., 790–800.

Lesser, V. R., & D. D. Corkill. 1983. "The Distributed Vehicle Monitoring Testbed: A Tool for Investigation Distributed Problem Solving Networks." *AI Magazine* 4 (3): 15–33.

Lindsay, R., B. G. Buchanan, E. A. Feigenbaum, & J. Lederberg. 1980. *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project.* McGraw-Hill, New York.

McCune, Brian P., & R. J. Drazovich. 1983. Radar with Sight and Knowledge. *Defense Electronics.* August.

Nagao M., T. Matsuyama, & H. Mori. 1979. Structured Analysis of Complex Photographs. *Proceedings of the 5th International Joint Conference on Artificial Intelligence.* 790–800.

Nagao M., & T. Matsuyama. 1980. *A Structural Analysis of Complex Aerial Photographs.* Plenum Press, New York.

Newell, A., 1969. Heuristic Programming: Ill-Structured Problems. In J. Aronofsky (editor), *Progress in Operations Research.* New York: John Wiley, 360-414.

Newell, A., & H. A. Simon. 1972. *Human Problem Solving.* Prentice Hall, Englewood Cliffs, N.J.

Newell, A., J. Barnett, C. Green, D. Klatt, J. C. R. Licklider, J. Munson, R. Reddy, & W. Woods. 1973. *Speech Understanding System: A Final Report of a Study Group.* North-Holland.

Nii, H. P., & E. A. Feigenbaum. 1978. "Rule-Based Understanding of Signals." In D. A. Waterman and R. Hayes-Roth eds., *Pattern-Directed Inference Systems.* New York: Academic Press, 483–501.

Nii, H. Penny., & N. Aiello. 1979. "AGE: A Knowledge-based Program for Building Knowledge-based Programs." *Proceedings of the 6th International Joint Conference on Artificial Intelligence.* Los Altos, California: William Kaufmann, Inc., 645–655.

Nii, H. P., E. A. Feigenbaum, J. J. Anton, & A. J. Rockmore. 1982. "Signal-to-Symbol Transformation: HASP/SIAP Case Study." *AI Magazine* 3 (2): 23–35.

Nii, H. P., 1986. *CAGE and POLIGON: Two Frameworks for Blackboard-Based Concurrent Problem Solving.* Tech. Rep. KSL-86-41, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Reddy, D. R., L. D. Erman, & R. B. Neely. June, 1973. "A Model and a System for Machine Recognition of Speech." *IEEE Transactions on Audio and Electroacoustics* AU-21: 229–238.

Reddy, D. R., L. D. Erman, & R. B. Neely. 1973. "The HEARSAY Speech Understanding System: An Example of the Recognition Process." *Proceedings of the 3rd International Joint Conference on Artificial Intelligence.* Los Altos, California: William Kaufmann, Inc., 185–193.

Rice, J. 1986. *Poligon: A System for Parallel Problem Solving.* Tech. Rep. KSL-86-19, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Shafer, S. A., A. Stentz, & C. Thorpe. 1986. "An Architecture for Sensor Fusion in a Mobile Robot." *Proceedings of the 1986 IEEE International Conference on Robotics and Automation.* (Forthcoming.)

Sheil, B. 1983. "Power Tools for Programmers." *Datamation*, February 1983, 131–144.

Simon, Herbert A. 1969. *The Sciences of the Artificial.* Cambridge, Massachusetts: MIT Press.

Simon, Herbert A. 1977 "Scientific Discovery and the Psychology of Problem Solving." In *Models of Discovery.* Boston, Massachusetts: D. Reidel Publishing Company.

Spain, D. S. 1983. "Application of Artificial Intelligence to Tactical Situation Assessment." *Proceedings of the 16th EASCON 83,* 457–464.

Terry, A. 1983. *The CRYSALIS Project: Hierarchical Control of Production Systems.* Tech. Rep. HPP-83-19, Stanford University, Heuristic Programming Project.

Waterman, D. A. 1985. *A Guide to Expert Systems.* Reading, Massachusetts: Addison-Wesley.

Weiss, S., & C. Kulikowski. 1984. *A Practical Guide to Building Expert Systems.* New Jersey: Rowman & Allanheld.

Williams M., H. Brown, & T. Barnes. May, 1984. *TRICERO Design Description.* Tech. Rep. ESL-NS539, ESL, Inc.

Williams, M. A. 1985. "Distributed, Cooperating Expert Systems for Signal Understanding." In *Proceedings of Seminar on AI Applications to Battlefield,* 3.4-1–3.4-6